

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Tóth Csaba

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>

DRAFT

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert ÁCs Tóth, Csaba	2019. november 16.	

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
0.0.5	2019-05-09	Minden feladat kidolgozva.	Tóth Csaba

Ajánlás

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

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

DRAFT

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	7
2.3. Változók értékének felcserélése	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	13
2.6. Helló, Google!	16
2.7. 100 éves a Brun téTEL	19
2.8. A Monty Hall probléma	21
3. Helló, Chomsky!	24
3.1. Decimálisból unárisba átváltó Turing gép	24
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	26
3.3. Hivatalos nyelv	27
3.4. Saját lexikális elemző	29
3.5. l33t.l	30
3.6. A források olvasása	34
3.7. Logikus	35
3.8. Deklaráció	36

4. Helló, Caesar!	40
4.1. double ** háromszögmátrix	40
4.2. C EXOR titkosító	42
4.3. Java EXOR titkosító	43
4.4. C EXOR törő	45
4.5. Neurális OR, AND és EXOR kapu	48
4.6. Hiba-visszaterjesztéses perceptron	49
5. Helló, Mandelbrot!	52
5.1. A Mandelbrot halmaz	52
5.2. A Mandelbrot halmaz a std::complex osztállyal	55
5.3. Biomorfok	59
5.4. A Mandelbrot halmaz CUDA megvalósítása	61
5.5. Mandelbrot nagyító és utazó C++ nyelven	63
5.6. Mandelbrot nagyító és utazó Java nyelven	68
6. Helló, Welch!	75
6.1. Első osztályom	75
6.2. LZW	78
6.3. Fabejárás	81
6.4. Tag a gyökér	83
6.5. Mutató a gyökér	91
6.6. Mozgató szemantika	91
7. Helló, Conway!	94
7.1. Hangyszimulációk	94
7.2. Qt C++ életjáték	97
7.3. Java életjáték	99
7.4. BrainB Benchmark	109
8. Helló, Schwarzenegger!	112
8.1. Szoftmax Py MNIST	112
8.2. Mély MNIST	116
8.3. Minecraft-MALMÖ	116

9. Helló, Chaitin!	117
9.1. Iteratív és rekurzív faktoriális Lisp-ben	117
9.2. Gimp Scheme Script-fu: króm effekt	117
9.3. Gimp Scheme Script-fu: név mandala	122
10. Helló, Gutenberg!	129
10.1. Programozási alapfogalmak	129
10.2. Programozás bevezetés	131
10.3. Programozás	132
11. Összegzés:	135
11.1. Passzolt feladatok:	135
11.2. Tutoráltjaim:	135
11.3. Tutorok:	136
11.4. Licensz	136
III. Második felvonás	137
12. Helló, Arroway!	139
12.1. OO szemlélet	139
12.2. "Gagyi"	142
12.3. Yoda	144
12.4. Kódolás from scratch	145
13. Helló, Berners-Lee!	148
13.1. Python: Bevezetés a mobilprogramozásba	148
13.2. Java: Java 2 útikalauz 5	149
14. Helló, Liskov!	151
14.1. Liskov helyettesítés sértése	151
14.2. Szülő-gyerek	153
14.3. Ciklomatikus komplexitás	154
14.4. Anti OO	155
15. Helló, Mandelbrot!	166
15.1. Reverse engineering UML osztálydiagram	166
15.2. Forward engineering UML osztálydiagram	167
15.3. BPMN	170

16. Helló, Chomsky!	172
16.1. Encoding	172
16.2. Saját Leet Cipher	174
16.3. Full Screen	178
17. Helló, Stroustrup!	181
17.1. JDK Osztályok	181
17.2. Másoló-mozgató szemantika és Összegzés	183
18. Helló, Gödel!	188
18.1. Gengszterek	188
18.2. Alternatív tabella rendezése	189
18.3. Gimp Scheme hack	191
19. Helló,!	194
19.1. FUTURE tevékenység editor	194
19.2. SamuCam	196
19.3. BrainB Slot-signal mechanizmus	199
20. Helló,Lauda!	201
20.1. Port Scan	201
20.2. AOP	202
20.3. JUnit teszt	204
21. Helló,Calvin!	206
21.1. MNIST	206
21.2. TensorFlow objektum detektáló	207
IV. Irodalomjegyzék	210
21.3. Általános	211
21.4. C	211
21.5. C++	211
21.6. Lisp	211

Ábrák jegyzéke

2.1.	1 szálat 100%-on használó ciklus	6
2.2.	1 szálat 0%-on használó ciklus	6
2.3.	Minden szálat 100%-on használó ciklus	7
2.4.	Linkmátrix	18
2.5.	Grafikus kimenet	21
2.6.	A szimuláció 100 kísérletre	23
3.1.	A gép felépítése	25
3.2.	Az állapotátmenet gráfja	26
3.3.	A C89-es fordítással kapott hibaüzenetek	29
3.4.	A program kimenete	30
3.5.	A program kimenete	34
4.1.	A program futása után kapott háromszögmátrix	42
4.2.	Titkosítás előtti és utáni szöveg	43
5.1.	Egy Mandelbrot halmaz	55
5.2.	A program egy féle kimenete	59
5.3.	Egy a program futtatásával kapott biomorf	61
5.4.	Egy 4-szeres nagyítás folyamata	73
5.5.	Egy 13-szoros nagyítás	74
6.1.	A program kimenete	78
6.2.	A program kimenete	81
6.3.	Inorder,postorder és postorder bejárások kimenetei	83
6.4.	A program kimenete	93
7.1.	Hangya	95
7.2.	Sikló alakzat	97

7.3. Egy pillanatkép	101
8.1. LVL9. Android Emulatorral Windowson	116
9.1. A program kipróbálása	122
9.2. A program kipróbálása	128
11.1. Lvl. 9	135
12.1. JDK forrás	143
12.2. JDK forrás:	144
12.3. A formula (forrás: https://en.wikipedia.org/wiki/Bailey%E2%80%93Borwein%E2%80%93Plouffe_formula)	145
14.1. A programok hiba nélkül futnak, de a Liskov elvet sértik	152
16.1. A program futáskor készített screenshot	174
18.1. sort metódus JDK leírás	190
18.2. A program futása a 2019/20-as magyar bajnokság eddigi állása alapján	191
19.1. A bug	195
19.2. Bug javítva	196
19.3. Felélesztve	200
20.1. Socket dokumentáció	201
20.2. Futás:	202
20.3. Futtatás	203
20.4. Az eredeti inorder és az AspectJ-s postorder kiiratás	204
20.5. Sikeres teszt	205
21.1. A program sikeresen felismeri a saját 8-ast	207

Táblázatok jegyzéke

14.1. Mérési eredmények táblázatba gyűjtve: 155

DRAFT

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

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 csemegeznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

1 szálat 100%-on használó ciklus: https://gitlab.com/tocsika7/prog1/blob/master/Turing/Vegtelen_ciklus/-vegtelen_100.c

1 szálat 0%-on használó ciklus: https://gitlab.com/tocsika7/prog1/blob/master/Turing/Vegtelen_ciklus/vegtelen_0.c

Minden szálat 100%-on használó ciklus: https://gitlab.com/tocsika7/prog1/blob/master/Turing/Vegtelen_ciklus/-vegtelen_multi.c

Tanulságok, tapasztalatok, magyarázat...

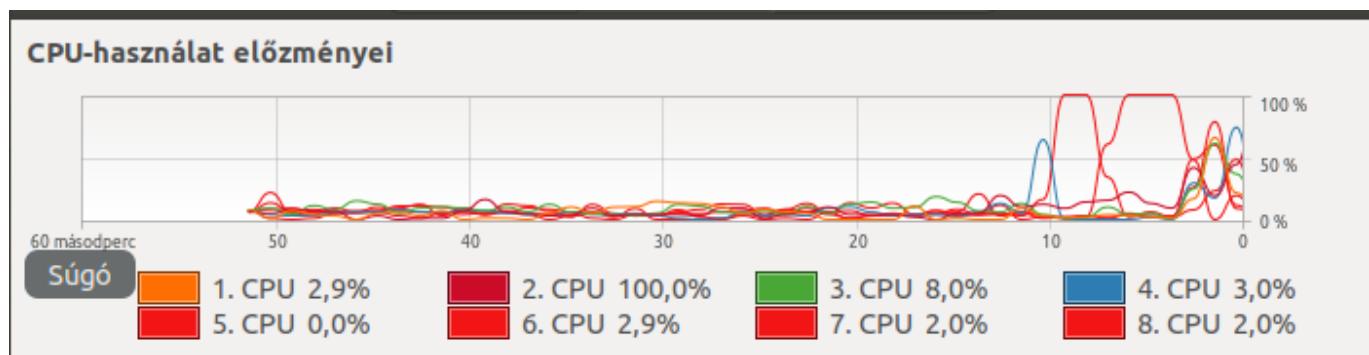
Annak egy módja, hogy végtelen ciklust írunk C-ben az, ha egy while ciklusnak 1-et adunk feltételnek. Ekkor ugyanis minden igaz. Ez a ciklus egy processzor szálat használ 100%-on. A while(1) ciklus tetszőlegesen helyettesíthető egy for(;;) ciklussal.

```
#include <stdio.h>

int main() {

    while(1) {}

    return 0;
}
```



2.1. ábra. 1 szálat 100%-on használó ciklus

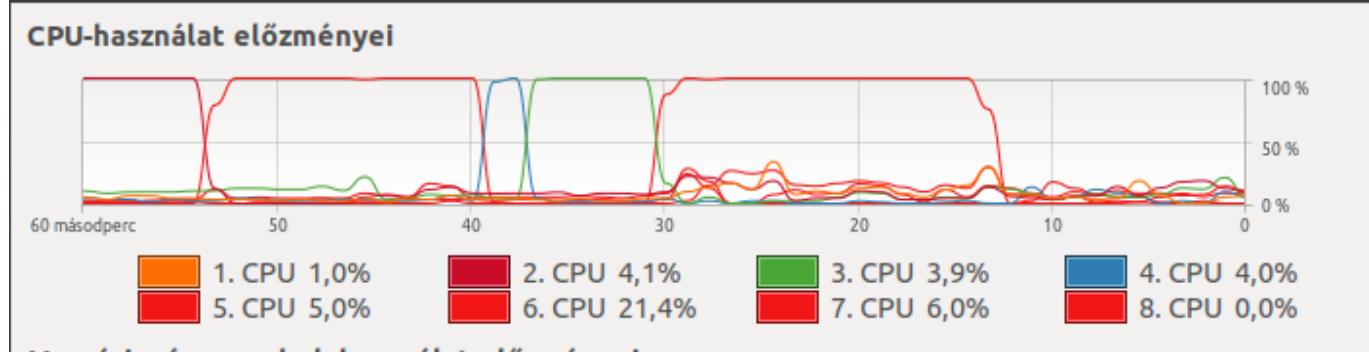
Ahhoz, hogy 0%-on használjuk a processzor egy szálát, a sleep() függvényt kell használnunk. Ehhez az programban include-olni kell az unistd.h headert. A sleep() függvény a zárójelben megadott másodpercek leteltéig pihenteti a ciklust, tehát addig nem használja a processzor adott szálát.

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

int main() {

    while(1)
    {
        sleep(1);
    }

    return 0;
}
```



2.2. ábra. 1 szálat 0%-on használó ciklus

Ahhoz, hogy egy ciklussal több szálat párhuzamosan használjunk egy lehetséges megoldás az OpenMP. A ciklus elé írt #pragma omp parallel és #pragma omp for (vagy for, ciklustól függően) parancsok szét-

osztják a processzor aktív szálai között a while ciklus parancsait. Ha ezt a megoldást használjuk a gcc vegtelen_multi.c -o output_nev -fopenmp módon fordítsuk a programot.

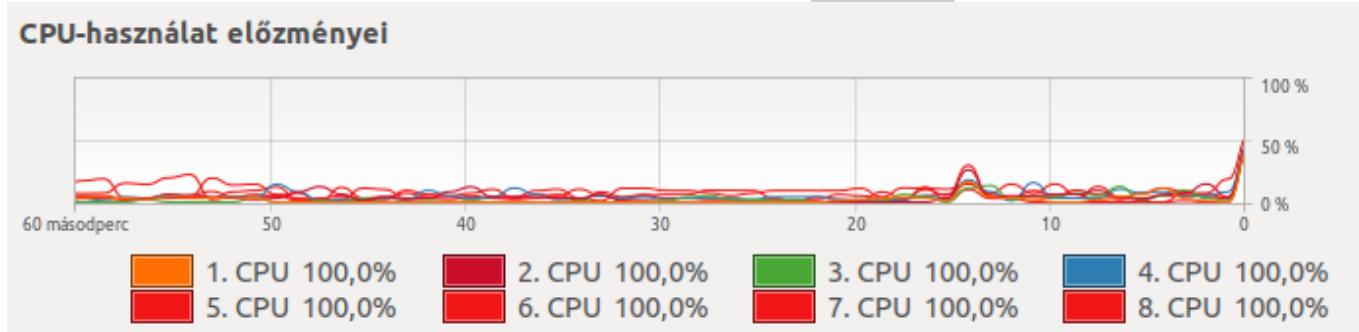
```
#include <stdio.h>
#include <unistd.h>

int main() {

#pragma omp parallel
#pragma omp while

while(1) { }

}
```



2.3. ábra. minden szálat 100%-on használó ciklus

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

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

Megoldás videó:

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

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

```
        return false;
    }

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Megoldás változók különbségével: https://gitlab.com/tocsika7/prog1/blob/master/Turing/Valtozo_csere/kulonbse

Megoldás változók szorzatával: https://gitlab.com/tocsika7/prog1/blob/master/Turing/Valtozo_csere/szorzas.c

Tanulságok, tapasztalatok, magyarázat...

Változók értékének felcserélését többféleképpen végezhetjük el logikai utasítás, és segédváltozó használata nélkül.

1. A két változó különbségének használatával

```
#include <stdio.h>

int main(){
    int a=5;
    int b=23;

    printf("a=%i\n",a);
    printf("b=%i\n",b);

    b = b - a;           // b=23-5=18
    a = a + b;           // a=5+18=23
    b = a - b;           // b=23-18=5
    printf("a=%i\n",a); //a=23
    printf("b=%i\n",b); //b=5

    return 0;
}
```

2. A két változó szorzatának használatával

```
#include <stdio.h>

int main(){
    int a=5;
    int b=23;

    printf("a=%i\n",a);
    printf("b=%i\n",b);

    printf("Szorzatos csere\n");
    b = b * a;                                //b=23*5=115
    a = b / a;                                //a=115/5=23
    b = b / a;                                //b=115/23=5
    printf("a=%i\n",a);                          //a=23
    printf("b=%i\n",b);                          //b=5

    return 0;
}
```

2.4. Labdapattogás

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

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

Megoldás forrása:

Megoldás if-ekkel: <https://bhaxor.blog.hu/2018/08/28/labdapattogas>

Megoldás if-ek használata nélkül: <https://gitlab.com/tocsika7/prog1/blob/master/Turing/Labdapattogas/ifnelkul.c>

Tanulságok, tapasztalatok, magyarázat...

1. Megoldás: if-ekkel:

A kód forrása:

A include-oljuk a curses.h header-t, erre az initscr() és mvprintw() függvények használata miatt van szükség.

//A kód forrása: <https://bhaxor.blog.hu/2018/08/28/labdapattogas>

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

```
int  
main ( void )  
{
```

Az initscr() függvény lekéri a terminál méretét és betölti az *ablak nevű változóba.

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

Deklaráljuk az x és y változókat, amelyek jelölik majd az x és y tengelyt, amin a labda haladni fog. Emellett deklaráljuk az xnov és ynov változókat, amikben azt tároljuk, hányat "lép" egyszerre a labda x és y irányba. Az mx és my változókba a terminál méretét tehát az x és y tengely végeit töltjük be a getmaxyx() függvény segítségével, az ablak változóból.

```
int x = 0;  
int y = 0;  
  
int xnov = 1;  
int ynov = 1;  
int mx;  
int my;  
  
for ( ; ; )  
{  
  
    getmaxyx ( ablak, my , mx );
```

Az mvprintw() egy printf()-hez hasonló függvény. Ezzel íratjuk ki a labda aktuális helyzetét az x és y tengelyeken. A refresh() függvénytel a terminált tudjuk frissíteni, míg a usleep() a labda gyorsaságát állítja. A usleep() a zárójelben megadott értéknyi mikroszekundumig felfüggeszti a ciklust. Minnél kevesebb az érték annál gyorsabban pattog a labda.

```
mvprintw ( y, x, "0" );  
  
refresh ();  
usleep ( 100000 );  
clear();
```

"Léptetjük" a labdát a korábban megadott értékekkel.

```
x = x + xnov;  
y = y + ynov;
```

Ha a labda elérte a terminál szélét, megsorozzuk -1-gyel a növekedés értékét, tehát elindul az ellenkező irányba.

```
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;
}

}

return 0;
}
```

2. Megoldás: if-ek használata nélkül:

Az if nélküli megoldáshoz szükség van két tömbre amelyeknek annyi eleme van, amilyen hosszú, illetve széles a terminál. Az if-es verzióban csak akkor kellett szorozni a labda irányát (-1-el), ha elérte a széleket. Ebben az esetben viszont ezt nem tudjuk ellenőrizni, ezért minden lépésnél szoroznunk kell 1-gyel. Ha a széleket éri el a labda akkor pedig -1-gyel kell szorozni. A szélek helyzetét a tömbök segítségével lehet meghatározni. A tömbök 0-dik és utolsó elemei a terminál szélei.

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

//Fordításhoz -lncurses kacsoló

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

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;
    getmaxyx ( ablak, my , mx );
```

```
int height[my];
int i=0;
for(;i<mx;i++) {
    width[i]=1;
}
i=0;
for(;i<my;i++) {
height[i]=1;
}
width[0]=-1;
height[0]=-1;
width[mx-1]=-1;
height[my-1]=-1;

for ( ; ; ) {

    mvprintw ( y, x, "0" );

    refresh () ; // 
    usleep ( 100000 );

    x = x + xnov;
    y = y + ynov;

    xnov*=width[x];
    ynov*=height[y];
}

return 0;
}
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

BogoMIPS:

https://gitlab.com/tocsika7/prog1/blob/master/Turing/Szohossz_BogoMips/bogomips.c

Szóhossz:

https://gitlab.com/tocsika7/prog1/blob/master/Turing/Szohossz_BogoMips/szohossz.c

Tanulságok, tapasztalatok, magyarázat...

A BogoMips-et Linus Torvalds a Linux fő fejlesztője írta. A a processzor gyorsaságát lehet vele megmérni.

```
// BHAX BogoMIPS
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// This program is based on
//
// - Linus Torvalds's original code (https://mirrors.edge.kernel.org/pub/linux/kernel/v1.0/linux-1.0.tar.gz init/main.c)
// - and Jeff Tranter's standalone version (archive.debian.org/debian/pool/main/s/sysutils/sysutils\_1.3.8.5.1.tar.gz).
//
// See also UDPORG
//

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

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

int
main (void)
{
    unsigned long long loops_per_sec = 1;
    unsigned long long ticks;

    printf ("Calibrating delay loop..");
    fflush (stdout);
```

A while ciklus bitshift operátorral folyamatosan eggyel balra tolja a loops_per_sec változó értékét, amivel 2 hatványokat képez. A ticks változónak értékül adjuk a clock() függvényt, ami visszaadja egy feladathoz felhasznált processzoridőt. Ezután meghívjuk a delay függvényt az adott 2 hatványra ami csak annyit fog tenni, hogy egy for ciklusban 1-től elmegy az adott hatványig. Az ehhez szükséges időt tárolja a ticks változó.

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

Ez a ciklus addig halad, amíg ticks-ek száma meg nem haladja CLOCK_PER_SEC-et. Ennek az értéke egy milliós nagyságrendű szám, ami az eredeti BogoMips-ben volt meghatározva. Ezután kiírja azt a loop értéket ahol a CLOK_PER_SEC-et adná a ciklus.

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

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

    return 0;
}

printf ("failed\n");
return -1;
}
```

A szóhossz megállapításra is ezt a BogoMIPS-ben használt bitshift-et használjuk. A segédváltozó azt számolja hány bitshifteléssel lesz 1-ből 0.

```
#include <stdio.h>

int main()
{
    int hossz=1;
    int seged=0;

    while (hossz!=0) {
        hossz<<=1;
        seged++;
        printf ("%d \n", hossz);
```

```
    }
    printf("A szóhossz a gépen ");
    printf("%d bites \n", seged);
    return 0;
}
```

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

<https://gitlab.com/tocsika7/prog1/blob/master/Turing/PageRank/pagerank.c>

Tanulságok, tapasztalatok, magyarázat...

A PagRank algoritmus az oldalakat minőségük szerint rendezи sorba, azok az oldalak kerülnek előre, amelyekre a jó minőségű lapok mutatnak.

A programhoz szükség lesz A math.h header-re az sqrt() függvény miatt.

```
//A kód forrása: https://progpter.blog.hu/2011/02/13/ ←
bearazzuk_a_masodik_labort

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

A kiir függvénytel az adott oldalak PageRank értékét fogjuk kiírni. Ezt az értéket tömbökben táruljuk, amit átadunk paraméterként a kiir függvénynek, emellett átadjuk még az oldalak számát a darab paraméterben.

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

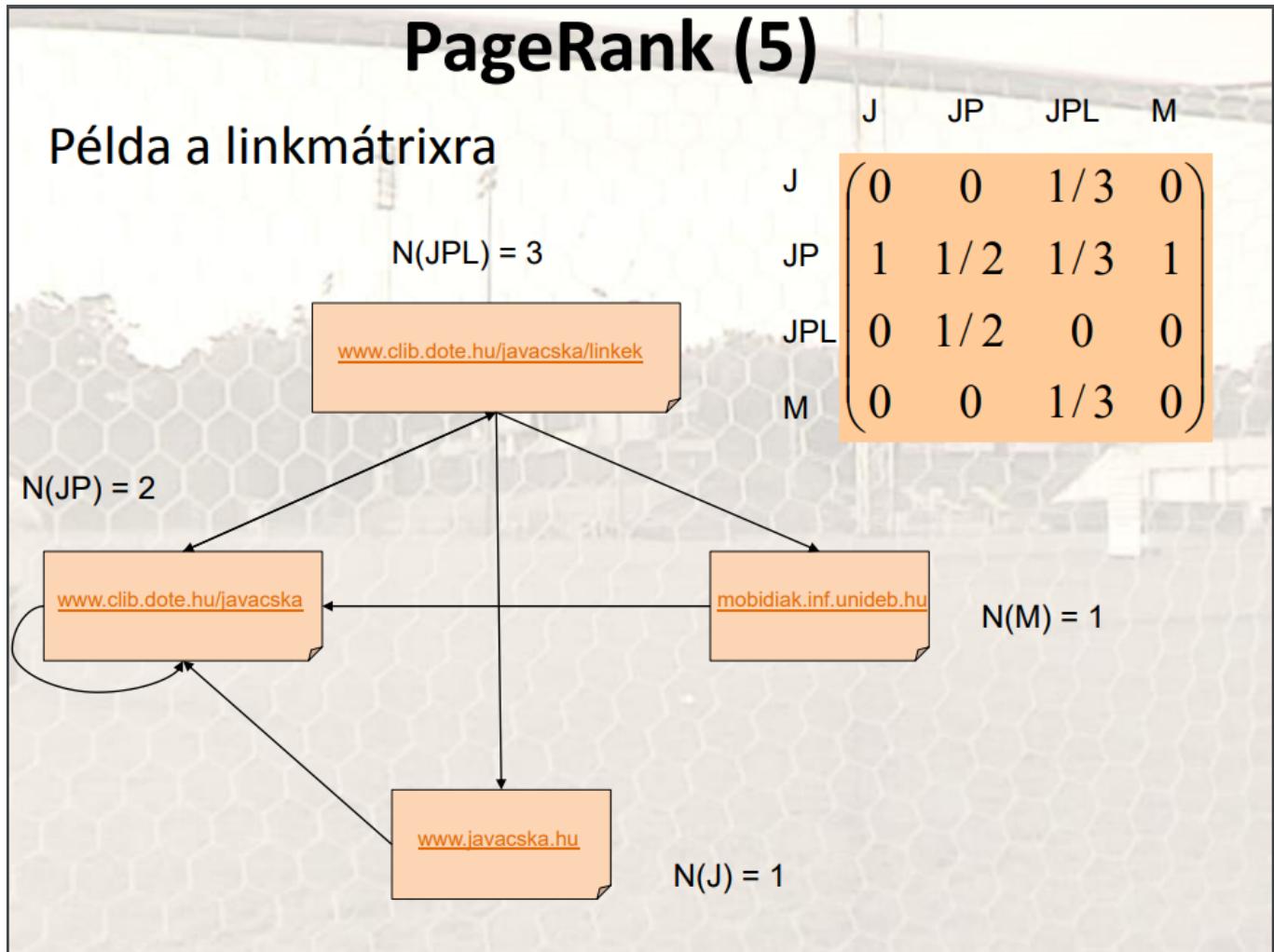
double tavolsag (double PR[], double PRv[], int n)
```

```
double sum=0.0;
for(int i=0;i<n;++i)
{
    sum+= (PR[i]-PRv[i])*(PR[i]-PRv[i]);
}
return sqrt (sum);
}
```

A main függvényben létrehozzuk az oldalak linkmátrixát. Úgy számoljuk ki az egyes oldalakhoz (oszlophoz) tartozó értéket, hogy megnézzük mutat-e sorra (másik oldalra), majd elosztjuk a kimenő linkek számával.

pl. Második oszlopban a JP oldal sorában:

- A JP két oldalra mutat: önmagára és JPL re, tehát minden értéket 2-vel kell osztani.
- J-re nem mutat a JP tehát az értéke $0/2 = 0$
- JP-re mutat JP tehát az értéke $1/2$
- JPL-re mutat JP tehát az értéke $1/2$
- M-re nem mutat a JP tehát az értéke $0/2$



2.4. ábra. Linkmátrix

A kép forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_2.pdf?fbclid=IwAR1tnKx5YZkAjp48d-LGhTnygZsgx0

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

A PR tömbben lesz a PageRank, A PRv tömbben pedig a következő PageRank lesz, amit az előzőből számolunk. Egy oldal PageRank-ját úgy számoljuk ki, hogy végigmegyünk a hozzáartozó soron az L linkmátrixban és minden elemet megszorozzuk a PageRank-jával, ami az első futásnál mindegyiknek 1/4. Ezután ezeket összeadjuk. Ezt követően kiszámoljuk a PRv-ben is a következőt a PR alapján. Ezt addig

csináljuk amíg nem kapjuk meg a rendes PageRank-ot. Ezt a távolság függvény ellenőrzi, ha a kevés a két PageRank távolsága készen van.

```
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};

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

    if(tavolsag(PR, PRv, 4) < 0.000001)
    {
        break;
    }
}

kiir (PR, 4);
return 0;
}
```

2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

Brun tétele Viggo Brun nevéhez fűződik. Azt mondja ki, hogy az ikerprímek (olyan prímek amelyeknek különbsége 2) reciprokának összege egy megadott értékhez, a Brun-konstanshoz tart. Azt, hogy az ikerprímek száma véges vagy végtelen a mai napig sem lett bebizonyítva.

A fentebb említett gondolatok forrása: <https://hu.wikipedia.org/wiki/Brun-t%C3%A9tel>

Ahhoz, hogy működjön a program telepítenünk kell, majd meg kell hívunk R-hez a matlab csomagot

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

```
install.packages("matlab")
library(matlab)
```

A későbbi könnyű alkalmazás érdekében az egész programot (a kirajzoláson kívül) az stp függvénybe fogjuk írni. A primes vektorba a primes() függvény segítségével töltünk be prímeket. A primes() a zárójelben megadott értékig generálja a prímszámokat. Ezt követően a diff vektorba kerülnek bele az egymást követő prímek különbségei. Ez úgy történik, hogy kivonjuk a primes vektor 2.-tól utolsó elemeiből a primes vektor 1.-től utolsó előtti elemeit.

```
stp <- function(x)
{
  primes = primes(x)
  diff = primes[2:length(primes)]-primes[1:length(primes)-1]
```

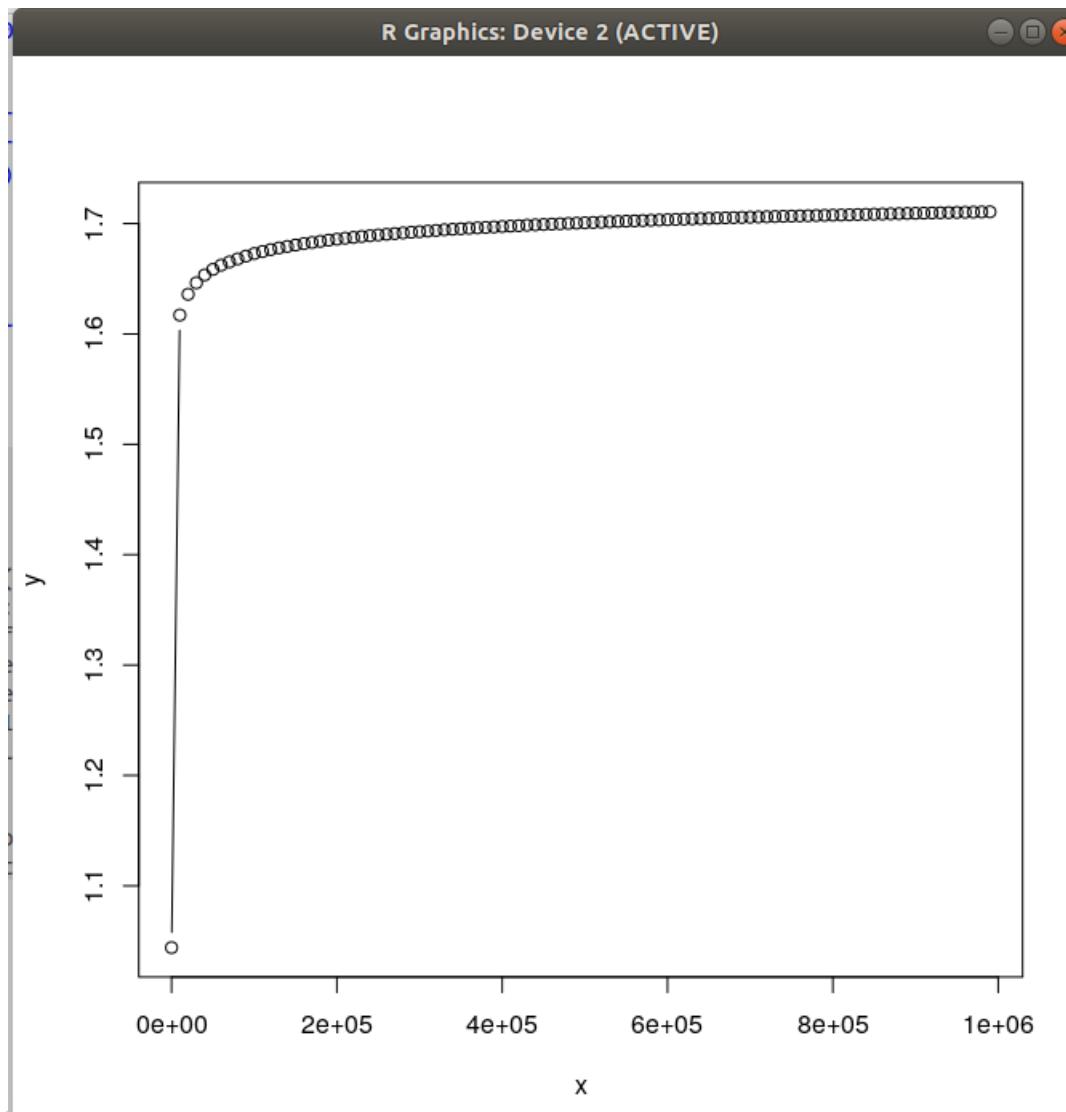
Ezt követően azt vizsgáljuk hol találhatóak az ikerprímek. Az idx vektorba diff vektor azon elem helyeit töltjük be, ahol a különbség 2 tehát a két prím ikerprím. Ezt a which függvénnyel tudjuk egyszerűen megtenni. A t1primes vektorba az ikerprímek első tagját töljük be a idx változó értékét használva míg a t2primes-ba ezzel egy 2-vel nagyobb értékeket tehát az ikerprímek 2. felét. Az rtlplust2 vektorba ezeknek az ikerpároknak vesszük a reciprokösszegét, majd ezeknek a reciprokösszegeknek a szummáját adjuk a stp függvény visszatérési értékének.

```
idx = which(diff==2)
t1primes = primes[idx]
t2primes = primes[idx]+2
rtlplust2 = 1/t1primes+1
return(sum(rtlplust2))
}
```

Az R grafikus megjelenítésre is lehetőséget ad, a következő parancsokkal tudjuk kirajzolni az adott ikerprímek reciprokösszegének az összegét és ellenőrizni, hogy ténylegesen a Brun-konstanshoz tart.

```
x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
```

```
plot(x,y,type="b")
```



2.5. ábra. Grafikus kimenet

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

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall problémát egy amerikai vetékedő műsörvezetőjéről neveztek el. A vetélkedő utolsó fordulójában a mutattak a 3 ajtót a játékosnak, amelyből 2 mögött értéktelen nyeremény volt és 1 mögött értékes. A

játékos kiválaszt egy ajtót majd mielőtt kinyílik, a műsorvezető is kinyit egyet előtte, ami mögött értéktelen nyeremény van (ő tudja melyik ajtó mögött mi van) és felteszi a játékosnak a kérdést szeretne-e cserálni. A kérdés az, hogy érdemes-e változtatni vagy nem. Erre fog egy megoldás adni a program.

A fentebb említett gondolatok forrása: https://hu.wikipedia.org/wiki/Monty_Hall-paradoxon

A kiserletek_szama változóban deklaráljuk hány szimulációt szertnénk futtatni. A kísérlet vektorba az értékes nyeremények helye, míg a játékos vektorba a játékos tippje kerül teljesen véletlenszerűen.

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

Egy for ciklusban végigmegyünk a kísérleteken (kísérlet és játékos vektorán) majd if-el megnézzük elsőre eltalálta-e a játékos a nyereményt. Ha igen a műsorvezető 2 érték közül választhat, mivel kivesszük a lehetőségei közül a játékos által választottat. Ha a játékos nem találja el egyből jól a nyeremény helyét, a műsorvezető már csak 1 értéket választhat, hiszen nem válaszhatja a játékos tippjét, illetve az ajtót ami mögött a nyeremény van. A műsorvezető választásainak lehetőségeit minden esetben a mibol vektorban tároljuk és ebből a vektorból fog kikerülni a műsorvezető tippje.

```
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)]
}
```

A program utolsó részében vektorokat deklarálunk. Ezek a vektorok(nemvaltoztatesnyer, valtoztatesnyer) azt tárolják hányszor nyert a játékos a kísérletek alatt ha: nem változtatott, vagy változtatott.

```
nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)
for (i in 1:kiserletek_szama)
{
  holvált = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvált[sample(1:length(holvált),1)]
}
valtoztatesnyer = which(kiserlet==valtoztat)

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

A szimulációk azt az eredményt adják, hogy a játékosnak megéri a váltás. Átlalában 2/3 arányban nyer ha változtat a döntésén. A program R nyelven van írva, ha futtatni szertnénk (Linuxon) le kell töltenünk az R-t a szoftver áruházból. Ezután terminálban R parancsot kell adni és a megjelenő felületen bemásoljuk az adott program forráskódját.

```
[1] "Kiserletek szama: 100"
> length(nemvaltoztatesnyer)
[1] 27
> length(valtoztatesnyer)
[1] 73
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.369863
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 100
```

2.6. ábra. A szimuláció 100 kísérletre

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Tutor: Borvíz Róbert https://github.com/BorvizRobi/prog_1_textbook/tree/master/beadando

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Chomsky/Turing-atmenet.png>

Tanulságok, tapasztalatok, magyarázat...

A Turing-gép Alan Turing angol matematikus nevéhez fűződik. Matematikai számolások elvégzésére alkalmas, a mai számítógépek egy leegyszerűsített elődje.

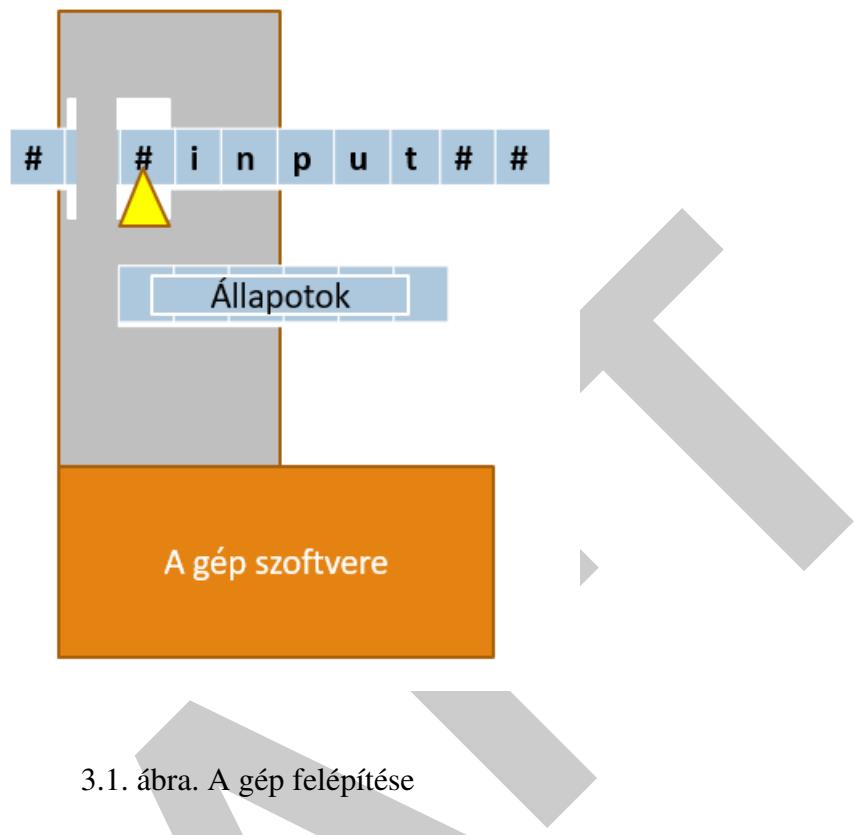
Részei:

- a memóriája egy felosztott végtelen szalag
- vezérlőegység, amiben a program van
- író-olvasó fej, aminek feladata megadott szimbólumokkal való írás és olvasás

Működése:

A gép aktuális pozíciója azt jelöli, hogy az aktuális cella hol van a memóriasalagon. A gépnek van egy aktuális állapota is, amivel a gépet programozni fogjuk. A gép lépésenként szimbólumokat olvas a memóriáról és a szimbólumtól és az állapotától függően 3 lépés mehet végbe:

- szimbólumot ír az aktuális cellába
- a memóriasalagon az olvasófej lép: balra, jobbra vagy marad ugyanott
- a vezérlőegység átváltja a gép állapotát



3.1. ábra. A gép felépítése

Egyes számrendszerbe való váltás esetén annyi egyest írunk, amekkora decimális szám értéke. Az állapotátmenet gráfjával ez úgy történik, hogy ha 1 értékű szimbólumot találunk, leírjuk annyiszor, mint amekkora a decimális szám.

Állapotok:

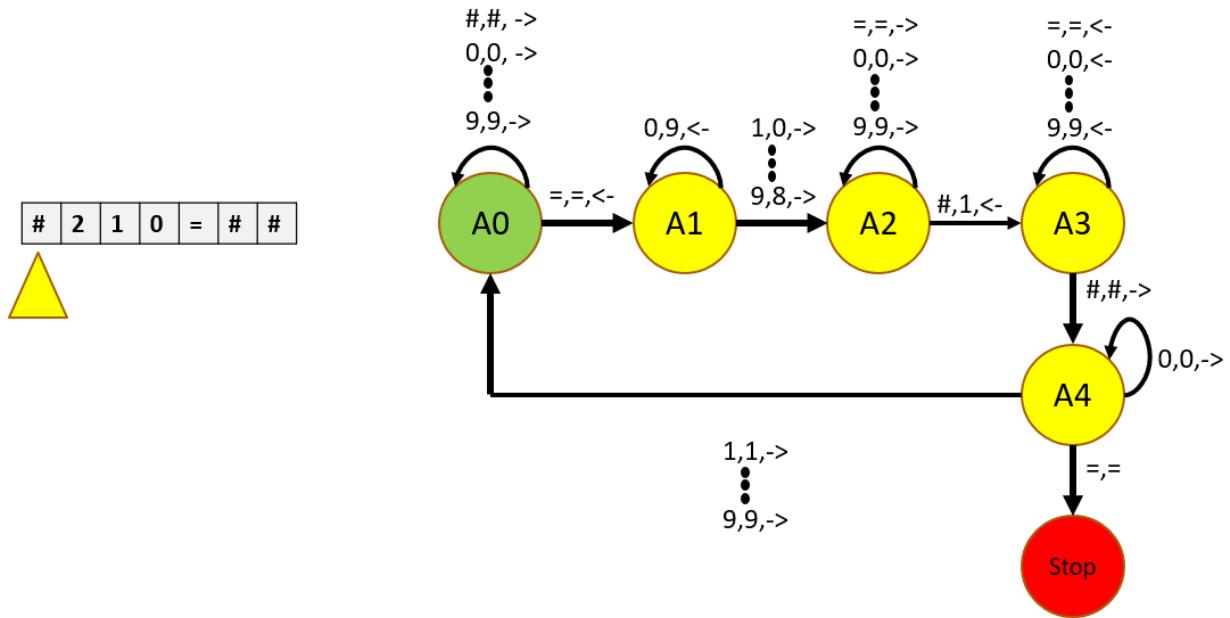
A0: A gép kezdő állapota. Ha a belolvasott szimbólum #, azaz üres akkor ugyanazt írjuk vissza, számok esetén is így járunk el. Ha = -et olvasunk akkor is visszaírjuk a =-et de lépünk egyet balra a memóriaszalagon és átállunk a következő állapotra.

A1: Ha a beolvasott szám 0, 9-est írunk, és maradunk az állapotban. minden más szám esetén eggyel kevesebbet írunk, jobbra lépünk a memóriaszalagon és átállunk a következő állapotra.

A2: Ha az olvasott szimbólum # tehát üres, akkor 1-est írunk, balra lépünk a szalagon és a következő állapotba lépünk. minden más esetben azt írjuk be, amit olvasunk, jobbra haladunk és nem változtatjuk az állapotot.

A3: Ha üres helyet találunk a szalagon beírunk egy #-et és jobbra lépünk állapotváltással. Egyébként azt írjuk amit olvasunk maradunk az állapotban és balra haladunk.

A4: Ha 0-t olvasunk 0-t írunk, haladunk jobbra és nem változtatunk az állapoton. Ha ilyenkor olvasunk be =-et akkor már az összes helyiérték kinullázódunk és a stop állapotra haladunk, amivel megállítjuk a gépet. Ha más számot kapunk akkor leírjuk, haladunk jobbra és visszatérünk a kezdeti állapotra és program újrakezdődik.



3.2. ábra. Az állapotátmenet gráfja

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Noam Chomsky amerikai nyelvész, a Massachusettsi Műszaki Egyetem (MIT) professzora, az Ő nevéhez fűződnek a Chomsky-féle nyelvosztályok. A Chomsky-féle hierarchia 4 típusra osztja a generatív nyelveket.

- 0-s típusú: átláthatós vagy mondatszerkezetű
- 1-es típusú: környezetfüggő
- 2-es típusú: környezetfüggetlen
- 3-as típusú: reguláris

A környezetfüggetlen grammákat például a programozási nyelveket leíró szintaktikák, például a következő feladatban tárgyalta BNF.

3.3. Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiáld 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/tocsika7/prog1/blob/master/Chomsky/Hivatkozasi_nyelv/hivatkozas.c

Tanulságok, tapasztalatok, magyarázat...

"Utasítás fogalma: Az utasítások egymást követően, sorban hajtódnak végre, az ettől való eltérést külön jelezzük. Az utasítások sokfélék lehetnek." (KERNIGHANRITCHIE C könyv)

A BNF vagy Backus-Naur forma egy átlalános szintaxis amit általában programozási nyelvek leírására használunk, de akármit le lehet írni vele. Átalános alakjára egy példa:

```
<szimbólum> ::= <kifejezés a szimbólumra>
```

Az utasítások változozatai:

A kifejezes utasítások egyszerre csak egy kifejezést képesek végrehajtani. Alakjuk: kifejezés;

Az összetett utasításokban vagy másnéven blokkokban egyszerre több utasítás is elhelyezhető egy helyén. Alakjuk { deklarációlista utasításlista };

Feltételes utasítás: (if) utasítás: A gép mindenkor else-es és else nélküli változatban kiértékeli az if-ben lévő kifejezést. Ha a kifejezés értéke igaz végrehajtja az utasítást, ha nem nem csinál semmit. Az else-es változatban az else után következő utasítást hajta végre a hamis if esetén.

If BNF-ben:

```
<if-> ::= if <feltétel> <utasítás> |
           if <feltétel> then <utasítás> else <utasítás>
```

While és do utasítások: A while utáni utasítás addig ismétlődik amíg a while-ban lévő kifejezés igaz. A vizsgálat mindenkor az utasítás végrehajtása előtt történik. A do while utasításban ez pontosan fordítva zajlik. Az utasítás a vizsgálat előtt hajtódnak végre, tehát egyszer legalább mindenkor lefut.

While és do while utasítások BNF-ben:

```
//While utasítás

<while ciklus> ::= while ( <feltétel> ) <utasítás>

//Do while ciklus

<do utasítás> ::= do <utasítás> while ( <feltétel> ) ;
```

For utasítás: A for utasítás 3 részből áll: Az első kifejezés a ciklust inicializálja. A másodikban van a feltétel, a harmadikban általában ciklusváltozót növeljük. A for utasítás bármely részkifejezése elhagyható, ha mindenkor elhagyjuk végtelen ciklus lesz belőle.

For BNF-ben:

```
<for ciklus> ::= for ( <kifejezés> ; <kifejezés>; <kifejezés> ) <↔  
utasítás>
```

A switch utasítás: A switch hatására a megadott kifejezés értékétől függően az vezérlés valameilyik utasítás egyikére adódik át. A switch értéke minden egész érték kell, hogy legyen. A switch blokk-ba kerülnek a case-ek amelyiken a switch a kifejezés után végigmegy és egyezést keres. Ha talál azt a case-t hajtja végre amelyik egyezik.

A switch BNF-ben

```
<switch utasítás> ::= switch ( <kifejezés> ) <switch blokk>
```

Break és continue utasítások: a break hatására megszűnik a break-et körülvevő while, for vagy switch utasítása, continue hatására pedig folytatódik.

A break és countinue szintaxisa:

```
<break> ::= <break>  
<countinue> ::= <countinue>
```

Goto utasítás és címkézett utasítás :A goto átadja a vezérlést a címkézett utasításnak. A címkézett utasítás goto célpontjaként szolgál, címkéjének az érvényességi tartománya az a függvény, amelyben deklaráljuk.

Ezenkívül létezik még a nulla utasítás, aminek a szintaxisa egyszerűen ;. Ha a for-ban kihagyjuk az egyik részt akkor az is nulla utasítás.

Ez a kódcsipet C99-ben lefordul, viszont C89-el nem. A for-on kívüli deklaráció csak C99-el és C11-el engedélyezett.

```
#include <stdio.h>  
  
int main() {  
  
    int i;  
    for(i=0;i<5;i++)  
  
        return 0;}
```

A C++ típusú kommentek sem fordulnak le C89-ben.

```
#include <stdio.h>  
  
int main() {  
  
    //komment ami ne fordul le  
  
    /* komment ami lefordul */  
  
    return 0;}
```

Fordítás C89-el: gcc hivat.c -o hivat -std=c89

Fordítás C99-el: gcc hivat.c -o hivat -std=c99

```
user@ubuntu:~/Asztal/Prog1_programs/chomsky$ gcc hivat.c -o hivat -std=c89
hivat.c: In function 'main':
hivat.c:4:1: error: 'for' loop initial declarations are only allowed in C99 or C
11 mode
  for(int i=0;i<5;i++);
  ^
hivat.c:4:1: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 to co
mpile your code
hivat.c:6:1: error: C++ style comments are not allowed in ISO C90
 //komment ami nem fordul le c89-ben
 ^
```

3.3. ábra. A C89-es fordítással kapott hibaüzenetek

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/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.l

Tanulságok, tapasztalatok, magyarázat...

Ebben a programban lexer segítségével fogjuk olvasni a standard bemenetet és kiszűrjük a valós számokat. A realnumbers változóban a valós számok számát fogjuk tárolni. A lexer megadott token-eket keres majd a bemeneten, ez a token lesz a digit, aminek megadjuk [0-9] számjegy karaktercsokrot.

```
//A kód forrása: https://gitlab.com/nbatfai/bhax/blob/master/ ←
  thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/ ←
    realnumber.l

%{
#include <stdio.h>
int realnumbers = 0;
%
digit [0-9]
```

Ebben a programrészben adjuk meg, hogyan nézzen ki a digit token, tehát milyen számokat ismerjen fel a lexer. A {digit}* azt jelenti, hogy a akármilyen számjegy lehet 0-9-ig. Ezt követően a (.{digit}+)? részben. A \ karakter levédi a . -ot, hogy a számok tizedes jegyeire legyen értelmezve. Alapvetően csak azt jelentené a pont, hogy a lexer akármelyik betűre ráilleszthető. A digit+-nál + azt jelzi ,hogy fog állni még a tizedes jegy után legalább egy számjegy. A ? mindezt opcionálissá teszi, hiszen ha egész számokat adunk meg akkor nem írunk tizedes vesszőt, de a programnak az egész számokat is fel kell ismerni hiszen azok is valós

számok. Ezután egy printf() függvényel kiíratjuk a talált számot string és double alakban is. Double alakot az atof() függvény segítségével tudunk konvertálni a stringből.

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

A main() függvényben ezután nincs más dolgunk, mint meghívni a lexikális elemzőt a yylex() függvénnel. Majd printf() függvényel kiíratni a talált valós számok darabszámát.

```
int main(){

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

Egy példa a fordításra:

```
lex -o lexer.c lexer.l
gcc lexer.c -o lexer -lfl
./lexer
```

A program futását a CTRL+D-vel tudjuk megállítani ugyanis ez a kombináció megállítja az inputot. Ha CTRL+C vel próbáljuk meg "kilőni" a programot nem fogja kiírni a számok darabszámát.

```
user@ubuntu:~/Asztal/Prog1_programs/chomsky$ ./lex
aggadsagg5asfsaags425falasfnng&7.8lg
aggadsagg[realnum=5 5.000000]asfsaags[realnum=425 425.000000]falasfnng&[realnum=
7.8 7.800000]lg
The number of real numbers is 3
```

3.4. ábra. A program kimenete

3.5. l33t.l

Lexelj össze egy l33t cipher!

Megoldás videó:

Megoldás forrása: https://gitlab.com/tocsika7/prog1/blob/master/Chomsky/Lexer_leet/leet.l

Tanulságok, tapasztalatok, magyarázat...

A leet (vagy l33t) egy módosított írásmód amit legfőképpen az interneten használnak. Egyes betűket és számokat megadott karakterekkel cserélnek ki, amik hasonlítanak rájuk. Az alábbi program lexer segítségével kicseréli a bemeneten kapott betűket vagy számokat a hozzájuk tartozó leet karakterek egyikére.

A program első részében létrehozunk egy cipher nevű struktúrát. Ez áll egy karakterből vagyis az angol abc szavaiból és számoktóból 0-9-ig, illetve egy 4 elemű pointer tömbből, aminek az elemei ezek a szavak, és számok leet változatai. Ezeket stringekben kell tárolni, mert van amelyik nem csak 1 karakter hosszú. Ezeket a l337d1c7 tömbben adjuk meg. Emellett definiáljuk még a L337SIZE konstansot ami a tömb méretét osztja el egy struktúra méretével. A tömbbe betöltött karakterek leet változatai a wikipédián megtalálhatóak.

```
/*
```

Fordítás:

```
$ lex -o 1337d1c7.c 1337d1c7.l
```

Futtatas:

```
$ gcc 1337d1c7.c -o 1337d1c7 -lfl  
(kilépés az input vége, azaz Ctrl+D)
```

Copyright (C) 2019

Norbert Bátfai, batfai.norbert@inf.unideb.hu

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

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

```
You should have received a copy of the GNU General Public License  
along with this program. If not, see <https://www.gnu.org/licenses/>.
```

```
*/
```

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <ctype.h>  
  
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))
```

```
struct cipher {  
    char c;  
    char *leet[4];  
} l337d1c7 [] = {  
  
    {'a', {"4", "4", "@", "/-\\"}},  
    {'b', {"b", "8", "|3", "|"}},  
    {'c', {"c", "(", "<", "{"}},
```

```

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

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

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

%}

```

A következő programrészben megadjuk a szabályokat a lexernek, tehát azt, hogy milyen karaktereket kereszen a bemeneten. Csak egy . -ot adunk meg, mert azt akarjuk, hogy minden bemenő karakterre értelmezve legyen. Ezután egy for ciklussal végigmegyünk a tömbön és megnézzük, hogy melyik karakterelem egyezik a bemenettel. A bemenetet a tolower() függvényel kisbetűvé konvertáljuk. Generálunk egy random számot 100-ig. Ha a szám:

- 91-től kisebb: Az első tömbelemre cseréljük a bemenetet
- 95-nél kisebb: A második tömbelemre cseréljük a bemenetet
- 98-nál kisebb: A harmadik tömbelemre cseréljük a bemenetet

- minden egyéb esetben: A negyedik tömbelemre cseréljük a bemenetet.

Ha megtörtént a csere a found változó értke 1-lesz, a ciklus kilép és olvassuk a következő karaktert. Ha a found 0 marad tehát nem találja meg az adott karaktert (pl. ékezetes betűk) akkor visszadaja a bemenetet.

```
%%
. {

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

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

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

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

found = 1;
break;
}

}

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

}
%%
```

A main függvényben nincs más hátra, mint meghívni a lexikális elemezőt. És egy srand() függvényt a random szám generáláshoz.

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

```
user@ubuntu:~/Asztal/Prog1_programs/chomsky$ ./l33t
Ez egy pelda
3z 3[y p3ld4
Egy masik pelda
3gy m4s1k |D3ld4
```

3.5. ábra. A program kimenete

3.6. A források olvasása

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

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

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

Bugok

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

i.

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

Ha a SIGINT jel kezelése nem volt figyelmen kívül hagyva, akkor ezután kezelje a jelkezelő függvény.

ii.

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

A a ciklus deklarál egy i változót és ameddig az kisebb mint 5, prefix módon növeli.

iii.

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

A a ciklus deklarál egy i változót és ameddig az kisebb mint 5 postfix módon növeli.

iv.

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

A ciklus deklarál egy i változót és ameddig az kisebb mint öt, egy tömb adott i edik elemének értékül adja az eggyel nagyobb i-t. Viszont a kód bugos, a futása nincs elrőe megadva. A tomb[i] = i++ résznél a bal oldali operandus használja az i-t ami a jobb oldalon módosítva van. Mivel az i értékét egyszerre használtuk és módosítottuk megadott sorrend megadása nélkül a futás ezért kiszámíthatatlan.

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

A ciklus deklarál egy i változót és ameddig az kisebb, mint 5 és a d mutató+1 megegyezik az s mutató+1 el, az i-t prefix módon növeli. Ez a csipet bugos, mert az és operátor jobb oldalán a pointerek miatt nem logikai értéket kapunk.

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

A kód kiiratja két függvény eredményét. Viszont bugos, mert az első függvény módosítja a-t ami a második függvény által használva van. Mivel nem mondta meg előre a használati sorrendet, ezért kiszámíthatatlan lesz a futás.

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

Kiírja az f függvény eredményét, az a paraméterrel és kiírja az a változót.

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

Kiírja az f függvény eredményét, az a referenciaját megadva paramétrül és a kiírja az a változót.

Megoldás forrása:

Megoldás videó:

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

Bármely x esetén létezik olyan y, hogy az x kisebb, mint az y és y prím.

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

Bármely x esetén létezik olyan y , hogy x kisebb, mint y , y prím és az y rákövetkezőjének a rákövetkezőjé is prím.

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

Létezik olyan y , hogy bármely x esetén ha x prím akkor x kisebb, mint y .

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

Létezik olyan y , hogy bármely x esetén, ha y kisebb, mint x akkor x nem prím.

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 változó.
- `int *b = &a;`

Egy egészre mutató pointert

- `int &r = a;`

Egy egész referenciaját

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

Egész elemeket tartalmazó tömböt

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

Tömb referenciáját

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

Egy pointer tömbbőt

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

Egy függvényt aminek a visszatérési értéke pointer

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

Egy pointert ami függvényre mutat

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

Egy függvényt ,ami egy egészet kap értékül és visszatérési értéke egy pointer ,ami egy függvényre mutat, amely 2 egészet kap értékül.

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

Egy pointert ami egy függvényre mutat, és ez a függvény egy egészet kap értékül és visszatérési értéke egy pointer ,ami egy függvényre mutat, amely 2 egészet kap értékül.

Megoldás video:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Chomsky/Deklaraciok/dec.cpp>

Tanulságok, tapasztalatok, magyarázat...

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

//Egészre mutató mutatót visszadó függvény
int *func(int a){
int *ptr2= &a;
std::cout<<*ptr2<<"\n";
return ptr2;}

//Egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó ←
, egészet kapó függvény
int seged(int a,int b){
int c=a+b;
return c;}
```

```
int (*func2(int a))(int a,int b){
int (*funcptr2)(int,int)=&segéd;
return funcptr2;
}

int main(){

//Egész
int a=5;
int b=6;

//Egész referenciaja
int &ref = a;

//Egészre mutató muató
int *ptr = &a;

//Egészek tömbje
int tomb[5];

//Egész tömb refenciája
int (&tombref)[5] = tomb;

//Egészre mutató mutatók tömbje
int *ptrtomb [5];

//Egészre mutató mutatót visszadó függvényre mutató muató
int* (*funcptr)(int);
funcptr = &(*func);

//Függvénymutató egy egészet visszaadó és két egészet kapó függvényre ←
// mutatót visszaadó, egészet kapó függvényre
int (*(*funcptr3)(int))(int, int);
funcptr3 = &(*func2);

std::cout<<*ptr<<" "<<&ref<<"\n";
std::cout<<funcptr<<"\n";

func(a);
```

```
return 0;
}
```

DRAFT

## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\* háromszögmátrix

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Cesar/dhm.c>

Tanulságok, tapasztalatok, magyarázat...

A háromszögmátrixok négyzetes mátrixok, két félét különböztetünk meg: az alsóháromszög- és felsőháromszög-mátrixot. Az alábbi program egy alsóháromszög-mátrixot fog előállítani. Ezeknek a mátrixoknak a főátlójá felett csupa 0 érték található.

Az nr változóban megadjuk a háromszögmátrix sorainak számát, a tm-nek ami egy pointerre mutató pointer lefoglalunk a memoriában egy mutatónyi helyet (8 byte), majd ki is iratjuk a helyét.

```
//A kód forrása: https://gitlab.com/nbatfai/bhax/blob/master/ ←
thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Cesar/tm.c

#include <stio.h>
#include <stdlib.h>

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

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

A malloc() függény lefoglal a memóriát és egy pointert térít vissza. Ha a lefoglalt méret 0 akkor vagy egy NULL pointert vagy egy olyan pointert térít vissza, amelyet később sikeresen át lehet adni a free() függvénynek. A tm-nek a malloc() segítsével lefoglalunk 40 byteot ( $5 \times 8$  (double \*)). Erre használnunk kell egy double \*\* típuskényszerítést, mert a malloc() alapvetően void\*-ot ad vissza. Ha nincs több memória és NULL értéket kapunk vissza a program megáll. Ezután még kiiratjuk amit a tm-ben visszaad a malloc().

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

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

Egy for ciklusban elmegyünk a sorok számáig. A malloc() függvényel az adott tm elemekbe betöljtük a hozzájuk tartozó memória lefoglalást, például: tm[0] = 8 \* 0 + 1 byte = 8 byte Itt szintén használunk double\* típuskényszerítést. Emellett még kiiratjuk a 0-dik betöltött elemét a tm-nek.

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

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

Ezután feltöljük a háromszög mátrixot egy dupla for ciklus segítségével számokkal 0-14 ig és ezeket ki is íratjuk.

```
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");
}
```

A program utolsó szakaszában a free() függvényel felszabadítjuk a pointerek által lefoglalt helyeket a memoriában.

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

free (tm);

return 0;
}
```

```
user@ubuntu:~/Asztal/Prog1_programs/ceasar$./tm
0x7ffe885df200
0x559670d9b670
0x559670d9b6a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
```

4.1. ábra. A program futása után kapott háromszögmátrix

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Cesar/e.c>

Tanulságok, tapasztalatok, magyarázat...

Definiálunk két konstansot a maximális kulcs méretet és buffer méretet. A a kulcs méret egyértelmű míg a buffer méret a beolvasott bytok maximális számát jelenti.

```
//A kód forrása:https://progpater.blog.hu/2011/02/15/ ↵
felvetelt_hirdet_a_cia

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

Deklarálunk két char tömböt amelyeknek a mérete annyi, mint a két max konstans.

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

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

A kulcs\_meret változóban strlen() függvénnyel lekérjük megadott parancssori argumentum (vagyis kulcs) hosszát. Az strcpy() függvénnyel pedig átmásoljuk a parancssori argumentumot bytonként a kulcsba.

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

A while ciklusban a read() függvényel az olvasott\_bytok változónak értékül adjuk a bufferból beolvasott bytok számát, ami a BUFFER\_MERETNÉL nem lehet nagyobb.

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

```
{
```

Egy for ciklusban elmegyünk 0-tól a beolvasott bytokig és a buffer adott byte-ját exorozzuk a kulcs adott byte-jával. A kulcs indexet minden byte után átírjuk az adott index mérettel való maradékos osztására. Így a titkosítás jobb lesz, mert a kulcs változatosabb.

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

```
{
```

```
 buffer[i] = buffer[i] ^ kulcs[kulcs_index];
```

```
 kulcs_index = (kulcs_index + 1) % kulcs_meret;
```

```
}
```

A write() függvényel beírjuk a titkosított szöveget bytonként a bufferba az olvasott bytokig.

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

```
}
```

```
}
```

```
user@ubuntu:~/Asztal/Prog1_programs/ceasar$ cat tiszta.szoveg
Ez itt egy tiszta szöveg
user@ubuntu:~/Asztal/Prog1_programs/ceasar$ cat titkos.szoveg
.1R
1B2C3D4E5F6G7H8I9J0K1L2M3N4O5P6Q7R8S9T0U1V2W3X4Y5Z6
user@ubuntu:~/Asztal/Prog1_programs/ceasar$
```

4.2. ábra. Titkosítás előtti és utáni szöveg

### 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Cesar/ExorTitkosito.java>

Tanulságok, tapasztalatok, magyarázat...

A java verzióban lényegi részében vagyis a titkosításban változás nincsen a C verzióhoz képest. Az egész programot egy osztályba írjuk, ami lesz a public class ExorTitkosito. Ezután a titkosítás az ExorTitkosito függvénybe írjuk. Ennek paramétereit lesznek egy string, ami lesz a kulcs, illetve az output és input csatornák. A kulcsszöveget a getBytes() metódussal alakítjuk át bytok tömbjére. A kimenocsatorna.write()-al beírjuk a titkosított szöveget a megadott output fájlba. Az olvasott bytokat a read() metódussal olvassuk be a megadott input fájlból.

```
//A kód forrása: Bátfai Norbert, Juhász István: Javát tanítok könyv 51. ←
o

public class ExorTitkosito {

 public static void ExorTitkosito(String kulcsszoveg, java.io.InputStream ←
 bejovoCsatorna, java.io.OutputStream kimenocsatorna) throws java.io. ←
 IOException {

 byte [] kulcs = kulcsszoveg.getBytes();

 byte [] buffer = new byte[256];

 int kulcsIndex = 0;

 int olvasottBajtok = 0;

 while(olvasottBajtok != -1) {

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

 buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);

 kulcsIndex = (kulcsIndex+1) % kulcs.length;

 }

 kimenocsatorna.write(buffer, 0, olvasottBajtok);
 olvasottBajtok=bejovoCsatorna.read(buffer);
 }
 }

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

 ExorTitkosito(args[0], System.in, System.out);
 } catch(java.io.IOException e) {
 e.printStackTrace();
 }
 }
}
```

}

## 4.4. C EXOR törő

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

Tutor: Borvíz Róbert [https://github.com/BorvizRobi/prog\\_1\\_textbook/tree/master/beadando](https://github.com/BorvizRobi/prog_1_textbook/tree/master/beadando)

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Cesar/t.c>

Tanulságok, tapasztalatok, magyarázat...

A program a szükséges headerek meghívásával és a szükséges konstansok definiálásával kezdődik.

```
//A kód forrása: https://progpater.blog.hu/2011/02/15/ ←
felvetelt_hirdet_a_cia

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

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

Az atlagos\_szohossz függvény elosztja a titkos szöveg méretét (byteokban) a szóközök számával, ezáltal megkapjuk egy szó hányszámos átlagosan.

```
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;
}
```

A tiszta\_lehet függvénytel megnezzük tartalmazza a szöveg a leggyakoribb magyar szavakat amik a következők: hogy,nem,az,ha. Megnézzük még, hogy az átlag szóhossz 6 és 9 között van-e. Ezt a két lépést a return-ben tesszük meg és operátorok segítségével. C-ben a logikai operátorok egy egész értéket adnak vissza. Ezzel a potenciális törések számát csökkenhetjük.

```
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");
}
```

Az exor függvény for ciklusában végigmegyünk a titkos szövegen és a titkos szöveget adott byte-jait exorozzuk a kulcs adot klucsindex-szel. A kules indexet mindenkor növeljük 1-el és a kules méretével osztjuk maradékosan. Ezáltal a kulcs\_index mindenkor 0-ról indul és ha eléri a kulcs\_meret konstansot akkor is kinullázódik.

```
void
exor (const char kulcs[], int kulcs_meret, char titkos[], int ←
 titkos_meret)
{
 int kulcs_index = 0;
 for (int i = 0; i < titkos_meret; ++i)
 {
 titkos[i] = titkos[i] ^ kulcs[kulcs_index];
 kulcs_index = (kulcs_index + 1) % kulcs_meret;
 }
}
```

Az exor\_tores függvényben meghívjuk az exor függvényt illetve returnben pedig a tiszta\_lehet függvényt. Ez a függvény azt vizsgálja sikeres volt-e a törés.

```
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);
}
```

A main függvény két tömböt hozunk létre, az egyikben a kulcsokat, míg a másikban a titkos szöveget fogjuk tárolni. Deklaráljuk p pointert, ami titkos szöveg első karakterére mutat. While ciklusában a read függvénytel betöljük az olvasott\_bajtok változóba az olvasott bytok számát. A read függvény paraméterei: honnan olvassuk be a szöveget: a 0-ról tehát a standard inputról, hol tároljuk a szöveget amit beolvassunk és mikkora a max méret amit beolvassunk. Ha a p és a titkos szöveg különbségének a buffer byteszámával vett összege kisebb, mint titkos szöveg maximális byte száma (4096), akkor a bufferig olvassa be a bytokat. Ha nagyobb akkor a titkos szöveg + 4096 byte - p ig olvassa be a bytokat. Ha a beolvás megtörtént a p-nek értékül adjuk az utolsó beolvastott byte memóriacímét.

```
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;
```

Ezután egy for ciklus segítségével kinullázzuk azokat a helyeket a titkos szövegben, ahol nincs semmi.

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

For ciklusokkal előállítjuk lehetséges kulcsokat, majd ha az exor törés függvényel kipróbáljuk őket és ha igazat ad kész vagyunk. Ha nem sikerült a törés visszafejtjük az eredeti szöveget és újrapróbálkozunk.

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

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

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

```
}
```

```
return 0;
}
```

## 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)

Tanulságok, tapasztalatok, magyarázat...

A neuronokról általánosan:

A neuron matematikai modellje szerint a neuron akkor "tüzel" ha a bemeneti értékek súlyozott összege egy bizonyos határértéket meghalad. A bemeneten számok érkeznek, amelyeknek van egy súly értéke.

A programban meghívjuk a neuralnet könyvtárat. Az a1-be és a2-be betöljtük a neuronok lehetséges értékeit, az OR-ba pedig ezek logikai 'vagy' műveletének értéket. Elmondjuk az összes lehetséges esetet a sikeres tanításhoz. Ezután a program elkezdi magát tanítani és beállítja magának a súlyokat. Ugyanilyen módon megtanítjuk neki az 'és' és 'exor' műveleteket. Exornál tanulási algoritmus annál hatékonyabb, minnél a több a rejttett neuronok száma.

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

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

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

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

plot(nn.or)

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

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

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

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

plot(nn.operand)

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

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

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)

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

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Cesar/main.cpp>

Tanulságok, tapasztalatok, magyarázat...

Ebben a programban egy Mandelbrot halmaz programozhoz írunk perceptron-t. A program első sorában létrhezzük a képet majd felveszük a size változóba a kép méretét Összeszorozzuk a magasságát a szélességével, a get\_width() és get\_height() függvényt használjuk a magasság és szélesség lekérésére.

Példányosítunk egy új perceptron objektumot, ami 3 rétegű:

- Az első rétegbe annyi neuront akarunk, amekkora a kép mérete
- A második rétegbe 256-ot, ez igazából akármennyi lehetne
- A harmadik rétegbe 1-et, mert ez az eredménynek lesz lefoglalva

A double\* image-ben helyet foglalunk a képnek és feltöljük a beolvasott képpel egy dupla for ciklussal. Az egyikben a szélességgig míg a másikban a magasságig megyünk el. A betöltésnél csak a piros színkódú komponenseket töltjük be a képhez. Ezután a perceptron, mint függvényt használjuk az eredmény (value változó) értékadásánál. A program végén a pointerek és a kép által lefoglalt memóriát szabadítjuk fel delete() függvényekkel.

//A kód forrása: <https://www.youtube.com/watch?v=XpBnR31BRJY>

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

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

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

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

 double* image = new double[size];

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

 double value = (*p)(image);

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

 delete p;
```

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

DRAFT

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

A feladatban tutoriáltam: Borvíz Róbert [https://github.com/BorvizRobi/prog\\_1\\_textbook/tree/master/beadando](https://github.com/BorvizRobi/prog_1_textbook/tree/master/beadando)

A Mandelbrot halmazt Benoît Mandelbrot fedezte fel és róla nevezték el 1982-ben. A halmazt olyan komplex számok alkotják melyek sorozata nem tart a végtelenbe. Ha a halmazt ábrázoljuk a komplex számsíkon egy jellegzetes ún. fraktál alakzatot kapunk. A halmazt úgy kapjuk meg, hogy felvesszük a komplex számsíkon egy rácsot, majd kiszámojuk, hogy a rács adott pontja melyik komplex számnak felel meg. Megvizsgáljuk a rács pontjait a  $z_{n+1} = z_n^2 + c$  képlet segítsével, amelyben a  $c$  az adott rácspont a  $z_0$  pedig az origó. Elmegyünk az origótól ( $z_0$ ) tól a  $z_1 = c$  ig. Majd a  $c$ -től függően további  $z$ -kbe. Ha ez az urágas kivezet egy 2 sugarú körből akkor az nem a Mandelbrot halmaz része, ha benne marad ebben a tartományban akkor pedig igen.

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Mandelbrot/mandel.cpp>

A program lényegi részét a mandel() függvénybe írjuk, amelynek a paraméterei a kép méretei. Deklaráljuk a szükséges változókat, majd egy dupla for ciklussal végigmegyünk a kép szélességén és magasságán. Mivel itt nem használjuk a komplex osztályt, ezért itt két változó lesz egy komplex szám valós és képzetes részei. (reC, imC illetve reZ, imZ). A reC és imC a rács csomópontjának megfelő komplex szám lesz, ami képletben a  $c$ . A reZ és az imZ a képletben szereplő  $z$  értéknek felelnek meg. Alkalmazzuk a fent említett képletet addig, ameddig a  $z$  abszolút értéke kisebb, mint 2 vagy elérjük az iterációs határt. Ha ez megtörtént átadjuk a kép paramétereinek az adott iterációt, mivel az adott komplex szám a halmaz eleme. A main() függvényben ellenőrizzük jól van-e használva a program: csak akkor fut ha helyesen, ha 2 paramétert adunk meg parancssori argumentumként (fájlnév, menteni való kép neve). Ellenkező esetben a helyes használatot írja ki. Alkalmazzuk a mandel() függvényt a kép adatain, amit a png++-al hozunk létre. Majd végigmegyünk a kép szélességén és magasságán és beállítjuk a kívánt színeket set\_pixel() függvénnnyel.

```
// mandelpngt.cpp
// Copyright (C) 2019
// Norbert Bátfa, batfa.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
```

```
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←
_01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
//
#include <iostream>
#include "png++/png.hpp"

#define MERET 600
#define ITER_HAT 32000

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

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

 float dx = (b - a) / szelesseg;
 float dy = (d - c) / magassag;
 float reC, imC, reZ, imZ, ujreZ, ujimZ;

 int iteracio = 0;

 for (int j = 0; j < magassag; ++j)
 {

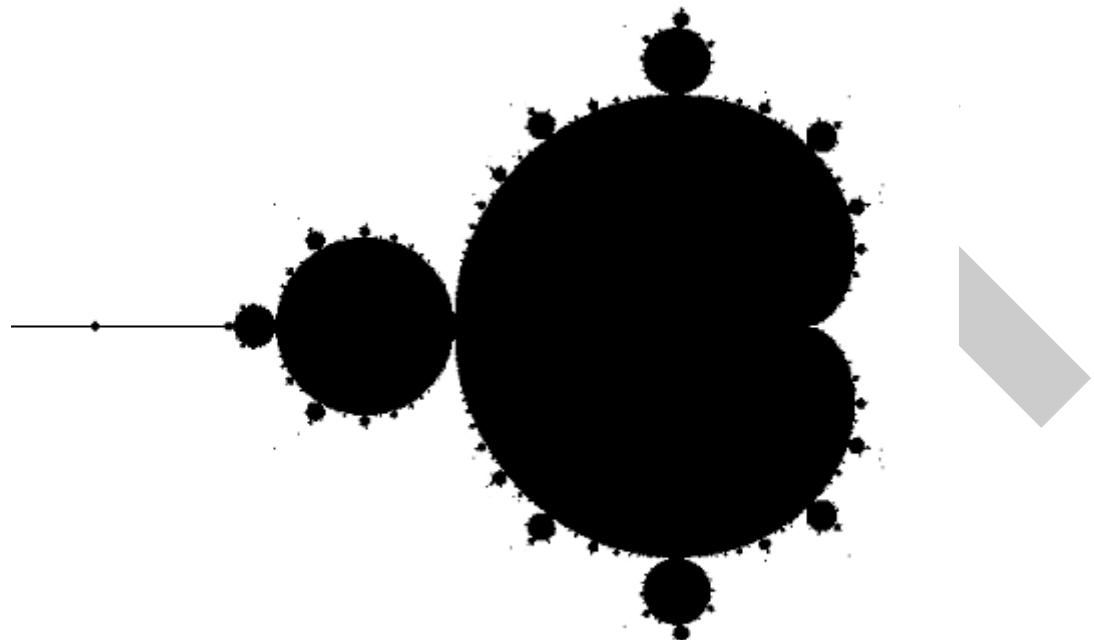
 for (int k = 0; k < szelesseg; ++k)
 {
 reC = a + k * dx;
 imC = d - j * dy;
 // z_0 = 0 = (reZ, imZ)
 reZ = 0;
 imZ = 0;
 iteracio = 0;
```

```
 while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
 {
 // z_{n+1} = z_n * z_n + c
 ujreZ = reZ * reZ - imZ * imZ + reC;
 ujimZ = 2 * reZ * imZ + imC;
 reZ = ujreZ;
 imZ = ujimZ;

 ++iteracio;
 }

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

int
main (int argc, char *argv[])
{
 if (argc != 2)
 {
 std::cout << "Hasznalat: ./mandelpng fajlnev";
 return -1;
 }
 int kepadat[MERET][MERET];
 mandel(kepadat);
 png::image<png::rgb_pixel> kep(MERET, MERET);
 for (int j = 0; j < MERET; ++j)
 {
 //sor = j;
 for (int k = 0; k < MERET; ++k)
 {
 kep.set_pixel(k, j,
 png::rgb_pixel(255 -
 (255 * kepadat[j][k]) / ITER_HAT) -->
 ,
 255 -
 (255 * kepadat[j][k]) / ITER_HAT) -->
 ,
 255 -
 (255 * kepadat[j][k]) / ITER_HAT));
 }
 }
 kep.write(argv[1]);
 std::cout << argv[1] << " mentve" << std::endl;
}
```



5.1. ábra. Egy Mandelbrot halmaz

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

Megoldás videó:

Megoldás forrása: [https://gitlab.com/tocsika7/prog1/blob/master/Mandelbrot/m\\_komplex.cpp](https://gitlab.com/tocsika7/prog1/blob/master/Mandelbrot/m_komplex.cpp)

A program a fenti algoritmus alapján számít és rajzol ki egy Mandelbrot halmazt. A program futásához szükség van a png++ és a complex header-ök meghívására. A programot azzal kezdjük, hogy deklaráljuk a létrehozandó halmaz képének szélességét és magasságát, illetve az iterációs határt. Ezeknek az értékeit parancssori argumentumként fogjuk megadni futtatásnál.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ←
// -0.01947381057309366392260585598705802112818 ←
// -0.0194738105725413418456426484226540196687 ←
```

```
0.7985057569338268601555341774655971676111 ←
0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ←
0.4127655418209589255340574709407519549131 ←
0.4127655418245818053080142817634623497725 ←
0.2135387051768746491386963270997512154281 ←
0.2135387051804975289126531379224616102874
// Nyomtatás:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1 --left-footer="←
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>
```

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

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

{

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

Ellenőrizzük, hogy a futtatáskor megadott argumentumok száma helyes-e, ha igen átadjuk őket a deklarált változóknak, ha nem a program nem fut le és kiírja a helyes futtatási módot. Az atoi() függvénytel konvertáljuk a megadott értékeket int-é és az atof-al pedig double-é.

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

PNG++ segítségével létrehozzuk a képet, maj egy dupla for ciklussal végigmegyünk a kép szélességén és magasságán és alkalmazzuk a előző feladatban tárgyalt képletet. A c két része lesz a reC és az imC változók míg a z-é a reZ és imZ váltók. Ezekből a complex osztály segítségével készítünk egy-egy komplex számot. A képletet addig használjuk amíg a z abszolút értéke kisebb, mint 4 (abs() függvény segítségével) és az iteráció nem érte el az iterációs határt. A set\_pixel() függvénytel állítjuk be a halmaz színeit majd a write() függvénytel el is mentjük a képet.

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

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

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

for (int j = 0; j < magassag; ++j)
{

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

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

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

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

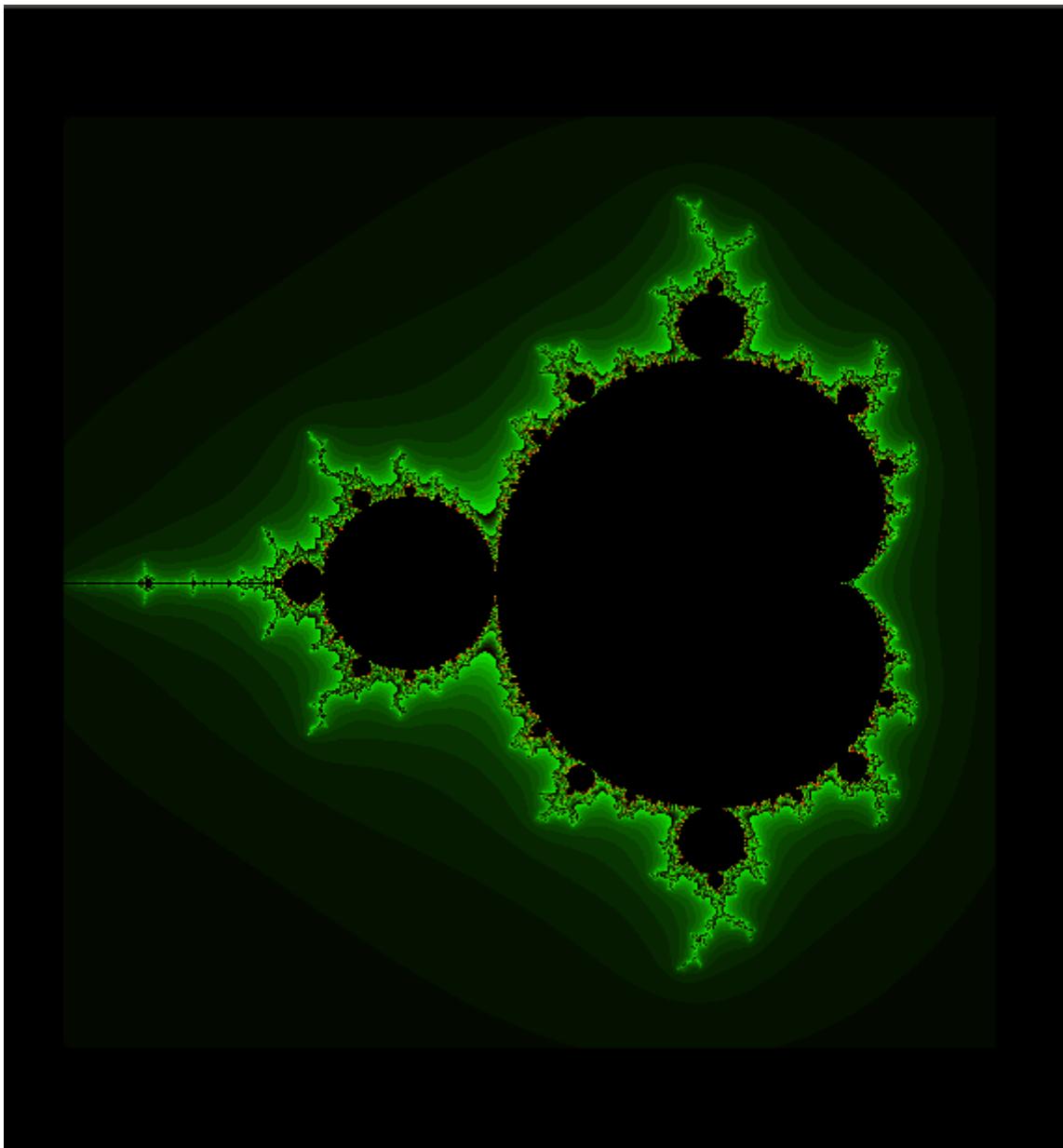
 ++iteracio;
}

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

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

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





5.2. ábra. A program egy féle kimenete

### 5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

A bimorfok vagy Júlia halmazok bizonyos komplex egyenletek megoldásainak ábrázolásai a komplex szám síkon. A Mandelbrot halmaz azért különleges, mert tartalmazza az összes ilyen halmazt. A Mandelbrot halmaz és a egy Júlia halmaz vagy biomorf megvalósításában a lényeges különbség, hogy a biomorfok

esetében a c egy konstans. A C++ megvalósításában nagyon hasonlít a Mandelbrot halmazhoz. A rácspontokon való futás szerepét a z\_n veszi át, míg a c-t már csak parancsosri argumentumként fogjuk megadni és a futás során nem változtatunk rajta. A dupla for ciklusban lesz egy harmadik for ciklus amelyben az iterációs határig megyünk el és a halmaz kiszámítására a  $z^3 + c$  képletet használjuk (erre sok féle képlet létezik amelyek változatos halmazokat adnak). A z\_n a pow() függvényel emeljük a 3. hatványra.

Változás az előző programhoz képest:

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatás:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -l --line-numbers=1 --left-footer="←
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
// color
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ←
// Vol19_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

for (int y = 0; y < magassag; ++y)
{
 for (int x = 0; x < szelesseg; ++x)
 {

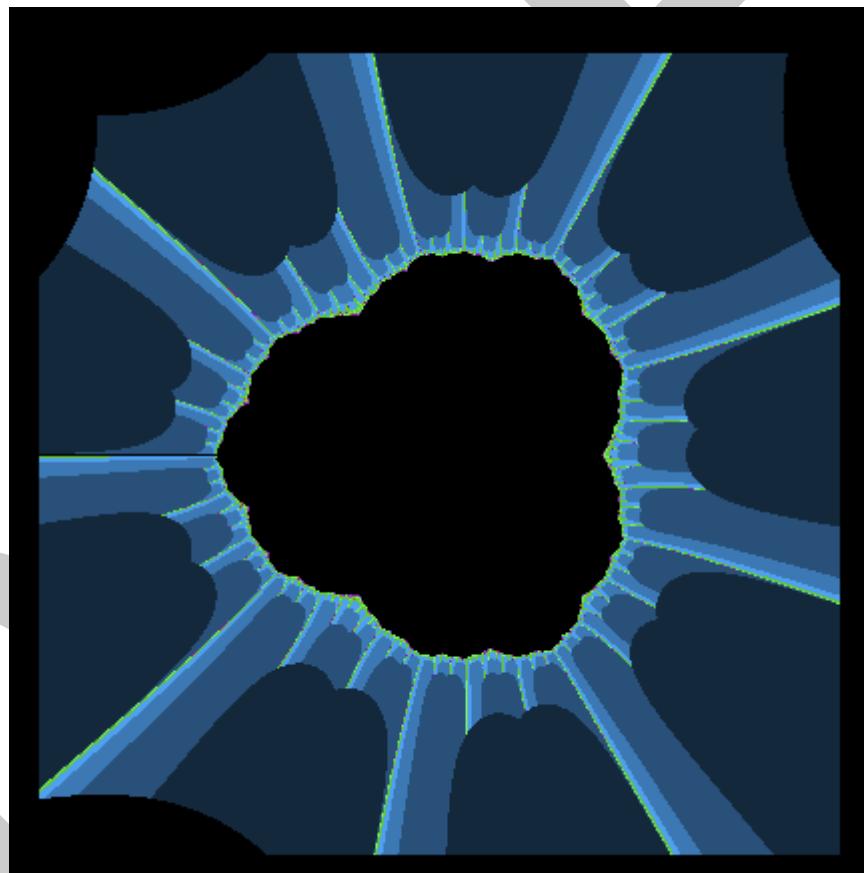
 double rez = xmin + x * dx;
```

```
double imZ = ymax - y * dy;
std::complex<double> z_n (reZ, imZ);

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

 z_n = std::pow(z_n, 3) + cc;

 if(std::real (z_n) > R || std::imag (z_n) > R)
 {
 iteracio = i;
 break;
 }
}
```



5.3. ábra. Egy a program futtatásával kapott biomorf

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

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

A CUDA-s megvalósítás nagymértékben csökkenti a programok futási idejét. Amíg egy általános szekvenciális program csak a processzor egy szálát használja a végrehajtáshoz, addig a CUDA-s program a GPU szálait használja. A Mandelbrot halmaz esetében 60x60-as blokkot hozunk létre és minden blokkban 100 szál fog működni. (600x600-as halmaz). Ez a megoldás körülbelül 50-70-szer gyorsabb megoldást eredményez, mint a szekvenciális. Létrehozzuk a mandelkernel nevű kernelt, ami a CUDA programozásban úgy használtható, mint C-ben egy függvény. minden szál, ami használja ezt a kernelt külön azonosítóval rendelkezik, ezek lesznek a tj és tk. Létrehozzuk a j és k blokkokat is, amikbe 10-szer annyi szálat helyezünk. A cudamandel függvényben lefoglaljuk a szükséges memóriát a cudaMalloc() függvénnnyel. Létrehozzuk a hálót ami dim3 típusú tehát 3 dimenziós. Majd meghívjuk a mandelkernel függvényt. A cudaMemcpy függvénnnyel átmásoljuk az adatokat a host memoriából a device memoriába. Majd a cudaFree() függvénnnyel felszabadítjuk a memóriát.

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs

_global__ void
mandelkernel (int *kepadat)
{

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

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

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

```
}

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

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

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

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

## 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 forrása: [https://gitlab.com/toesika7/prog1/tree/master/Mandelbrot/Mandelbrot\\_Nagyito\\_C++](https://gitlab.com/toesika7/prog1/tree/master/Mandelbrot/Mandelbrot_Nagyito_C++)

Megoldás videó:

A programot gui-ját QT-ban oldjuk meg ehhez nem 1 programfájlt alkalmazunk, hanem 3 .cpp fájlt 2 headert és egy .pro fajlt. A Frak.pro fájlban a qt-nak adjuk meg a header-öket illetve a forrásfájlokat.

Frak.pro :

```
#####
Automatically generated by qmake (3.0) Sun Mar 30 10:39:23 2014
#####

QT += widgets
TEMPLATE = app
TARGET = Frak
INCLUDEPATH += .

Input
HEADERS += frakablak.h frakszal.h
SOURCES += frakablak.cpp frakszal.cpp main.cpp
```

A következőkben a minden forrásához tartozik egy külön header fájl, amiben a szükséges osztályokat, illetve függvényeket deklaráljuk.

frakablak.h és frakablak.cpp :

A frakablak fájlban és headerben a halmazt megjelenítő ablakot készítjük el. A program konstruktőrában beállítjuk az ablak nevét illetve paramétereit. minden nagyításnál újraszámítjuk a képet, ezt a paintEvent() QT függvényel tesszük. A vissza() függvényel a kép színeit állítjuk be.

```
// frakablak.cpp
//
// Mandelbrot halmaz rajzoló
// Programozó Páternoszter
//
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail.com ←
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1 Bár a Nokia Qt SDK éppen tartalmaz egy Mandelbrotos példát, de
// ezt nem tartottam megfelelőnek első Qt programként ajánlani, mert elég
// bonyolult: használ kölcsönös kizárást stb. Ezért "from scratch" megírtam
// egy sajátot a Javát tanítokhoz írt dallamára:
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
```

```
//

#include "frakablak.h"

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

 int magassag = (int)(szelesseg * ((d-c) / (b-a)));

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

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

}

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

void FrakAblak::paintEvent(QPaintEvent*) {
 QPainter qpainter(this);
 qpainter.drawImage(0, 0, *fraktal);
 qpainter.end();
}

void FrakAblak::vissza(int magassag, int *sor, int meret, int hatar)
{
 for(int i=0; i<meret; ++i) {
 // QRgb szin = qRgb(0, 255-sor[i], 0);
 QRgb szin;
 if(sor[i] == hatar)
 szin = qRgb(0,0,0);
 else
 szin = qRgb(
 255-sor[i],
 255-sor[i]%64,
 255-sor[i]%16);

 fraktal->setPixel(i, magassag, szin);
 }
 update();
}
```

## frakszal.h és frakszal.cpp

A frakszal.cpp fájlban a konstruktorban szükséges változókat állítjuk be. A run() függvényben pedig a halmazt létrehozó már az előző programokban tárgyalt számítás zajlik. A szinezést nem itt végezzük, hanem a frakablak programban, illetve az adott iterációt is a vissza függvényel átadjuk a frakablak forrásnak hogy az ki tudja rajzolni.

```
// frakszal.cpp
//
// Mandelbrot halmaz rajzoló
// Programozó Páternoszter
//
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail.com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
// Version history:
//
// 0.0.1 Bár a Nokia Qt SDK éppen tartalmaz egy Mandelbrotos példát, de
// ezt nem tartottam megfelelőnek első Qt programként ajánlani, mert elég
// bonyolult: használ kölcsönös kizárást stb. Ezért "from scratch" megírtam
// egy sajátot a Javát tanítokhoz írt dallamára:
```

```
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
//

#include "frakSzal.h"

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

 egySor = new int[szelessseg];
}

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

// A szál kódját a Javát tanítokhoz írt Java kódomból vettetem át
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
// mivel itt az algoritmust is leírtam/lerajzoltam, így meghagytam
// a kommenteket, hogy a hallgató könnyen hozzáolvashassa az "elméletet",
// ha érdekli.

void FrakSzal::run()
{
 // A [a,b]x[c,d] tartományon milyen sűrű a
 // megadott szélesség, magasság háló:
 double dx = (b-a)/szelessseg;
 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<szelessseg; ++k) {
 // c = (reC, imC) a háló rácspontjainak
 // megfelelő komplex szám
 reC = a+k*dx;
 imC = d-j*dy;
 // z_0 = 0 = (reZ, imZ)
 reZ = 0;
```

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

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

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

A fordítás:

```
qmake Frak.pro
make Frak
./Frak
```

## 5.6. Mandelbrot nagyító és utazó Java nyelven

A Mandelbrot nagyítót java-ban úgy oldjuk meg, hogy egy java-s Mandelbrot halmaz megvalósítás osztályát terjesztjük ki.

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Mandelbrot/MandelBrotHalmazNagyito.java>

```
/*
 * MandelbrotHalmazNagyito.java
 *
 * DIGIT 2005, Javat tanitok
 * Batfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyito es kirajzolo osztaly.
 *
 * @author Batfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.2
 */
public class MandelbrotHalmazNagyito extends MandelbrotHalmaz {
 /**
 * A nagyitando kijelolt teruletet bal felső sarka.
 */
 private int x, y;
 /**
 * A nagyitando kijelolt terulet szeessege es magassaga.
 */
 private int mx, my;
 /**
 * Letrehoz egy a Mandelbrot halmazt a komplex sik
 * [a,b]x[c,d] tartomanya felett kiszamolo es nygitani tudo
 * <code>MandelbrotHalmazNagyito</code> objektumot.
 *
 * @param a [a,b]x[c,d] tartomany a koordinataja.
 * @param b [a,b]x[c,d] tartomany b koordinataja.
 * @param c [a,b]x[c,d] tartomany c koordinataja.
 * @param d [a,b]x[c,d] tartomany d koordinataja.
 * @param szelesseg a halmazt tartalmazo tomb szeessege.
 * @param iteraciosHatar a szamitas pontossaga.
 */
 public MandelbrotHalmazNagyito(double a, double b, double c, double d,
 int szelesseg, int iteraciosHatar) {
 // Az os osztaly konstruktoranak hivasa
 super(a, b, c, d, szelesseg, iteraciosHatar);
 setTitle("A Mandelbrot halmaz nagyitasai");
 }
}
```

A a nagyítandó területet a bal egérgombbal tudjuk kijelölni. Az egér kattintásait egy listener segítségével dolgozzuk fel. Majd megnézzük hol van a mutató pozíciója. A kijelölő téglalap bal felső sarkának a koordinátái az x és y tagjai az osztálynak, míg a az mx és my a téglalap magassága és szélessége. A mousePressed() függvényben ezeket az adatokat kérjük le. Az mx és my-t 0-ra állítjuk és majd akkor fogjuk növelni ha a téglalapot vonszoljuk. A vonszolás mértékét egy egér mozgását érzékelő listener-rel kapjuk meg. Ha felengedjük az egert újrakezdi a számolást és csinál egy új objektumot. Ha kivonjuk téglalap x és y koordinátáiból az egérmutató aktuális helyzetét akkor megkapjuk a kirajzolandó téglalap kellő méreteit. A repaint() hívás gondoskodik arról, hogy a megfelelő téglalap legyen kirajzolva.

```
addMouseListener(new java.awt.event.MouseAdapter() {
 // Eger kattintassal jeloljuk ki a nagyitando teruletet
 // bal felső sarkat vagy ugyancsak eger kattintassal
 // vizsgaljuk egy adott pont iteracioit:
```

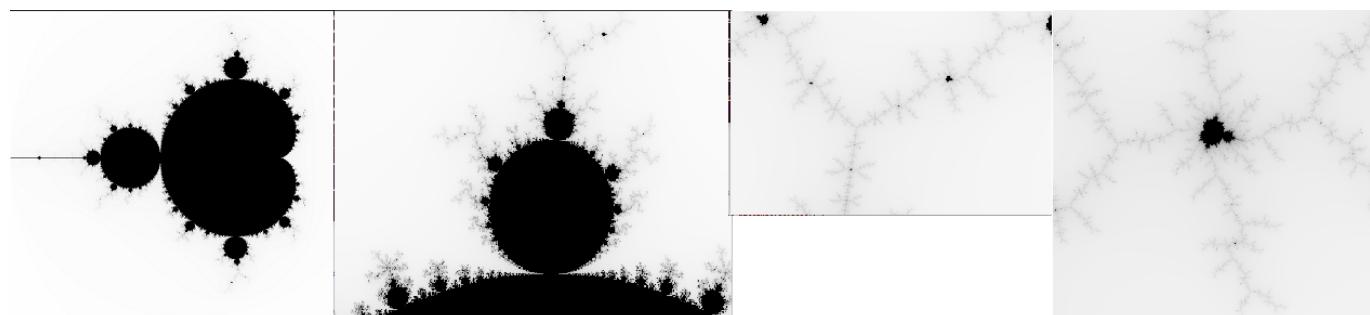
```
public void mousePressed(java.awt.event.MouseEvent m) {
 // Az egermutato pozicioja
 x = m.getX();
 y = m.getY();
 // Az 1. eger gombbal a nagyitando terulet kijeloleset
 // vegezzuk:
 if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
 // A nagyitando kijelolt teruletet bal felso sarka: (x, ←
 y)
 // es szelessege (majd a vonszolas noveli)
 mx = 0;
 my = 0;
 repaint();
 } else {
 // Nem az 1. eger gombbal az egermutato mutatta c
 // komplex szambol inditott iteraciokat vizsgalhatjuk
 MandelbrotIteraciok iteraciok =
 new MandelbrotIteraciok(
 MandelbrotHalmazNagyito.this, 50);
 new Thread(iteraciok).start();
 }
}
// Vonszolva kijelolunk egy teruletet...
// Ha felengedjuk, akkor a kijelolt terulet
// ujraszamitasa indul:
public void mouseReleased(java.awt.event.MouseEvent m) {
 if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
 double dx = (MandelbrotHalmazNagyito.this.b
 - MandelbrotHalmazNagyito.this.a)
 /MandelbrotHalmazNagyito.this.szelesseg;
 double dy = (MandelbrotHalmazNagyito.this.d
 - MandelbrotHalmazNagyito.this.c)
 /MandelbrotHalmazNagyito.this.magassag;
 // Az uj Mandelbrot nagyito objektum elkeszitese:
 new MandelbrotHalmazNagyito(
 MandelbrotHalmazNagyito.this.a+x*dx,
 MandelbrotHalmazNagyito.this.a+x*dx+mx*dx,
 MandelbrotHalmazNagyito.this.d-y*dy-my*dy,
 MandelbrotHalmazNagyito.this.d-y*dy,
 600,
 MandelbrotHalmazNagyito.this.iteraciosHatar)
 }
}
});
// Eger mozgas esemenyek feldolgozasa:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
 // Vonszolassal jeloljuk ki a negyzetet:
 public void mouseDragged(java.awt.event.MouseEvent m) {
 // A nagyitando kijelolt terulet szelessege es magassaga:
 mx = m.getX() - x;
```

```
 my = m.getY() - y;
 repaint();
 }
});
```

Ezután kidolgozzuk a pillanatkép készítő funkciót. Kirajzoljuk a területet jelző tégalalapot és visszaadjuk az egérmutató akutális helyzetét. A main()-ben ezek után csak példányosítunk egy MandelBrotHalmazNagyito objektumot.

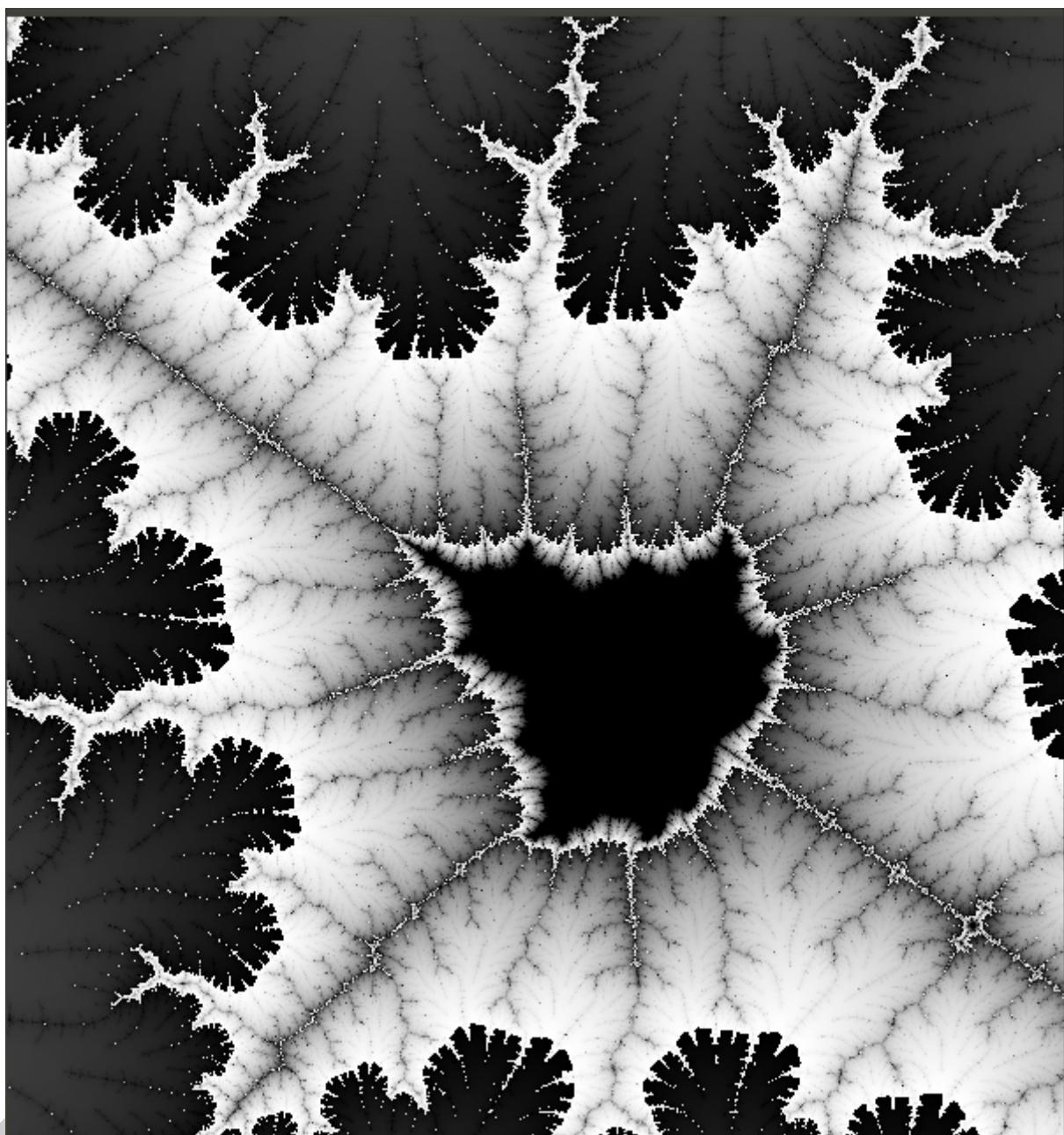
```
/***
 * Pillanatfelvetelek készítése.
 */
public void pillanatfelvetel() {
 // Az elmentendo kep elkeszitese:
 java.awt.image.BufferedImage mentKep =
 new java.awt.image.BufferedImage(szelessseg, magassag,
 java.awt.image.BufferedImage.TYPE_INT_RGB);
 java.awt.Graphics g = mentKep.getGraphics();
 g.drawImage(kep, 0, 0, this);
 g.setColor(java.awt.Color.BLACK);
 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=" + iteraciosHatar, 10, 75);
 if(szamitasFut) {
 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 pillanatfelvetel kep fajl nevenek kepzese:
 StringBuffer sb = new StringBuffer();
 sb = sb.delete(0, sb.length());
 sb.append("MandelbrotHalmazNagyitas_");
 sb.append(++pillanatfelvetelszamlalo);
 sb.append("_");
 // A fajl nevebe belelevesszuk, hogy melyik tartomanyban
 // talaltuk 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(mentKep, "png",
 new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
 e.printStackTrace();
}
}
/***
 * A nagyítando kijelolt teruletet jelző negyzet kirajzolása.
 */
public void paint(java.awt.Graphics g) {
 // A Mandelbrot halmaz kirajzolása
 g.drawImage(kep, 0, 0, this);
 // Ha eppen fut a szamitas, akkor egy voros
 // vonallal jelöljük, hogy melyik sorban tart:
 if(szamitasFut) {
 g.setColor(java.awt.Color.RED);
 g.drawLine(0, sor, getWidth(), sor);
 }
 // A jelző negyzet kirajzolása:
 g.setColor(java.awt.Color.GREEN);
 g.drawRect(x, y, mx, my);
}
/**
 * Hol áll az egérmutató?
 * @return int a kijelolt pont oszlop pozicioja.
 */
public int getX() {
 return x;
}
/**
 * Hol áll az egérmutató?
 * @return int a kijelolt pont sor pozicioja.
 */
public int getY() {
 return y;
}
/**
 * Peldanyosit egy Mandelbrot halmazt nagyító obektumot.
 */
public static void main(String[] args) {
 // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
 // tartományában keressük egy 600x600-as haloval és az
 // aktualis nagyítási pontossággal:
 new MandelbrotHalmazNagyito(-2.0, .7, -1.35, 1.35, 600, 255);
}
}
```



5.4. ábra. Egy 4-szeres nagyítás folyamata

DRAFT



5.5. ábra. Egy 13-szoros nagyítás

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

C++: [https://gitlab.com/tocsika7/prog1/blob/master/Welch/polar\\_generator.cpp](https://gitlab.com/tocsika7/prog1/blob/master/Welch/polar_generator.cpp)

Java: <https://gitlab.com/tocsika7/prog1/blob/master/Welch/Pol%C3%A1rGener%C3%A1tor.java>

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

A C++ verzióban azzal kezdünk, hogy létrehozzunk egy Random osztályt, majd annak public részében egy konstruktort, destruktort és egy get (random szám lekérő) függvényt. A private részben pedig egy exists (van-e korábbi random érték) és egy value (random szám értéke) változókat.

```
// A kód forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsocpp/random/random.cpp

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

class Random {
public:
 Random();
 ~Random() {}
 double get();
```

```
private:
 bool exist;
 double value;
};
```

Jobban kifejtjük a Random() konstruktort és get() függvényt,a konstruktorban, exist változót hamisra állítjuk (nincs random szám) és inicializálunk a srand() függvényt. A get() függvényben megnézzük, hamis-e az exist változó, ha igen deklarálunk 5 változót amit később az algoritmusban fogunk használni.

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

double Random::get () {

 if (!exist)
 {

 double u1, u2, v1, v2, w;
```

Maga az algoritmus is a get() függvényben zajlik le egy do while ciklusban a következő képpen:

```
do {
 u1 = std::rand () / (RAND_MAX + 1.0);
 u2 = std::rand () / (RAND_MAX + 1.0);
 v1 = 2 * u1 - 1;
 v2 = 2 * u2 - 1;
 w = v1 * v1 + v2 * v2;
}
while (w > 1);
double r = std::sqrt ((-2 * std::log (w)) / w);
value = r * v2;
exist = !exist;
return r * v1;
```

Az else ágban megnézzük van-e már korábban generált érték, ha van azt adjuk vissza.

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

A main()-ben nincs más hátra, mint létrehozni egy objektumot (rnd) a random osztályból és arra meghívni a get() függvényt egy for ciklusos kiiratásban.

```
int main()
{
 Random rnd;
 for (int i = 0; i < 10; ++i) std::cout << rnd.get() << std::endl;
}
```

Java-ban is ugyanúgy csinálunk minden, a különbség az, hogy a Random osztálynak nincsen private része, minden a publicba írunk. Emellett nem csinálunk destruktort, az exist változót pedig igaz-ról indítjuk.

//A program forrása: Bátfa Norbert, Juhász István: Javát tanítok könyv ←  
57. o.

```
public class Random { //Random osztály

boolean exist = true; //Van-e már korábbi érték
double value; //Ha van mi

public Random() { //Konstruktor létrehozása

exist = true;

}

public double get() { //Get függvény létrehozása

if(exist) {

 double u1,u2,v1,v2,w;
 do { //Az algoritmus kezdete
 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));

 value = r*v2;
 exist = !exist;

 return r*v1;
 //Az algoritmus vége
} else { //Ha van korábbi random érték, azt
 adja vissza
 exist= !exist;
}
```

←

```
 return value;
 }
}

public static void main(String[] args) {

 Random g = new Random();

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

}
}
```

```
user@ubuntu:~/Asztal/Prog1_programs/welch/Polart_alg$./a.out
-0.726425
-0.224597
-0.941281
0.130016
1.03797
-1.15403
-0.713387
-1.89291
-0.794773
-0.0448993
```

6.1. ábra. A program kimenete

## 6.2. LZW

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

Megoldás video:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Welch/binfa.c>

A programot azzal kezdjük, hogy készítünk egy binfa struktúrát, ami áll egy értékből, egy bal és jobb oldali mutatóból. Létrehozzuk még ezen kívül a BINFA és BINFA\_PTR típusokat. Ha új BINFA\_PTR elemet hozunk létre létrejön BINFA\_PTR típusú p változó és megnézzük, hogy nem üres-e a fa tehát létezik-e az adott csomópont.

//A program forrása: <https://progpater.blog.hu/2011/02/19/> ↵  
gyonyor\_a\_tomor

```
#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;

BINFA_PTR
uj_elem ()
{
 BINFA_PTR p;

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

Deklaráljuk a kiir() és szabadít() függvényeket későbbi használatra, majd a main() függvényben példányosítjuk a gyökér elemet aminek a jelölése '/' lesz. Erre a gyökérelemre ráállítjuk a fa mutatót.

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

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

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

Olvassuk a bemenetet, a kimenetre (standart output) pedig írunk. Megnézzük, hogy 0-t kell-e betenni a fába, ha igen megnézzük az adott csomópontnak van-e nullás gyereke. Olyan esetben, ha nincs ilyen gyerek akkor létrehozunk egyet és ráállítjuk a fa mutatót, majd vissza gyökérre. Már létező gyerek esetén, szimplán csak ráállítjuk a mutatót. Ezt ugyanígy megcsináljuk egyes gyerek esetén is. Az újonnan létrehozott elemek gyermekei nullák.

```
while (read (0, (void *) &b, 1))
{
 write (1, &b, 1);
 if (b == '0')
 {

 if (fa->bal_nulla == NULL)
```

```
{
 fa->bal_nulla = uj_elem ();
 fa->bal_nulla->ertek = 0;
 fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
 fa = gyoker;
}
else
{
 fa = fa->bal_nulla;
}
}

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

A kiir() függvénybe megmérjük a mélységet és kiírjuk a fát inorder feldolgozás szerint, tehát először az egyes elemeket, majd a gyökeret és utoljára pedig a nullás elemeket. A szabadít függvényben felszabadítjuk az adott elemek és gyermekeik által lefoglalt memóriát.

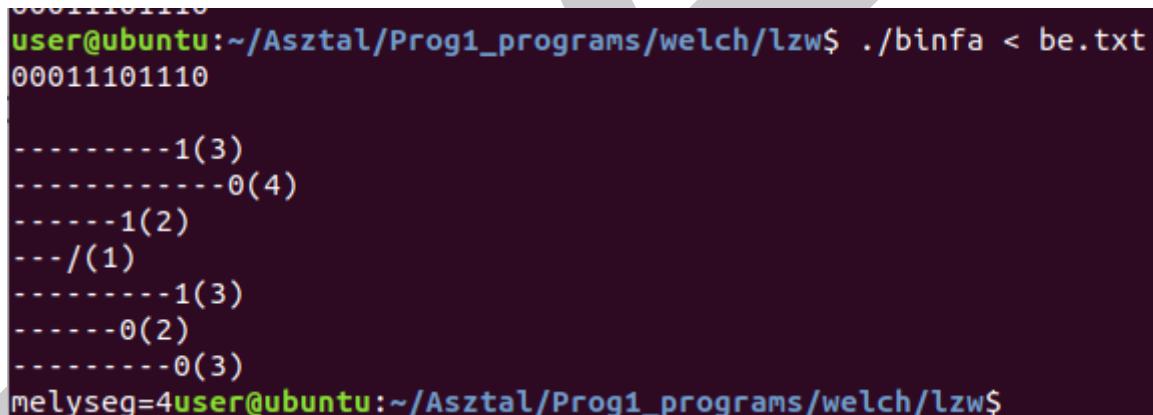
```
printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d", max_melyseg);
szabadit (gyoker);
}

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;
}
}

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



```
00011101110
user@ubuntu:~/Asztal/Prog1_programs/welch/lzw$./binfa < be.txt
00011101110

-----1(3)
-----0(4)
----1(2)
--/(1)
----1(3)
----0(2)
----0(3)
melyseg=4user@ubuntu:~/Asztal/Prog1_programs/welch/lzw$
```

6.2. ábra. A program kimenete

### 6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/blob/master/Welch/Fabejarasok>

A különböző bejárási módokhoz a 4. feladat programját használtam alapul. A feldolgozás módjának megváltóztatásához csak a kiír() függvényben kell változtatásokat végezniünk. A postorder megoldásban először a

egyes gyermeket, majd a nullás gyermeket és legvégül a gyökeret dolgozzuk fel. Jól látható hogy a kiir() függvényt először az egyes gyermekre,a nullas gyermekre hívjuk meg, majd a gyökeret átadjuk az os-nek.

```
void kiir (Csomopont * elem, std::ostream & os)
{
 if (elem != NULL)
 {
 ++melyseg;
 kiir (elem->egyesGyermek (), os);
 kiir (elem->>nullasGyermek (), os);
 for (int i = 0; i < melyseg; ++i)
 os << "---";
 os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;

 --melyseg;
 }
}
```

Preorder feldolgozásnál ennek pontosan ellenkezően járunk el. Először feldolgozzuk a gyökeret,azután az egyes gyemeket majd a nullás gyermeket. Gyakorlatban tehát átadjuk az os-nek a gyökeret, majd meghívjuk a kiir() függvényt az egyes, majd a nullás gyermekre.

```
void kiir (Csmopont * elem, std::ostream & os)
{
 if (elem != NULL)
 {
 ++melyseg;

 for (int i = 0; i < melyseg; ++i)
 os << "---";
 os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
 kiir (elem->egyesGyermek (), os);
 kiir (elem->>nullasGyermek (), os);

 --melyseg;
 }
}
```

|           |           |           |
|-----------|-----------|-----------|
| -----1(2) | -----0(3) | -----/(0) |
| -----0(3) | -----1(2) | -----1(1) |
| -----1(1) | -----0(2) | -----1(2) |
| -----0(2) | -----1(1) | -----0(3) |
| -----/(0) | -----0(5) | -----0(2) |
| -----1(3) | -----0(4) | -----0(1) |
| -----0(4) | -----1(3) | -----1(2) |
| -----0(5) | -----0(3) | -----1(3) |
| -----1(2) | -----1(2) | -----0(4) |
| -----0(3) | -----0(5) | -----0(5) |
| -----0(1) | -----1(4) | -----0(3) |
| -----1(4) | -----0(4) | -----0(2) |
| -----0(5) | -----1(3) | -----1(3) |
| -----1(3) | -----0(6) | -----1(4) |
| -----0(4) | -----0(5) | -----0(5) |
| -----0(2) | -----0(4) | -----0(4) |
| -----0(3) | -----0(3) | -----0(3) |
| -----0(4) | -----0(2) | -----0(4) |
| -----0(5) | -----0(1) | -----0(5) |
| -----0(6) | -----/(0) | -----0(6) |
| depth = 6 | depth = 6 | depth = 6 |

6.3. ábra. Inorder,postorder és postorder bejárások kimenetei

## 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: [https://gitlab.com/tocsika7/prog1/blob/master/Welch/binfa\\_osztalyokkal.cpp](https://gitlab.com/tocsika7/prog1/blob/master/Welch/binfa_osztalyokkal.cpp)

A programot azzal kezdjük, hogy létrehozzuk az LZWBInFa osztályt, aminek a public részébe létrehozunk egy konstruktort, ami ráállítja a fa mutatót a gyökérre. Létrehozzuk ennek a destrukturát is, amiben meghívjuk a szabadít() függvényt a gyökér egyes és nullás elemeire.

```
// z3a7.cpp
//
// Együtt támadjuk meg: http://progater.blog.hu/2011/04/14/ ←
// egyutt_tamadjuk_meg
// LZW fa építő 3. C++ átirata a C változatból (+mélység, atlag és szórás)
// Programozó Páternoszter
//
// Copyright (C) 2011, 2012, Bátfai Norbert, nbatfai@inf.unideb.hu, ←
// nbatfai@gmail.com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
```

```
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.

class LZWBInFa
{
public:

 LZWBInFa () :fa (&gyoker)
 {
 }
 ~LZWBInFa ()
 {
 szabadít (gyoker.egyesGyermek ());
 szabadít (gyoker.nullasGyermek ());
 }
}
```

Tagfüggvényként túlterheljük a beszúró operátort, amivel betölthetjük a fába az inputot. A b formális paraméterként azt jelöli, amit be akarunk a fába tölteni. Megnézzük egy if-el, 0-t kell-e betenni a fába, majd megnézzük van-e az adott csomópontnak 0-s eleme, tehát a fa mutató mutat-e ilyenre. Ha nincs akkor létrehozunk egyet. (A Csomopont osztályból példányosítjuk az új Csomópontot 0-s paraméterrel), majd ezek után ráállítjuk a fa mutatóját az új nullás gyerekére, és visszaállítjuk a mutatót a gyökérre, mert az algoritmus ez követeli meg.

```
void operator<< (char b)
{

 if (b == '0')
```

```
{

 if (!fa->nullasGyermek ())
 {

 Csomopont *uj = new Csomopont ('0');

 fa->ujNullasGyermek (uj);

 fa = &gyoker;
 }
 else
 {

 fa = fa->nullasGyermek ();
 }
}
```

Ellenkező esetben, ha a b nem nulla, akkor ugyanúgy járunk el csak 1-es gyerekre. Tehát megnézzük van-e már 1-es gyereke az adott csomópontnak, ha nincs példányosítunk egyet és ráállítjuk a mutatót. Illetve még deklarálunk egy getMelyseg() függvényt későbbi használatra.

```
else
{

 if (!fa->egyesGyermek ())
 {
 Csomopont *uj = new Csomopont ('1');
 fa->ujEgyesGyermek (uj);
 fa = &gyoker;
 }

 else
 {
 fa = fa->egyesGyermek ();
 }
}
int getMelyseg (void);
```

A kcir() függvényt friend-ként deklaráljuk, hogy hozzá tudjunk félni a program private részeihez is.

```
friend std::ostream & operator<< (std::ostream & os, LZWBInFa & bf)
{
 bf.kcir (os);
 return os;
}
void kiir (std::ostream & os)
{
 melyseg = 0;
 kiir (&gyoker, os);
```

```
}
```

Létrehozzuk a privát Csomópont osztályt. Ennek a konstruktora alapértelmezetten a gyökér betűvel ('/') hozza létre a csomópontot. Ilyet hívunk a fából aki tagként tartalmazza a gyökeret. Egyébként ha valami betűvel hívjuk, akkor azt teszi a betű tagba a két gyermekre mutató mutatókat pedig 0-ra állítja.

```
private:
 class Csomopont
 {
 public:

 Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
 {
 }
 ~Csonopont ()
 {
 };
```

Megnézzük mi az aktuális csomópont bal oldali (nullás) és jobb oldali (egyes) gyereke, majd beállítjuk, hogy legyen az aktuális csomópontnak gy a bal vagy jobb oldali gyereke. Az utolsó részben pedig az nézzük meg milyen betűt hordoz a csomópont.

```
Csonopont *nullasGyermek () const
{
 return balNulla;
}

Csonopont *egyesGyermek () const
{
 return jobbEgy;
}

void ujNullasGyermek (Csonopont * gy)
{
 balNulla = gy;
}

void ujEgyesGyermek (Csonopont * gy)
{
 jobbEgy = gy;
}

char getBetu () const
{
 return betu;
}
```

A private részben a deklaráljuk a változókat és letiltjuk a másoló konstruktort ha nem másolható a Csomópont.

```
private:

 //Milyen betűt hordoz a Csomopont
 char betu;

 Csomopont *balNulla;
 Csomopont *jobbEgy;

 Csomopont (const Csomopont &);
 Csomopont & operator= (const Csomopont &);
```

A fa minden csomópontra megfelelően mutatja az algoritmusnak megfelelő információkat.

```
Csomopont *fa;

int melyseg;

LZWBinFa (const LZWBinFa &);
LZWBinFa & operator= (const LZWBinFa &);
```

Kiiratjuk az os-re az aktuális csomópontot, de csak akkor ha létezik az elem, ha nincs Csomópont leállítjuk a rekurziót. A mélységből levonunk egyet mert a postorder bejárásban képest alapértelmezett 1-el nagyobb. Először az egyes gyermeket dolgozzuk fel, utána a gyökeret, majd a nullás gyermeket az inorder bejárásnak megfelelően. Ha ez megvan felszabadítjuk a szabadít() függvényt a csomópontot, a gyerekeivel kezdve.

```
void kiir (Csomopont * elem, std::ostream & os)
{
 if (elem != NULL)
 {
 ++melyseg;
 kiir (elem->egyesGyermek (), os);

 for (int i = 0; i < melyseg; ++i)
 os << "---";
 os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
 kiir (elem->>nullasGyermek (), os);
 --melyseg;
 }
}

void szabadit (Csomopont * elem)
{
 if (elem != NULL)
 {
 szabadit (elem->eqyesGyermek ());
 }
}
```

```
 szabadit (elem->nullasGyermek ());
 delete elem;
}
}
```

A fában a gyökér kitüntetett elem ezér protected.

```
protected:
 Csomopont gyoker;
 int maxMelyseg;

 void rmelyseg (Csmopont * elem);

};
```

Lekérjük a max mélységet az rmelyseg() függvényben és ezt a getMelyseg() függvényben alkalmazzuk is.

```
int
LZWBinFa::getMelyseg (void)
{
 melyseg = maxMelyseg = 0;
 rmelyseg (&gyoker);
 return maxMelyseg - 1;
}

void
LZWBinFa::rmelyseg (Csmopont * elem)
{
 if (elem != NULL)
 {
 ++melyseg;
 if (melyseg > maxMelyseg)
 maxMelyseg = melyseg;
 rmelyseg (elem->egyesGyermek ());
 // ez a postorder bejáráshoz képest
 // 1-el nagyobb mélység, ezért -1
 rmelyseg (elem->>nullasGyermek ());
 --melyseg;
 }
}
```

A usage() függvénybe kiiratjuk a program helyes használatát és ezt meghívjuk a main()-ben rossz használat (kevés argumentum) esetére. Eltároljuk a a bemenő fájl nevét és megnézzük hogy van-e -o kapcsoló a futtatáskor, ha nincs kiírjuk a helyes használatot és azt hogy nem létezik a bementi fájl.

```
void
usage (void)
```

```
{
 std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

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

 if (argc != 4)
 {

 usage ();

 return -1;
 }

 char *inFile = *++argv;

 if (*((++argv) + 1) != 'o')
 {
 usage ();
 return -2;
 }

 std::fstream beFile (inFile, std::ios_base::in);

 if (!beFile)
 {
 std::cout << inFile << " nem létezik..." << std::endl;
 usage ();
 return -3;
 }
}
```

Dekalráljuk az output fájlt illetve a b-t amivel a fájlból olvassuk a a bájtokat. Binárisan olvassuk a be-menetet, de a kimenő fájlt karakteresen írjuk ki. A b-ben lévő bájt bitjeit egyenként megnézzük és addig maszkolunk amíg az if fejébe a legmagasabb helyiértékű bit vizsgálatát nem írjuk. Ha a vizsgált bit 1, akkor 1 betűt írunk a binfa objektumba, ha 0 akkor 0-t.

```
std::fstream kiFile (*++argv, std::ios_base::out);

unsigned char b;
LZWBinFa binFa;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
 if (b == 0x0a)
 break;
```

```
bool kommentben = false;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
{
 if (b == 0x3e)
 // > karakter
 kommentben = true;
 continue;
}

if (b == 0x0a)
 // újsor
 kommentben = false;
 continue;
}

if (kommentben)
 continue;

if (b == 0x4e) // N betű
 continue;

for (int i = 0; i < 8; ++i)
{
 if (b & 0x80)

 binFa << '1';
 else

 binFa << '0';
 b <= 1;
}

}
```

A kiFile-ban kiírjuk a fát és a mélységet, majd bezárjuk a be és ki fájlokat.

```
kiFile << binFa;

kiFile << "depth = " << binFa.getMelyseg () << std::endl;

kiFile.close ();
beFile.close ();

return 0;
}
```

## 6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: [https://gitlab.com/tocsika7/prog1/blob/master/Welch/binfa\\_mutato\\_gyoker.cpp](https://gitlab.com/tocsika7/prog1/blob/master/Welch/binfa_mutato_gyoker.cpp)

Ahhoz, hogy átírjuk a gyökeret pointer-re annyit kell tennünk, hogy át kell írnunk a a protected részben Csomopont \*gyoker -re. Emellett a konstrukturban új '/' csomópontként vesszük fel a gyökeret és ráállítjuk a fa mutatót is. A destrukturban pedig a gyoker és annak a gyermekei felszabadításáról kell gondoskodni. A végső lépésként ki kell cserélnünk azokat a helyeket, ahol referenciaiként van átadva a gyökér, egyszerű értékkadásra.

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

}

protected:
 Csomopont *gyoker;
```

## 6.6. Mozgató szemantika

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

Tutor: Racs Tamás <https://gitlab.com/cant0r/bhax>

Megoldás videó:

Megoldás forrása: [https://gitlab.com/tocsika7/prog1/blob/master/Welch/binfa\\_move\\_ctor.cpp](https://gitlab.com/tocsika7/prog1/blob/master/Welch/binfa_move_ctor.cpp)

A mozgató szemantika írásához azt a programot használjuk amiben a gyökér mutató volt. Készítünk egy mozgató konstruktort és a mozgató értékkadást. A konstruktorban az eredeti gyökeret kinullázuk, majd az eredeti fa \*this mutatóját ráállítjuk az új fára. Ezt azért tehetjük meg, mert a move() függvényel készítünk belőle rvalue típusú referenciát. A mozgató értékkadásban a swap() függvényel cseréljük, (ami a háttérben a move() függvényel dolgozik) ki a régi és az új gyökeret, illetve az aktuális csomópontra mutató pointereket. Az \*this mutatóval pedig az új fát adjuk vissza.

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

 gyoker = nullptr;
 *this = std::move(regi);

}

LZWBinFa& operator = (LZWBinFa&& regi){
 std::swap(gyoker, regi.gyoker);
 std::swap(fa, regi.fa);
 return *this;

}

kiFile << "Az eredeti fa: \n";
kiFile << binFa;

kiFile << "depth = " << binFa.getMelyseg () << std::endl;

LZWBinFa binFa2 = std::move(binFa);

kiFile <<"A másolt fa: \n";
kiFile << binFa2;

kiFile << "depth = " << binFa2.getMelyseg () << std::endl;

kiFile.close ();
beFile.close ();
```



```
Az eredeti fa:
-----1(2)
-----0(3)
-----1(1)
-----0(2)
/(0)
-----1(3)
-----0(4)
-----0(5)
-----1(2)
-----0(3)
-----0(1)
-----1(4)
-----0(5)
-----1(3)
-----0(4)
-----0(2)
-----0(3)
-----0(4)
-----0(5)
-----0(6)
depth = 6
Az új fa:
-----1(2)
-----0(3)
-----1(1)
-----0(2)
/(0)
-----1(3)
-----0(4)
-----0(5)
-----1(2)
-----0(3)
-----0(1)
-----1(4)
-----0(5)
-----1(3)
-----0(4)
-----0(2)
-----0(3)
-----0(4)
-----0(5)
-----0(6)
depth = 6
```

6.4. ábra. A program kimenete

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyszimulációk

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

A feladatban tutoriáltam: Borvíz Róbert [https://github.com/BorvizRobi/prog\\_1\\_textbook/tree/master/beadando](https://github.com/BorvizRobi/prog_1_textbook/tree/master/beadando)

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

Megoldás forrása: <https://gitlab.com/tocsika7/prog1/tree/master/Conway/Hangya>

Tanulságok, tapasztalatok, magyarázat...

A hangyszimulációs programban a hangyák természetes viselkedését akarjuk szimulálni, miszerint ha a hangyák elhagyják a bojt és szétszélednek, feromonokat bocsátanak ki, amik alapján újra egymásra találnak és utakat képeznek amin haladnak.

Először a program header-jeinek tulajdonságait ismertetem:

Ant.h tulajdonságai:

- oszlop
- sor
- irány

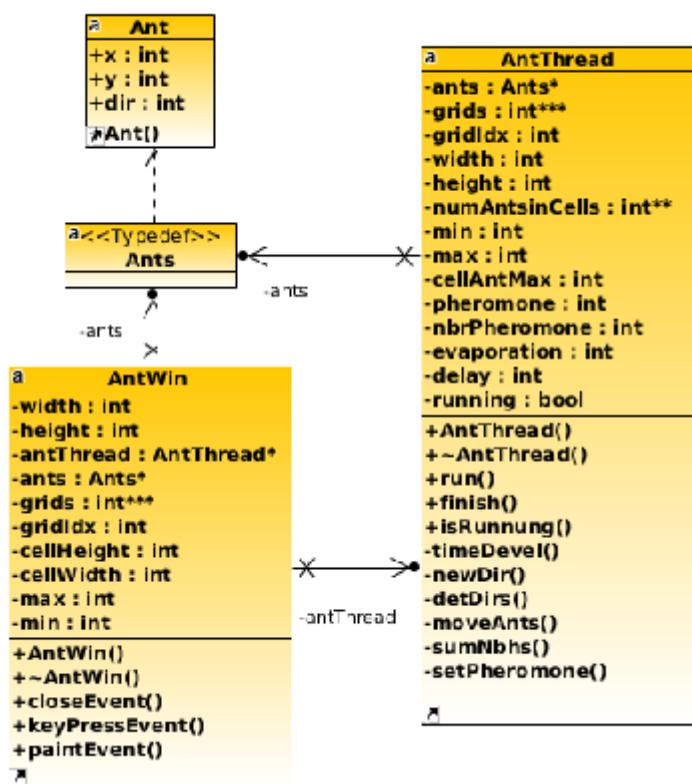
AntWin.h tulajdonságai:

- szélesség
- magasság
- egy szál amivel a hangya számításokat végzi
- két rács tulajdonság
- cellamagasság és szélesség

- maximum és minimum feromon érték

AntThread.h tulajdonságai:

- hangya tulajdonság
- két féle rács tulajdonság
- magasság, szélesség
- hangyákat tartalmazó cellák száma
- feromon
- feromon szám
- feromon párolgás mértéke
- késleltetés
- futó tulajdonság



7.1. ábra. Hangya

Forráskódok elemzése:

### 1. antthread.cpp

forrás:

A programot azzal kezdjük hogy include-oljuk a szükséges header-öket qt-ból, a math.h-t illetve az antthread headert. Létrehozunk egy AntThread objektumot, kellő tulajdonságokkal változónként. A hangyákat tartalmazó celláknak helyet foglalunk memóriában, majd kezdeti számukat beállítjuk 0-ra. Random számot generálunk és a hangyáknak ennek a segítségével adjuk meg majd az oszlopát és sorát. Ha ezek megvannak növeljük a hangyákat tartalmazó cellák értékét és a hangyat betöljtük egy vektorba. A newDir(), detDirs() és a moveAnts() függvényekkel fogjuk a hangyák mozgását módosítani. A newDir()-ben megnézzük, rossz-e az irányuk és a detDirs()-ben pedig beállítjuk hová menjenek. A timeDevel() függvényben azt nézzük meg milyen rácspontok veszik körbe a hangyát, ennek segítségével fogjuk a mozgást és a feromonok kibocsátását is beállítani. A feromon kibocsátást a setPheromon() függvény végezi. A run() függvényben a ha fut program akkor a futtató szál alvását halasztjuk, ha meg van állítva, meghívjuk a timeDevel() függvényt, egyébként kilépünk. Kell még egy destruktort is írnunk, ami felszabadítja a használt memóriát.

### 2. antwin.cpp

Az antwin.cpp-ben a program ablakát hozzuk létre, beállítjuk a nevét, illetve az ablak paramétereit (szélesség, magasság, cellák mérete..stb.). Létrehozunk egy Ants és egy AntThread objektumot majd az AntThread-et elindítjuk. Beépített QT-s függvényekkel beállítjuk a hangyák színét, illetve azt, hogy kellőképpen változzon, amit az ablakon látunk a program futásának megfelelően. A program végére írnunk egy destruktort is amivel a lefoglalt memóriát szabadítjuk fel.

Futtatás:

```
qmake myrmecologist.pro
make myrmecologist
.myrmecologist
```

Licensz:

```
// BHAX Myrmecologist
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// https://bhaxor.blog.hu/2018/09/26/hangyaszimulacioik
```

```
// https://bhaxor.blog.hu/2018/10/10/myrmecologist
//
```

## 7.2. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: [https://gitlab.com/tocsika7/prog1/tree/master/Conway/Eletjatek\\_C++](https://gitlab.com/tocsika7/prog1/tree/master/Conway/Eletjatek_C++)

Tanulságok, tapasztalatok, magyarázat...

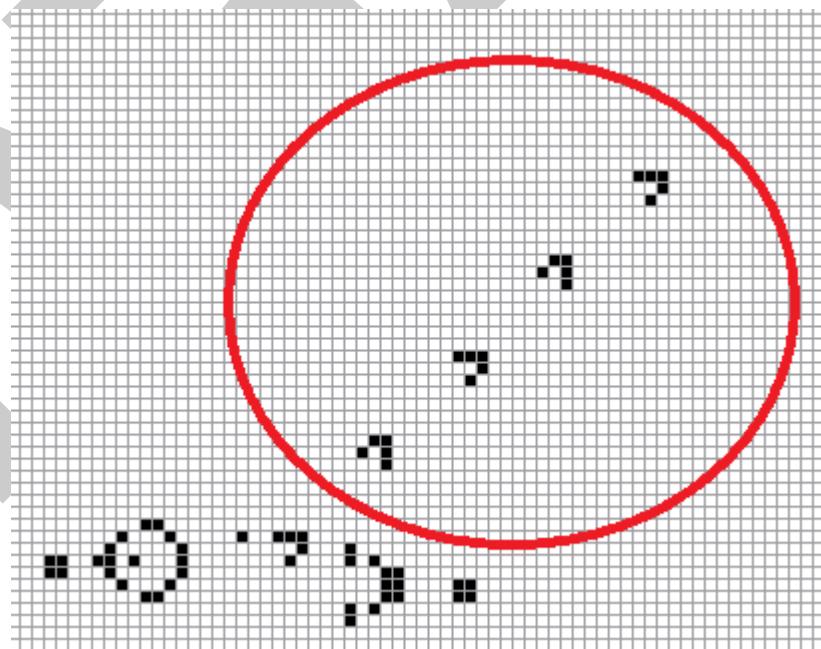
Az életjátékról átlalánosan:

Az életjátéket John Conway a Cambridge-i Egyetem professzora találta ki. Egy olyan játék, amivel nem kell játszani, hiszen csak a kezdőalakzatot adjuk meg, utána a működés automatikus. A játék egy négyzetrácsos táblán zajlik, melynek mezői a sejtek. minden sejtnek vannak szomszédai, ami alatt a szomszédos 8 cellát értjük. A játék elején megadjuk a kezdő sejteket és ezután más dolgunk nincsen. A játék körökből áll, amiben a sejteknek 3 lehetséges változása lehet végbe:

- túlél, ha a szomszédainak száma 2 vagy 3
- elpusztul, ha több mint 3 vagy kevesebb mint 2 szomszédja van
- új sejt jön létre olyan helyen, ahol a szomszédos cellákban 3 sejt van

A sikló ágyú ún. sikló alakzatokat hoz létre, ezeknek az a különleges tulajdonságuk hogy átlósan mozognak és nem pusztulnak el.

A fenti gondolatok forrása: <https://hu.wikipedia.org/wiki/%C3%89letj%C3%A1t%C3%A9k>



7.2. ábra. Sikló alakzat

A programot több részből építjük fel a headerökben deklaráljuk a szükséges függvényeket és változókat, amiket majd később használni fogunk a forráskódokban.

A header-ek tulajdonságai:

#### 1.sejtablak.h:

- A konstruktörben a sejt két lehetséges állapotát adjuk meg. (élő, halott)
- Két rácsot használunk: az egyik a sejtekállapotát jelzi a  $t_n$  pillanatban, a másik a  $t_{n+1}$  pillanatban ez \*\*\*racsok pointer.
- A \*\*racs a valamelyik ilyen rácsállapotra mutat.
- A racsIndex azt mutatja melyik az aktuális rács
- Megadjuk még ezen kívül a cellák, illetve a sejttér magasság és szélességét.

Deklarálunk még pár függvényt melyek a következők: siklo(), sikloKilovo() illetve QT beépített paintEvent().

#### 2.sejtszal.h

- \*\*\*racsok
- sélesség, magasság
- racsIndex
- varakozas, ami a sejt  $t_n$  és  $t_{n+1}$  állapota közötti valós időt jelzi.

Deklaráljuk még a ideFejlodes() és a szomszedokSzama() függvényt.

Forráskódok:

#### sejtablak.cpp

Létrehozunk egy SejtAblak objektumot, beállítjuk az ablak nevét, szélességét, magasságát, cellák paramétereit. Lefoglaljuk a szükséges tárhelyet majd a rács minden celláját halottra állítjuk. Ezután helyezzük le az "élőlényeket" a rácson, tehát meghívjuk a sikloKilovo() függvényt. Létrehozunk egy új sejtszál objektumot és elindítjuk a "játékot". A paintEvent() függvényteljesen rajzoljuk ki az aktuális eseményeket. Végigmegyünk a soron és az oszlopokon és kirajzoljuk az aktuális sejt cellát. A destruktőrben felszabadítjuk a lefoglalt memóriát. A program utolsó részében kidolgozzuk a sikló és siklóKilövő előlényeket létrehozó függvényeket. A sikló az adott irányban halad és másolja magát a sejttérben, a sikló kilövő pedig ilyeneket lő ki.

#### sejtszal.cpp

A sejtszal.cpp programot azzal kezdjük, hogy létrehozunk egy Sejtszál objektumot a szükséges változókkal. A szomszedokSzama() függvényben végigmegyünk a sejt 8 szomszédján a vizsgált sejtet kihagyva. És azt nézzük meg, hogy ezeknek mi az állapota. Az ideFejlodes() függvényben a játékszabályoknak megfelelően alakítjuk a sejt állapotát a lekért szomszédok állapota alapján.

Futtatás:

```
qmake Sejtauto.pro
make Sejtauto
.Sejtauto
```

## Licensz

```
// Életjáték rajzoló
// Programozó Páternoszter

// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail.com

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

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

// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.

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

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

// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
```

## 7.3. Java életjáték

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

Megoldás videó:

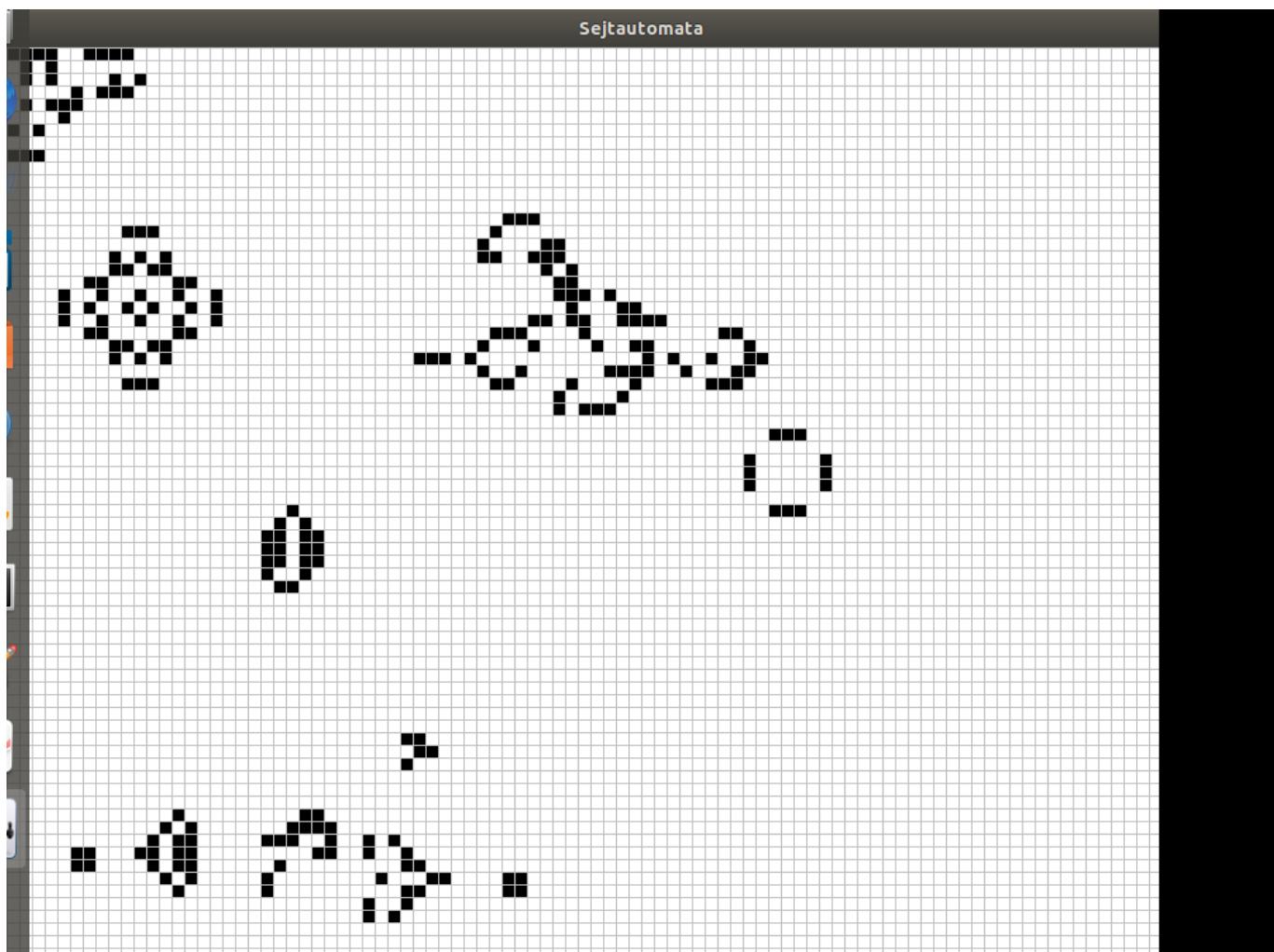
Megoldás forrása: <https://gitlab.com/toesika7/prog1/blob/master/Conway/Sejtautomata.java>

Tanulságok, tapasztalatok, magyarázat...

A Java verzió esetén az egész programot egy osztályba írjuk. Felvesszük a szükséges változókat, illetve még pár plusz változót amik a pillanatképek készítéséhez kellenek majd. Elkészítjuk a rácsot, majd minden cellát halottá állítunk. Létrehozunk egy listenert, ami azt érzékeli, hogy bezárjuk-e az ablakot. Ha bezárjuk, akkor a program is leáll. Egy keyListener segítségével már bizonyos billentyűket is érzékel a program. Változtathatók(k,n billentyűkkel) az cellák méretét vagy akár pillanatképet is készíthetünk(s). Egy MouseListenerrel az egér kattintásokat is vizsgáljuk. Kattintással kijelöljük a nagyítandó területet amit egy négyzettel tudunk majd nagítani. Ennek a négyzetnek a vonzolására létrehozunk egy MouseMotionListener is. Beállítjuk a kezdeti cellaméreteket illetve a program abalakának tulajdonságait. A sejtteret egy külön szalon futtatjuk, a kirajzolására pedig a paint() függvényt használjuk. A setColor() függvénytel állítjuk be sejtek megfelelő színeit. Ha készítettünk pillanatfelvételt akkor a program kikapcsolja nehogy túl sok készüljön. Ezután a jól ismert módon megnézzük a sejt szomszédait majd annak alapján állítjuk be az állapotát. Itt is létrehozunk egy sikloKilovo és siklo() függvényeket amik alapból rajta lesznek a sejtteren. A pillanatfelvételek létrehozására is készítünk egy függvényt ami beépített függvények segítségével létrehoz egy sejtautomata.png képet. A main() függvényben példányosítunk egy Sejtautomata objektumot a megírt osztályból, ennek 100 oszlopa és 75 sora lesz.

Futtatás:

```
javac Sejtautomata.java
java Sejtautomata
```



7.3. ábra. Egy pillanatkép

Forrás:

```
/*
 * Sejtautomata.java
 *
 * DIGIT 2005, Javat tanitok
 * Batfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * Sejtautomata osztaly.
 *
 * @author Batfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class Sejtautomata extends java.awt.Frame implements Runnable {
 /** Egy sejt lehet elo */
 public static final boolean eLo = true;
```

```
/** vagy halott */
public static final boolean HALOTT = false;
/** Ket racsot hasznalunk majd, az egyik a sejtter allapotat
 * a t_n, a masik a t_n+1 idopillanatban jellemzi. */
protected boolean[][][] racsok = new boolean[2][][];
/** Valamelyik racsra mutat, technikai jellegu, hogy ne kelljen a
 * [2][][]-bol az also dimenziot hasznalni, mert vagy az egyikre
 * allitjuk, vagy a masikra. */
protected boolean[][] racs;
/** Megmutatja melyik racs az aktualis: [racsIndex][][] */
protected int racsIndex = 0;
/** Pixelben egy cella adatai. */
protected int cellaSzelesseg = 20;
protected int cellaMagassag = 20;
/** A sejtter nagysaga, azaz hanyoszor hany cella van? */
protected int szelesseg = 20;
protected int magassag = 10;
/** A sejtter ket egymast koveto t_n es t_n+1 diszkret idopillanata
 * kozotti valos ido. */
protected int varakozas = 1000;
// Pillanatfelvetel keszitesehez
private java.awt.Robot robot;
/** Keszitsunk pillanatfelvetelt? */
private boolean pillanatfelvetel = false;
/** A pillanatfelvetelek szamzasahoz. */
private static int pillanatfelvetelSzamlalo = 0;
/**
 * Letrehoz egy <code>Sejtautomata</code> objektumot.
 *
 * @param szelesseg a sejtter szelessege.
 * @param magassag a sejtter szelessege.
 */
public Sejtautomata(int szelesseg, int magassag) {
 this.szelesseg = szelesseg;
 this.magassag = magassag;
 // A ket racs elkeszitese
 racsok[0] = new boolean[magassag][szelesseg];
 racsok[1] = new boolean[magassag][szelesseg];
 racsIndex = 0;
 racs = racsok[racsIndex];
 // A kiindulo racs minden cellaja HALOTT
 for(int i=0; i<rcs.length; ++i)
 for(int j=0; j<rcs[0].length; ++j)
 racs[i][j] = HALOTT;
 // A kiindulo racsra "elolenyeket" helyezunk
 //siklo(rcs, 2, 2);
 sikloKilovo(rcs, 5, 60);
 // Az ablak bezarasakor kilepunk a programbol.
 addWindowListener(new java.awt.event.WindowAdapter() {
 public void windowClosing(java.awt.event.WindowEvent e) {
```

```
 setVisible(false);
 System.exit(0);
 }
});

// A billentyuzetrol erkezo esemenyek feldolgozasa
addKeyListener(new java.awt.event.KeyAdapter() {
 // Az 'k', 'n', 'l', 'g' es 's' gombok lenyomasat figyeljuk
 public void keyPressed(java.awt.event.KeyEvent e) {
 if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
 // Felezuk a cella mereteit:
 cellaSzelesseg /= 2;
 cellaMagassag /= 2;
 setSize(Sejtautomata.this.szelesseg*cellaSzelesseg,
 Sejtautomata.this.magassag*cellaMagassag);
 validate();
 } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
 // Duplazzuk a cella mereteit:
 cellaSzelesseg *= 2;
 cellaMagassag *= 2;
 setSize(Sejtautomata.this.szelesseg*cellaSzelesseg,
 Sejtautomata.this.magassag*cellaMagassag);
 validate();
 } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S) {
 pillanatfelvetel = !pillanatfelvetel;
 } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
 varakozas /= 2;
 else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
 varakozas *= 2;
 repaint();
 }
});
// Eger kattinto esemenyek feldolgozasa:
addMouseListener(new java.awt.event.MouseAdapter() {
 // Eger kattintassal jeloljuk ki a nagyitando teruletet
 // bal felso sarkat vagy ugyancsak eger kattintassal
 // vizsgaljuk egy adott pont iteracioit:
 public void mousePressed(java.awt.event.MouseEvent m) {
 // Az egermutato pozicioja
 int x = m.getX()/cellaSzelesseg;
 int y = m.getY()/cellaMagassag;
 racsok[racsIndex][y][x] = !rcsok[racsIndex][y][x];
 repaint();
 }
});
// Eger mozgas esemenyek feldolgozasa:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
 // Vonszolossal jeloljuk ki a negyzetet:
 public void mouseDragged(java.awt.event.MouseEvent m) {
 int x = m.getX()/cellaSzelesseg;
 int y = m.getY()/cellaMagassag;
```

```
 racsok[racsIndex][y][x] = eLo;
 repaint();
 }
});
// Cellameretek kezdetben
cellaSzelesseg = 10;
cellaMagassag = 10;
// Pillanatfelvetel készítésehez:
try {
 robot = new java.awt.Robot(
 java.awt.GraphicsEnvironment.
 getLocalGraphicsEnvironment().
 getDefaultScreenDevice());
} catch(java.awt.AWTException e) {
 e.printStackTrace();
}
// A program ablakanak adatai:
setTitle("Sejtautomata");
setResizable(false);
setSize(szelesseg*cellaSzelesseg,
 magassag*cellaMagassag);
setVisible(true);
// A sejtter eletrekeltese:
new Thread(this).start();
}
/** A sejtter kirajzolása. */
public void paint(java.awt.Graphics g) {
 // Az aktualis
 boolean [][] racs = racsok[racsIndex];
 // racsot rajzoljuk ki:
 for(int i=0; i<rcs.length; ++i) { // vegig leped a soron
 for(int j=0; j<rcs[0].length; ++j) { // s az oszlopok
 // Sejt cella kirajzolása
 if(rcs[i][j] == eLo)
 g.setColor(java.awt.Color.BLACK);
 else
 g.setColor(java.awt.Color.WHITE);
 g.fillRect(j*cellaSzelesseg, i*cellaMagassag,
 cellaSzelesseg, cellaMagassag);
 // Racs kirajzolása
 g.setColor(java.awt.Color.LIGHT_GRAY);
 g.drawRect(j*cellaSzelesseg, i*cellaMagassag,
 cellaSzelesseg, cellaMagassag);
 }
 }
 // Készítünk pillanatfelvetelt?
 if(pillanatfelvetel) {
 // a biztonság kedveért egy kép készítése után
 // kikapcsoljuk a pillanatfelvetelt, hogy a
 // programmal ismerkedő olvasó ne irja tele a
```

```
// fajlrendszeret a pillanatfelvetelekkel
pillanatfelvetel = false;
pillanatfelvetel(robot.createScreenCapture
 (new java.awt.Rectangle
 (getLocation().x, getLocation().y,
 szelesseg*cellaSzelesseg,
 magassag*cellaMagassag)));
}

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

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

 if(racs[s] [o] == allapot)
 ++allapotuSzomszed;
 }
 return allapotuSzomszed;
}
/***
 * A sejtter idobel fejlodese a John H. Conway fele
 * eletjatek sejtautomata szabalyai alapjan tortenik.
 * A szabalyok reszletes ismerteteset lasd peldaul a
```

```
* [MATEK JaTeK] hivatkozasban (Csakany Bela: Diszkret
* matematikai jatekok. Polygon, Szeged 1998. 171. oldal.)
*/
public void idoFejlodes() {

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

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

 int elok = szomszedokSzama(racsElotte, i, j, eLo);

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

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

 idoFejlodes();
 repaint();
 }
}
/**
 * A sejtterbe "elolenyeket" helyezunk, ez a "siklo".
 * Adott irányban halad, masolja magat a sejtterben.
 * Az eloleny ismerteteset lasd peldaul a
 * [MATEK JaTeK] hivatkozasban (Csakany Bela: Diszkret
 * matematikai jatekok. Polygon, Szeged 1998. 172. oldal.)
```

```
*
* @param racs a sejtter ahova ezt az allatkat helyezzuk
* @param x a befoglalo tegla bal felső sarkának oszlopa
* @param y a befoglalo tegla bal felső sarkának sora
*/
public void siklo(boolean [][] racs, int x, int y) {

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

}
/**
 * A sejtterbe "elolenyeket" helyezünk, ez a "siklo agyu".
 * Adott irányban siklokat ló ki.
 * Az eloleny ismertetését lásd például a
 * [MATEK JaTeX] hivatkozásban /Csakány Béla: Diszkret
 * matematikai játékok. Polygon, Szeged 1998. 173. oldal./,
 * de itt az abra hibás, egy oszloppal tölt meg balra a
 * bal oldali 4 sejtes negyzetet. A helyes agyu rajzat
 * lásd pl. az [eLET CIKK] hivatkozásban /Robert T.
 * Wainwright: Life is Universal./ (Megemlíthetjük, hogy
 * mindenkető tartalmaz két felesleges sejtet is.)
 *
 * @param racs a sejtter ahova ezt az allatkat helyezzük
 * @param x a befoglalo tegla bal felső sarkának oszlopa
 * @param y a befoglalo tegla bal felső sarkának sora
 */
public void sikloKilovo(boolean [][] racs, int x, int y) {

 racs[y+ 6][x+ 0] = eLo;
 racs[y+ 6][x+ 1] = eLo;
 racs[y+ 7][x+ 0] = eLo;
 racs[y+ 7][x+ 1] = eLo;

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

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

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

 racs[y+ 6][x+ 11] = eLo;
 racs[y+ 6][x+ 15] = eLo;
 racs[y+ 6][x+ 16] = eLo;
```

```
racs[y+ 6][x+ 22] = eLo;
racs[y+ 6][x+ 23] = eLo;
racs[y+ 6][x+ 24] = eLo;
racs[y+ 6][x+ 25] = eLo;

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

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

racs[y+ 9][x+ 13] = eLo;
racs[y+ 9][x+ 21] = eLo;
racs[y+ 9][x+ 22] = eLo;
racs[y+ 9][x+ 23] = eLo;
racs[y+ 9][x+ 24] = eLo;
racs[y+ 9][x+ 34] = eLo;
racs[y+ 9][x+ 35] = eLo;

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

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

}

/** Pillanatfelvetelek keszítése. */
public void pillanatfelvetel(java.awt.image.BufferedImage felvetel) {
 // A pillanatfelvetel kép fajlneve
 StringBuffer sb = new StringBuffer();
 sb = sb.delete(0, sb.length());
 sb.append("sejtautomata");
 sb.append(++pillanatfelvetelszamlalo);
 sb.append(".png");
 // png formátumú képet mentünk
 try {
 javax.imageio.ImageIO.write(felvetel, "png",
 new java.io.File(sb.toString()));
 } catch(java.io.IOException e) {
 e.printStackTrace();
 }
}
```

```
 }
 }
 // Ne villogjon a felület (mert a "gyari" update()
 // lemeszelne a vasszon felületet).
 public void update(java.awt.Graphics g) {
 paint(g);
 }
 /**
 * Peldanyosít egy Conway-féle eletjátek szabalyos
 * sejtter objektumot.
 */
 public static void main(String[] args) {
 // 100 oszlop, 75 sor merettel:
 new Sejtautomata(100, 75);
 }
}
```

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: [https://gitlab.com/tocsika7/prog1/tree/master/Conway/Brain\\_B](https://gitlab.com/tocsika7/prog1/tree/master/Conway/Brain_B)

Tanulságok, tapasztalatok, magyarázat...

A programról általánosan:

A BrainB egy olyan program, ami felméri az emberek agyának kognitív képességeit egy játék segítségével. A program négyzeteket hoz létre bennük körökkel és a Samu nevű négyzet körén kell tartanunk az egeret, amíg a pozíciója változik és új négyzetek is megjelennek. A program több részből áll: 2 header-ből amiben deklaráljuk a szükséges osztályokat és függvényeket és az ehhez tartozó programokból, illetve egy .pro fájlból amiben átadjuk ezeket a header-öket és forrásprogramokat a QT-nak.

Header-ök tulajdonságai:

1. BrainBThread.h

Typdef-e készítünk egy Hero típusú vektort. A Hero osztály tulajdonságai a következők:

- x
- y
- szín
- sebesség
- cond
- név string

Készítünk egy konstruktort, amivel a SamuEntropy nevű hero-t hozzuk majd létre. A move() függvényben a hősöknek adunk egy random mozgásirányt. OpenCV segítségével hozzuk létre a hősöket jelölő négyzeteket. Deklarálunk még változókat, függvényeket, és vektorokat.

Vektorok:

- lostBPS
- foundBPS

Függvények:

- delay()
- devel()
- noHeroes(), a hősök atkuális száma
- mean(), ami kiszámolja az adott vektorelemek átlagát
- var(), ami az adott vektorelemek, gyökének az átlagát adja.
- get\_bps(), get\_w()
- get\_paused(), get\_nofPaused()
- decComp(), ami csökkenti a hős gyorsaságát, ha elér egy értéket.
- get\_bps(), get\_w()
- incComp és draw(), amivel az új hősöket létrehozzuk, a drawban()
- getT() és finish() ami időmérő függvények.

## 2. BrainBWin.h

Ebben a header programban is deklaráljuk a szükséges változókat és függvényeket. Néhány függvényt átemelünk az előző header-ből, ezek a mean() és a var() függvények. Használunk még beépített QT eseménykezelő függvényeket, amik pl. frissítik majd a program ablakát, érzékelik az egér mozgását, kattintást illetve a billentyűzetet. Lesz még egy függvényünk, ami időértékeket konvertál (millisec2minsec). A save() függvényben, a pillanatfelvétel készítésre adunk lehetőséget illetve a egy file-ba írunk statisztikákat a játék futása közben.

Forráskódok:

BrainBThread.cpp

A BrainThread.cpp-ben létrehozunk egy BrainThread objektumot és készítünk 5 herot, amiket aztán bele töltünk a heroes vektorba. Elkezdjük a futtatást és meghívjuk még a pause() és set\_paused() függvényeket a megállításra.

BrainBWin.cpp

Létrehozzuk a program ablakát. Az endAndStats() függvényel tudjuk a játékossal hogy a stat-jait kiírtuk egy fájlba és ott megtalálhatja őket. Az updateHeroes() függvényel a hősök akutális helyzetét frissítjük.(elveszette-e a játékos,stb.) Egy paintEvent() függvényel kiiratjuk, hogy mi a játékos dolga,az időt, illetve frissítjük a programablakot. Egérkattintással tudjuk megállítani a játékot és felengedésre pedig elindul. Ezt egy mousePress- és egy mouseReleaseEvent függvényel érjük el. Lesz még egy egér mozgását érzéklelő függvény is.

Licensz:

```
/**
 * @brief Benchmarking Cognitive Abilities of the Brain with Computer Games
 *
 * @file BrainBWin.cpp
 * @author Norbert Bátfai <nbatfai@gmail.com>
 * @version 6.0.1
 *
 * @section LICENSE
 *
 * Copyright (C) 2017, 2018 Norbert Bátfai, nbatfai@gmail.com
 *
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program. If not, see <http://www.gnu.org/licenses/>.
 *
 * @section DESCRIPTION
 */
```

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

Python

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

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

Tanulságok, tapasztalatok, magyarázat...

A programról általánosan:

A prgoram lényege, hogy egy mesterséges intelligenciát megtanítunk arra, hogy a beérkező képekről le tudja olvasni, a kézzel írott számot. A TensorFlow a Google gépi tanulást segítő könyvtára, az ilyen könyvtárak közül talán a legnépszerűbb. Többféle nyelven lehet használni, illetve lehet futtatni CPU-n és GPU-n egyaránt (pl. CUDA). Mi a szükséges kódot Python-ban fogjuk írni, mert abban a legegyszerűbb. Gráfokkal fogunk dolgozni amiket majd, betölünk a TensorFlow-ba. Az MNIST egy hatalmas méretű adatbázis, ami képeket tartalmaz kézzel írott, számokról és sok gépi tanulást alkalmazó programban használják, mi is ezt fogjuk használni. A hálózat tanításához 60.000 db képet használunk majd a teszteléshez pedig, ezektől különböző 10.000 db-ot.

Forráskód:

A programban meghívjük a szükséges könyvtárakat, majd létrehozzuk a tensor modellt. Ez a modellt áll egy placeholderból (x), amiben azt tároljuk mekkora képeket várunk majd a tanításához (Ez esetben 784 byte). Deklarálunk két változót is, amik a súlyt és a bias-t tartalmazzák (W,b). Az y-ban az x-el összeszorozzuk a súlyt majd hozzáadjuk a bias-t. A cross-entropy részben azt érjük el, hogy minnél jobb legyen a tanítás, jobb becslést akarunk elérni. Elkezdjük a hálózat tanítását, majd leteszteljük az MNIST adatbázis néhány képével.

```
Copyright 2015 The TensorFlow Authors. All Rights Reserved.
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
```

```

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
↵
=====

Norbert Batfai, 27 Nov 2016
Some modifications and additions to the original code:
https://github.com/tensorflow/tensorflow/blob/r0.11/tensorflow/examples/ ↵
tutorials/mnist/mnist_softmax.py
See also http://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol
↵
=====

"""A very simple MNIST classifier.

See extensive documentation at
http://tensorflow.org/tutorials/mnist/beginners/index.md
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse

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

import tensorflow as tf

import matplotlib.pyplot

FLAGS = None

def readimg():
 file = tf.read_file("sajat8a.png")
 img = tf.image.decode_png(file)
 return img

def main(_):
 mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

 # Create the model
```

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

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

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

Test trained model
print("-- A halozat tesztelese")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("-- Pontossag: ", sess.run(accuracy, feed_dict={x: mnist.test.←
 images,
 y_: mnist.test.labels}))
print("-----")

print("-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a ←
 tovabbelpeshez 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 saját kezi 8-asom felismerése, mutatom a számot, a ←
továbblepeshez 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()

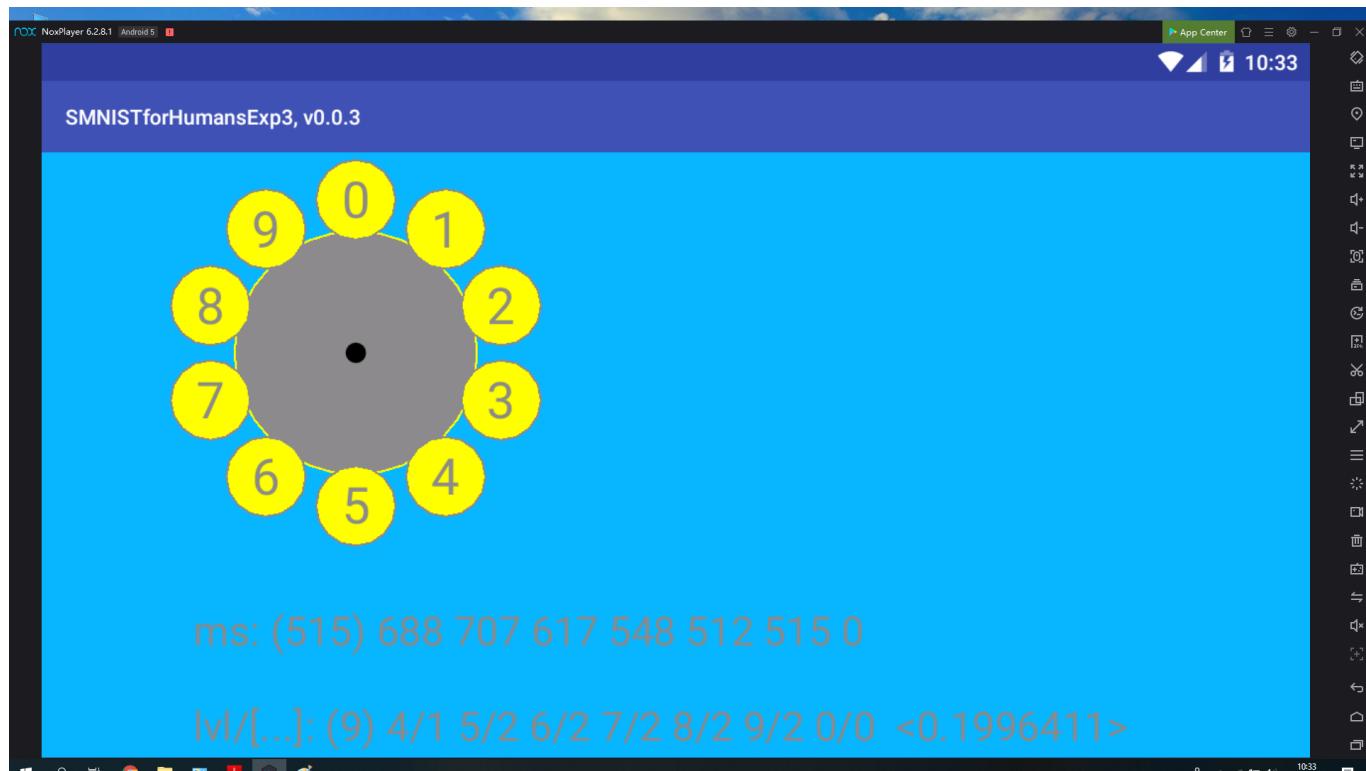
classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

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 további két feladatot az SMNIST for Humans kutatás-támogatás értelmében passzolom.





8.1. ábra. LVL9. Android Emulatorral Windowson

## 8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9. fejezet

# Helló, Chaitin!

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

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

Megoldás forrása: [https://gitlab.com/tocsika7/prog1/tree/master/Chaitin/Lisp\\_Faktori%C3%A1lis](https://gitlab.com/tocsika7/prog1/tree/master/Chaitin/Lisp_Faktori%C3%A1lis)

Megoldás rekurzívan:

Definiáljuk a fakt függvényt n paraméterrel. Egy if-el megnézzük kisebb-e az n, mint 1. Ha nem, akkor megszorozzuk n-1-el, amit a rekurzívan meghívott fakt függvénytel érjük el. Fontos megjegyezni hogy lisp-ben a műveleteket preorder módon használjuk.

```
(define (fakt n) (if (< n 1) 1 (* n (fakt (- n 1)))))
```

Megoldás iteratívan:

Az iteratív megoldásban is használunk rekurziót, de igazából mégsem mert az alap függvény nem tartalmaz. Készítünk egy segédfüggvényt a faktoriális függvényen belül két paraméterrel (num, count); Megnézzük a count nagyobb-e ,mint a megeadott szám ha igen visszadjuk a num változót. Ha nem elkezdjük a rekurziót tehát a számot szorozzuk a számlálóval és a számlálót növeljük. Ez addig fog menni amíg a számláló nem éri el az alapfüggvényben megadott szám értékét.

```
(define (fakt n)
(define (segéd num count)
 (if (> count n)
 num
 (segéd (* num count) (+ count 1)))
)
)
(segéd 1 1))
```

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

A program első részében deklarálunk néhány függvényt: A color-curve függvényben eltárolunk egy tömbben 8 értéket, amik a GIMP színgörbékének bizonyos értékei. Az elem függvénnyel az kérjük le egy listáról, hogy a megadott elem hányadik. A lisp-ben a változók lehetnek listák tehát több elemük is lehet. A wh függvényben az adott szöveg magasságát és szélességét kérjük le.

```
; bhax_chrome3.scm
;
; BHAX-Chrome creates a chrome effect on a given text.
; Copyright (C) 2019
; Norbert Bátfai, batfai.norbert@inf.unideb.hu
; Nándor Bátfai, batfai.nandi@gmail.com
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with this program. If not, see <https://www.gnu.org/licenses/>.
;
; Version history
;
; This Scheme code is partially based on the Gimp tutorial
; http://penguinpeta.com/b2evo/index.php?p=351
; (the interactive steps of this tutorial are written in Scheme)
;
; https://bhaxor.blog.hu/2019/01/10/ ←
; a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv
;

(define (color-curve)
 (let* (
 (tomb (cons-array 8 'byte))
)
 (aset tomb 0 0)
 (aset tomb 1 0)
 (aset tomb 2 50)
 (aset tomb 3 190)
 (aset tomb 4 110)
 (aset tomb 5 20)
 (aset tomb 6 200)
```

```
(aset tomb 7 190)
tomb)
)

;(color-curve)

(define (elem x lista)

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

)

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

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

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

Létrehozzuk a chrome függvényt amelynek a paraméterei:

- szöveg
- betűtípus
- betűméret
- szélesség
- magasság
- szín
- színátmenet

A további részekben lépésenként haladunk a szöveg változtatásával amelyek a következők:

- 1: Létrehozzuk egy fekete háttérreget egy fehér középre toltszöveggel.
- 2: Egy erős gauss elmosódást rakunk a képre.
- 3: A színek határszintjének alsó és felső határának megváltoztatásával élesítünk az elmosódott képen.

- 4: Újra alakalmazzuk a gauss elomsást.
- 5: Szín szerinti kijelöléssel kijelöljük a fekete területet.
- 6: Létrehozunk majd kijelölünk egy átlásztó réteget.
- 7: Az előbb létrehozott átlásztó réteget kitöljük egy sötétszürkéből világosszürkébe való színátmenettel.
- 8: Bucka leképezést alkalmazunk a második réteget az első réteg használatával.
- 9: A színgörbéket módosítjuk, ezeken egy hullám mintát állítunk be.

```
(define (script-fu-bhax-chrome text font fontsize width height color ←
 gradient)
(let*
 (
 (image (car (gimp-image-new width height 0)))
 (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
 LAYER-MODE-NORMAL-LEGACY)))
 (textfs)
 (text-width (car (text-wh text font fontsize)))
 (text-height (elem 2 (text-wh text font fontsize)))
 (layer2)
)
;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND)
(gimp-context-set-foreground '(255 255 255))

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

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

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

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

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

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

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

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

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

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

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

A program végén elmentjük a GIMP menüjébe a scriptet és adunk neki alapértelmezett paramétereket.

```
(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"
"
```



9.1. ábra. A program kipróbálása

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

A mandalát elforgatott kézírott szövegekből készítjük, rakunk rá két kört keretnek és a közepére is írhatunk szöveget.

Forráskód:

A programot néhány függvény deklarálásával kezdjük. Fontos megjegyezni, hogy lisp-ben a változók listák tehát egyszerre több elemük is lehet. Az elem függvényt azt kérjük le egy listáról, hogy a megadott elem hárnyadik. Lekérjük a szöveg szélességét a text-width függvényrel. Alapból a beépített gimp függvény több értéket ad vissza, ezért használjuk a car függvényt, hogy csak a legelső a paramétert adjuk vissza tehát a szélességet. A text-wh függvényrel egy hasonló módszerrel egy listába lekérjük az adott szöveg szélességét és magasságát.

```
; bhax_mandala9.scm
;
; BHAX-Mandala creates a mandala from a text box.
; Copyright (C) 2019 Norbert Bátfai, batfai.norbert@inf.unideb.hu
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU General Public License as published by
```

```
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with this program. If not, see <https://www.gnu.org/licenses/>.
;
; Version history
;
; This Scheme code is partially based on the Python code
; Pat625_Mandala_With_Your_Name.py by Tin Tran, which is released under ↪
; the GNU GPL v3, see
; https://gimplearn.net/viewtopic.php? ↪
; Pat625-Mandala-With-Your-Name-Script-for-GIMP?t=269&p=976
;
; https://bhaxor.blog.hu/2019/01/10/ ↪
; a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

(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 mandala script definiálása 8 paraméterrel, amik a következők: szöveg amiből készül, szöveg középen, betűtípus, betű méret, szélesség, magasság, szöveg színének RGB kódja, színátmennet. A lisp-ben a változókat a let\* függvény segítségével definiáljuk. RGB képet hozunk létre a megadott szélességgel és magassággal. Elkészítjük a háttérreget illetve a szükséges szövegeket. Beállítunk egy előtérszínt amivel a háttérét kitöljtjük, majd értékül adjuk neki a paraméterként kapott színt. Beállítjuk ezen kívül a betűméretet és betűsítlust.

```
(define (script-fu-bhax-mandala text text2 font fontsize width height ←
 color gradient)
(let*
(
 (image (car (gimp-image-new width height 0)))
 (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
 LAYER-MODE-NORMAL-LEGACY)))
 (textfs)
 (text-layer)
 (text-width (text-width text font fontsize)))
;;
 (text2-width (car (text-wh text2 font fontsize)))
 (text2-height (elem 2 (text-wh text2 font fontsize)))
;;
 (textfs-width)
 (textfs-height)
 (gradient-layer)
)
(gimp-image-insert-layer image layer 0 0)

(gimp-context-set-foreground '(0 255 0))
(gimp-drawable-fill layer FILL-FOREGROUND)
(gimp-image-undo-disable image)

(gimp-context-set-foreground color)

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
 height 2))
(gimp-layer-resize-to-image-size textfs)
```

Elforgatjuk a szövegeket úgy hogy a pi adott hatványaival szorozzuk fokszámukat.

```
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
 (gimp-image-insert-layer image text-layer 0 -1)
 (gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
 (set! textfs (car (gimp-image-merge-down image text-layer ←
 CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
 (gimp-image-insert-layer image text-layer 0 -1)
 (gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
 (set! textfs (car (gimp-image-merge-down image text-layer ←
 CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
 (gimp-image-insert-layer image text-layer 0 -1)
 (gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
 (set! textfs (car (gimp-image-merge-down image text-layer ←
 CLIP-TO-BOTTOM-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
 (gimp-image-insert-layer image text-layer 0 -1)
 (gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
 (set! textfs (car (gimp-image-merge-down image text-layer ←
 CLIP-TO-BOTTOM-LAYER)))
```

Elipszis kivágásokkal létrehozzuk a kereteket.

```
(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
 (set! textfs-width (+ (car (gimp-drawable-width textfs)) 100))
 (set! textfs-height (+ (car (gimp-drawable-height textfs)) 100))

 (gimp-layer-resize-to-image-size textfs)

 (gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
 (/ textfs-width 2)) 18)
 (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
 textfs-height 36))
 (plug-in-sel2path RUN-NONINTERACTIVE image textfs)

 (gimp-context-set-brush-size 22)
 (gimp-edit-stroke textfs)

 (set! textfs-width (- textfs-width 70))

 (set! textfs-height (- textfs-height 70))
```

```
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
 (/ textfs-width 2)) 18)
 (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
 textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)
```

Beállítjuk a színátmenet, majd létrehozzuk a középen elhelyezett szöveget.

```
(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE "←
 gradient" 100 LAYER-MODE-NORMAL-LEGACY)))

(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
 GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ (/ ←
 width 2) (/ textfs-width 2)) 8) (/ height 2))

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)
```

```
(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
)))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
 height 2) (/ text2-height 2)))

; (gimp-selection-none image)
; (gimp-image-flatten image)

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

Regisztráljuk a GIMP fájlmenüjébe a scriptet és beállítunk alapértelmezett paramétereket.

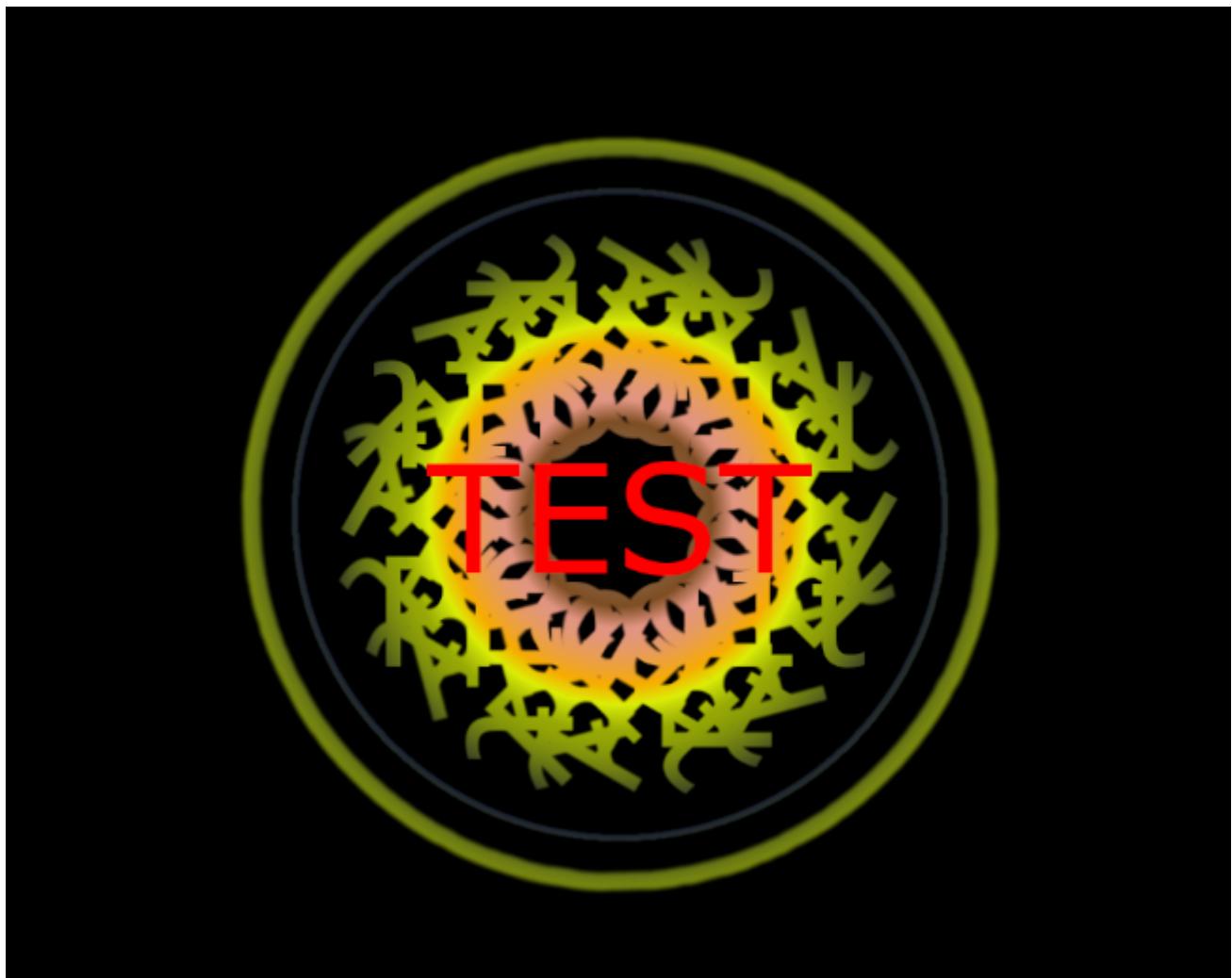
```
; (script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Ruge Boogie" 120 1920 ←
 1080 '(255 0 0) "Shadows 3")

(script-fu-register "script-fu-bhax-mandala"
 "Mandala9"
```

```
"Creates a mandala from a text box."
"Norbert Bátfai"
"Copyright 2019, Norbert Bátfai"
"January 9, 2019"
"
SF-STRING "Text" "Bátf41 Haxor"
SF-STRING "Text2" "BHAX"
SF-FONT "Font" "Sans"
SF-ADJUSTMENT "Font size" '(100 1 1000 1 10 0 1)
SF-VALUE "Width" "1000"
SF-VALUE "Height" "1000"
SF-COLOR "Color" '(255 0 0)
SF-GRADIENT "Gradient" "Deep Sea"
)
```

```
(script-fu-menu-register "script-fu-bhax-mandala"
 "<Image>/File/Create/BHAX"
)
```

DRAFT



9.2. ábra. A program kipróbálása

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Programozási alapfogalmak

[?]

A programozási nyelveket több szempont szerint csoportosíthatjuk. Ha aszerint nézzük, hogy mennyire férünk hozzá a gép hardveréhez a kóddal, akkor 3 szintet különbözünk meg: a gépi kódot, assembly és a magas szintű nyelveket. A magas szintű nyelven írt kódot forráskódnak hívjuk, és ennek a felépítését nyelvtani és tartalmi szabályok együttese határozza meg, amit hivatkozási nyelvnek nevezünk. A processzor alapvetően gépi kódokat tud értelmezni, ezért a magas szintű nyelven írt kódokat valamilyen módszerrel át kell alakítanunk, hogy futtatni tudjuk. Erre vagy fordítóprogramos (pl. C) vagy interpreteres (pl. Python) megoldást használunk, esetenként mindenből. A fordítóprogram a kódból csinál egy tárgyprogramot, amit egységként kezel és bizonyos lépésekkel gépi kódot készít belőle. Az interpreteres verzióban nincs tárgyprogram, soronként értelmezi és végrehajtja a programot. A különböző platformokra megírt fordítóprogramokat és interpretek implementációknak nevezzük, ezeknek a nagy problémája a kompatibilitás egymással, illetve akár a hivatkozási nyelvel is. Aszerint is lehet a nyelveket csoportosítani, hogy algoritmussal vagy anélkül (pl. probléma megadással) írjuk le őket, eszerint léteznek az imperatív nyelvek (pl. objektumorientált nyelvek) és a deklaratív nyelvek (pl. logikai nyelvek).

Az adattípusok absztrakt programozási eszközök, amiket valamilyen konkrét programozási eszköz részeként használunk. Vannak olyan nyelvek, amelyek használnak adattípusokat és vannak, amelyek nem. Hárrom fő tulajdonságuk van: a tartomány azon részei az adatszerkezetnek, amelyeket a konkrét programozási eszközök felvethetnek értékül. A műveletek tulajdonság, az ilyen tartomány elemekkel végezhető műveleteket jelöli. A reprezentáció tulajdonság pedig azt jelöli hol tároljuk a memoriában és mekkora méretben. Egy nyelvben általában előre meg vannak határozva a használható adattípusok, de néhány esetben lehetőség van saját típusok létrehozására is. (pl. C). Lehetőség van arra is, hogy egy adattípusból létrehozzunk egy altípust a tartományának csökkentésével. Az adattípusokat alapvetően két nagy csoportra oszthatjuk: az egyszerű típusúak (pl. egész, valós, logikai) tartománya olyan értékeket tartalmaz, amelyek egyediek, az összetett típusúak (pl. tömb, rekord, mutató) tartományának értekeinek típusa van.

A nevesített konstans egy olyan eszköz, amit akkor használunk, ha egy értéket sokat kell majd használni a program során, de változtatnunk nem kell rajta. Hárrom részből áll, van neve típusa és érteke.

A változóknak már 4 tulajdonsága van. Ugyanúgy van értéke és neve, de már azt is megadhatjuk, hol helyezkedik el a memoriában, ami a cím tulajdonság, emellett van egy attribútum tulajdonsága ilyen pl. a

típus. Az attribútumokat deklaráció során állítjuk be a legtöbb esetben. A deklaráció lehet manuális (explicit), betűkhöz rendelhetünk attribútumokat és az adott kezdőbetűjű változók kapják meg őket (implicit) és történhet a fordítóprogram által (automatikus). A címkiosztásnak is több módja van: lehet már futás előtti (statikus), végezheti a rendszer a program adott állapotától függően (dinamikus) és megadhatjuk mi is. A kezdőérték utasítás is lehet különböző megadhatjuk manuálisan, de néhány esetben, ha ezt nem tesszük meg akkor a fordítóprogram ad meg egy címet, hogy nem memóriaszemét legyen a helyén. C-ben az egyszerű típusokat aritmetikainak az összetett típusokat származtatottnak nevezzük. Fontos megemlíteni, hogy a karakter típusoknak van egy beépített szám értéke, illetve a logikai típus nem elérhető, és a logikai kifejezések egészekkel vannak megoldva (0-igaz, minden más-hamis). A rekordokat általában struktúrákkal ábrázoljuk.

A kifejezéseket arra használjuk, hogy adott értékeknek új értékeket adjunk a programban. Állnak operandumokból, amik lehetnek változók, konstansok és függvényhívások, illetve operátorokból, amik a műveleti jelek. Szükség esetén zárójeleket is használhatunk. Az kifejezéseket megkülönböztetjük aszerint, hogy hány operandust használnak. Ezen kívül alakjuk is változhat lehet: prefix (\* a b), infix (a \* b), postfix (a b \*). A magas szintű nyelveben általában infix alakot használjuk, de például a lisp nyelv a prefix alakot használja. A műveletek feldolgozási sorrendje is alakfüggő, infix-ben ez nem egyértelmű ezért használunk precedencia táblázatot és zárójeleket. Néhány nyelvben léteznek úgynevezett rövidzár operátorok, amiknél nem kell az egész kifejezést kiértékelni, hogy megkapjuk az eredményt. (pl. és operátor). Nyelvenként változik az is, hogy egy kifejezésnél csak azonos típusú vagy különböző típusú operandusokkal végzünk műveleteket. Utóbbi esetben lényeges megemlíteni, hogy ez típuskonverzió segítségével jön lére.

Az utasítások a programozási nyelvek nélkülezhetetlen elmei. Két nagy csoportjuk van a deklarációs és a végrehajtandó utasítások. A deklarációs utasítások, azok az utasítások, ahol, változókat, konstansokat stb. deklarálunk. A végrehajtandó utasításoknak már több csoportja létezik. Egy nagyobb csoport a vezérlési szerkezetet megvalósító utasítások. Ide soroljuk pl. amikor értéket adunk valamilyen változónak vagy konstansnak, az elágazásokat biztosító utasításokat tehát az if-et vagy több elágazás esetén a switch-et. Megemlíthető még az ugró utasítás (goto) is, de annak használatát már tudjuk helyettesíteni és néhány esetben veszélyes lehet. A következő csoport azok az utasítások, amikkel egy tevékenységet többször is végrehajthatunk, tehát a ciklus utasítások. Ezeken belül még többféle csoportosítás lehetséges. A legfontosabb megemlíteni azt, hogy a ciklus elől vagy hátul tesztelő. Az elől tesztelő esetben a ciklus először vizsgálja a feltételt majd fut le, míg a hátul tesztelő esetén minden esetben lefut legalább egyszer az utasításblokk majd utána nézi a ciklus a feltételt. A tipikus példa az elől, illetve hátul tesztelő ciklusra a while és a do-while. Ezen kívül még néhány ciklusnak meghatározhatjuk hány futás történjen, ilyen esetben viszont minden ciklusváltozót használunk a fejében.

Az, hogy a programunkat, hogy tagoljuk nyelvenként változik. Vannak nyelvek, amelyekben külön-külön fordítható önálló részeket írhatunk még más nyelvekben egységeként fordítjuk a kódot és csak annak a belső részeit mélyíthetjük, illetve létezik ezek kombinációja is. Az alprogramok az eljárásorientált nyelvekben jelennek meg, arra használjuk őket, ha a program egy részét többször szeretnénk használni. Az ilyen alprogramokban általánosan megírjuk minek kell történnie és ezt a főprogramban csak meghívjuk ahányszor csak szükséges. Az alprogramokban formális paramétereket használunk. Két típust különböztetünk meg: az eljárásokat és a függvényeket. Az eljárás valamilyen tevékenységet végrehajt és az eredményét tudjuk használni a meghívott helyen. A függvények minden valamilyen értéket adnak vissza, ezt visszatérési értéknek nevezzük és a függvény deklarálásakor meghatározzuk a típusát. Fontos megemlíteni, hogy az itt deklarált változók lokálisak, tehát csak itt használhatók a programban. Hívási láncnak vagy rekurzióknak nevezzük azt a jelenséget, ha egy programrészben egy másik programrész meghívunk. Nincs meghatározva hány rekurzív hívás van „összefűzve”. A programrészben meghívhatjuk önmagát vagy már egy korábban meghívott részt.

A paraméterátadás a programrészek közötti kommunikáció egy módja. Két résztvevője van: a hívó és a

hívott, a hívó lehet akármilyen programrész, a hívott viszont csak alprogram lehet. A paraméterátadást az információ iránya, tartalma szerint csoportosítjuk. A C-ben az érték szerinti átadást használjuk. Az érték szerinti átadáskor a formális karakter megkapja az adott értéket, ilyenkor értékmásolás történik. Ez csak akkor lehetséges, ha az alprogramban a formális paraméternek van címe. Az érték átadódik és ezzel az alprogram a saját területén dolgozik, tehát az információ áramlás csak egyirányú. Hátránya, hogy lassú lehet nagyobb adatcsoporthoz való másolásánál. A C++-ban már megjelent a referencia szerinti átadás is.

A blokk egy másik program belsejében elhelyezkedő programegység, amely tetszőleges utasításokat tartalmaz. A kezdetét és végét valamilyen speciális karakter jelöli. Rákerülhet automatikusan a vezérlés vagy ráléphetünk goto utasítással. A változóknak, konstansoknak, adatszerkezeteknek stb. van egy hatásköre, ami azt jelenti, hogy a program mely részein értelmezhető, végezhetők vele műveletek. Ennek beállítása két módon történhet: statikus és dinamikus. Statikus esetben a fordítóprogram dolgozik, végigmegy a kódon és egyes programrészekre megnézi lokális-e az adott név. Ezt addig csinálja amíg el nem érjük a legkülső részét a programnak. Ilyenkor kaphatunk hibát vagy deklarálódhat automatikusan a név. Az ilyen módú beállításnál az adott programrészben deklarált változók lokálisak. Ha nem ott deklaráltuk őket, de elérhetők akkor pedig globálisak. A dinamikus megoldásban úgynevezett hívási láncot használunk. Az deklarált változók hatásköre a deklarálásuk helyének programrésze és minden olyan programrész, ami ezzel hívási láncban van. Az ilyen hatáskörkezelést a futtató rendszer végzi.

Az input-output kezelés a legtöbb programozási nyelvnél változik, azonban a legtöbb nyelvre jellemző, hogy középpontjában az állomány áll. Ez áll egy fizikai és logikai részből és funkciója szerint csoportosítjuk. Az input állományból csak olvasni lehet és már létezik. Az output állomány nem létezik a létrehozása előtt és ebbe csak írni lehet. Az input-output állományok ennek keveréke, lehet olvasni-írni és létezik fel-dolgozás előtt, de a tartalmát megváltoztathatjuk. Az állományokból való munka több lépésből áll: a logikai részt deklarálni kell, majd megfeleltetni a fizikai állománnyal. Ezt követően tudjuk megnyitni, feldolgozni, majd bezárni. A C-ben az I/O műveletek alapból nem részei a nyelvnek, standard könyvtári függvényeket használunk hozzájuk.

A kivételkezelés arra szolgál, hogy a megszakításoknak a kezelését a program szintjén tudjuk végrehajtani. Egy megadott esemény után lép életbe, arra az eseményre reagálva. A bizonyos kivételek kezelése letiltható és engedélyezhető.

A programok alapvetően szekvenciálisak tehát sorrendben hajták végre az utasításokat. Az erőforráshasz-nálatnak megfelelően a programokat csoportosíthatjuk szálakra és folyamatokra. A folyamatok csak külön-külön míg a szálak közösen birtokolhatják az adott erőforrásokat. A szálak kommunikálnak egymással, időközönként találkoznak és adatcserét hajthatnak végre és egymással versenyben vannak az erőforrások-kert. Biztosítanunk kell, hogy ne használjanak egy időben adatokat pl. az egyik változtatja a másik felveszi értékül.

## 10.2. Programozás bevezetés

### [KERNIGHANRITCHIE]

A C-ben, mint sok más nyelvben a változókat aszerint csoportosítjuk, hogy milyen értéket tárolunk bennük. A számokat két csoportra oszthatjuk, az egészekre és a törtekre. Az egészeket tároló változó az int, ehhez tartozhatnak szimbólumok, amelyek a méretét változtatják. Ezek a short, long és az unsigned. A számokhoz tartoznak még a törteket tartalmazó változók, ezeknek két típusa van a float és a double. A float-ban 7, a double-ben pedig 14 tizedespontosságig tárolhatjuk a törteket. Emellett van a char típus, ami egy karakternyi értéket tárol. C-ben minden változót deklarálnunk kell a használata előtt. A változók bitmérete

viszont változhat. Ha globális konstansokat akarunk létrehozni, akkor arra azt definiálnunk kell. Ezt a program során nem változtathatjuk meg, akkor szoktuk használni, ha egy értékkel sokat számolunk, de nem változtatjuk az értékét.

A nyelv vezérlésátadó utasításai azt mondják meg mikor kerül egy utasításra sor a lefutás során. A C-ben a következő vezérlésátadó utasítások szerepelnek: If utasítást akkor használunk a programban, ha többféle lehetőséget szeretnénk megadni a kifejezés kimenetétől függően. A gép kiértékeli az if fejében lévő kiértékelést majd, ha igazat kapott végrehajtja az utasítást. Ha nem igaz értéket kapott alap esetben nem történik semmi, de írhatunk egy else ágat, amivel az ilyen esetekre adunk megoldást. Az if-else ágakat akármeddig fűzhetjük egymásba viszont ez nem ajánlott mert bonyolult nehezen érthető kódot kapunk. Ha mégis használunk egymásba ágyazott if-else kifejezéseket, akkor fontos megemlíteni, hogy az else a hozzá legközelebbi if-hez tartozik. A switch utasítással egymásba ágyazott if-else-eket valósíthatunk meg elegánsabb módon. A fejében ugyanúgy van egy kifejezés, ami kiért ékelődik majd egyezéseket keres a megadott case-ekkel. Ezek a case-ek egymástól különböző egész értékek lehetnek és hozzájuk utasításblokkok tartoznak, amelyeket break utasításokkal zárunk. A program sorban halad és egyezéseket keres a case-ekben a kifejezés értékével és ha talál lefuttatja hozzá tartozó utasítást. Ha nem talál a default utasítás fut le. A while utasításban a gép kiértékeli a fejben található kifejezést, majd, ha az értéke nem nulla, akkor végrehajtja az adott utasítást és ismételten kiértékel. Addig zajlik ez az értékelés és futtatás amíg a megadott kifejezésre nem kapunk nullát. A for utasítás három részből áll, amelyek mindegyike egy kifejezés. Ezek általában következőképpen néznek ki: Az első kifejezésben deklarálunk egy ciklusváltozót, amit a harmadikban növelünk, a második kifejezés pedig egy a ciklusváltozóhoz köthető relációs kifejezés. Ezek bármelyike elhagyható, de az utána következő pontosvesszőt ki kell raktunk, viszont, ha relációs kifejezést hagyjuk ki akkor a ciklus végtelen lesz. A do-while utasítás a vizsgálatot a ciklus végén az utasítások végrehajtása után végzi el, tehát legalább egyszer minden végrehajtódnak az utasításai. A gép először végrehajtja az utasításokat, majd kiértékeli a kifejezést. Ha a kifejezés értéke hamis lesz akkor ér véget a ciklus, pontosan úgy, mint a sima while ciklus esetében. A break utasítást arra használjuk, hogy egy ciklusból való kiléést ne a relációs utasítással vezérlejük. A break-el a vizsgálat előtt is ki lehet ugrani egy for, while vagy do-while ciklusból, vagy akár switch-ből. A break utasítás hatásra a vezérlés azonnal kilép a legbelőző zárt ciklusból. A continue utasítás szorosan kapcsolódik a break-hez, de kevesebbet használjuk. A continue utasítás hatására a ciklus következő részére ugrik a vezérlés, for esetén az újra inicializálásra. A goto utasítással címkékre ugorhatunk, olyan esetekben használjuk a legtöbb esetben, ha egymásba ágyazott ciklusokból akarunk kilépni. A használata nem ajánlott veszélyes lehet.

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

## 10.3. Programozás

[BMECPP]

A C++ a C nyelv egy továbbfejlesztése, az újítások közül sok a C-ben esetlegesen nem biztonságos elemeket cseréli megbízhatóbbra. A C++ nem objektumorientált újításai az alábbiak: Ha egy függvénynek nem adunk paramétereket akkor az a C-től eltérően, ahol ilyen esetben akármennyi paramétert megadhattunk, itt azt jelenti, hogy a paraméterlista üres. Ezen kívül kikerült még a nyelvből az alapértelmezett visszatérési érték, minden nem void típusú függvénynél meg kell adnunk. Kivétel erre a main () függvény amire a fordítóprogram a return hiányában return 0; -t fordít. Fontos újítás a bool típus megjelenése, a C-ben a logikai értékeket int vagy enum típusokkal tárolták, C++ -ban erre már ott van bool típus ami igaz, vagy hamis értéket vehet fel. Bekerült a több bájtos stringek (pl. UNICODE karakterek) tárolására alkalmas wchar

típus, amit C-ben csak bizonyos könyvtárak használata mellett tudtunk alkalmazni. A változódeklarációk már mindenhol szerepelhetnek, deklarálhatunk változókat közvetlenül a használatuk előtt. Lehetőség van a függvénynevek túlterhelésére is. C-ben a függvényeket a neve azonosítja egyértelműen ezért a hatáskörén belül nem lehet egy másik ugyanolyan nevű létrehozni. Erre már a C++-ban van lehetőség, mert itt a függvényeket nem csak a neük, hanem a paraméterlistájuk szerint csoportosítjuk. Ennek az az egyik előnye, hogy nem kell minden új már-már erőltetett függvényneveket kitalálni. A fordító a névfordítás technikával különbözteti meg őket. Lehetőség van arra is, hogy a függvényeknek paramétereinek megadjuk alapértelmezett értékeket, arra az esetre, ha nem kapnak értéket. A C-ben, ha paraméterként áadtunk egy változót akkor azt érték szerinti átadással tehetük. És az a paramétert a függvényen belül nem változtathattuk, csak akkor, ha pointereket használunk a függvényben, ami az érték szerinti paraméterátadás. A C++-ban lehetőségünk van a referencia szerinti paraméterátadásra, mivel bekerült a referencia típus is. Ilyen típust a referencia neve elő írt „&” jellel hozunk létre, a referencia típusának megfelelő változóval. Ha a referencia változik az eredeti változó is fog. Ezt a típust általában a cím szerinti paraméterátadásnál szoktuk használni, mert ha a változókra hivatkoznánk így a kódot áltáthatatlanná tenné.

Az objektumorientált programozás egyik alapelve az egységebe zárás, ami annyit jelent, hogy akármiből csinálhatunk egy adatstruktúrát, a hozzá tartozó tulajdonságokkal. Ezt az adatstruktúrát osztálynak nevezzük. Az osztályok példányait objektumoknak nevezzük, és fontos tulajdonságuk az adatrejtés, ami annyit jelent, hogy egyes elemek nem férnek hozzá a többi elemhez. Lehetőségünk van az öröklődésre tehát létrehozhatunk speciális osztályokat az osztályokon belül, amik öröklik az alap osztály tulajdonságait. pl. a dolgozó osztályon belül lehet egy eladó osztály. Amikor egy ilyen osztályt példányosítunk egy új objektumot hozunk létre. Ezek az objektumorientáltság alapelvei. Az osztályokon belül létrehozott változókat tagváltozóknak a függvényeket tagfüggvényeknek nevezzük. Ha egy változót deklarálunk az adott osztályból minden tagváltozónak történik memória foglalás. A tagfüggvényeket deklarálhatjuk a struktúrán belül és a törzsét a struktúrán kívül. A tagváltozókkal ellentétben tagfüggvények csak egy példában foglalnak helyet a memóriában. Az adatok elrejtésére a private kulcsszót használjuk a private: után felsorolt tagváltozók és függvények csak az osztályban lesznek láthatók és tagfüggvényekkel férhetünk hozzájuk. Az objektumok fontos része a konstruktur, amivel példányosításkor inicializálni tudjuk az objektumunkat. Egy speciális függvény, ami minden egyes példányosításkor meghívódik. Egyszerre többet is írhatunk egy objektumhoz, ha nem írnunk egyet sem akkor a fordítóprogram alapértelmezetten ír egyet, ami semmit nem csinál. A konstruktur párja a destruktur, ami akkor hívódik meg, ha egy objektum megszűnik. Ez az objektum által használt erőforrásokat szabadítja fel. C-ben a dinamikus memória kezelés a malloc és a free függvények használatával oldottuk meg. A malloc függvény viszont nem teszt különbséget a típusok között és az objektumok memória foglalására képtelen. C++-ban ezeket oldja a meg a new és a delete függvény. Osztályok esetén a new függvény után, ha üres zárójelet használunk a paraméter nélküli konstruktur esetében, ha a konstruktornak vannak paraméterei azokat meg kell adnunk a new függvényben. A konstruktur egy speciális típusa a másolókonstruktur. Ugyanúgy működik, mint más konstruktorkor, de az objektum inicializálásakor egy másik már létező objektumot másol le. A C++-ban lehetőség van arra, hogy egy osztály feljegosítson más osztályokat vagy függvényeket, hogy a védett részeihez hozzáférjenek. Ha a függvény elő írjuk a friend kulcsszót akkor az adott függvénynek olyan jogai vannak az osztály részeinek hozzáféréséhez, mintha az adott osztály tagfüggvénye lenne. Készíthetünk friend osztályokat is, amik hozzá tudnak férni az adott osztály védett elemeihez. A friend viszony nem sérti a védett elemek elvét, hiszen csak korlátozott módon férhetünk hozzájuk és ritkán alkalmazzuk. A statikus változók olyan változók az osztályban, amelyek az adott osztályhoz tartoznak és nem változnak példányosított objektumonként, akkor is használhatók, ha még az osztály még nem példányosított egy objektumot sem. Használhatunk statikus tagfüggvényeket is, előnye, hogy objektum nélkül tudjuk használni, viszont csak az osztály statikus részeit éri el. A C++-ban megadhatunk osztályon belüli típusdefiníciókat, amiket beágyazott definícióknak

nevezünk. Ezeknek az elérhetősége az osztály elérhetőségétől függ.

A C++ -ban a matematikai műveleteket az operátorok végzik. Ezek a beépített típusokra érvényesek, ahhoz, hogy a saját típusainkon használjuk őket úgynevezett túlterhelést kell alkalmazni. Ezeknek az operátoroknak van egy alap végrehajtódási sorrendje, amit zárójelekkel manipulálhatunk. Ezen kívül van egy argumentumok és egy visszatérési értékük. A C++-ban megjelent néhány új operátor is pl.: a hatáskör operátor és a pointer-tag operátor. Itt az operátorokat felfoghatjuk úgy, egy speciális függvény és sajátot is írhatunk, de ennek nem az a célja, hogy már a jól bevált operátorokat újraírjuk hanem hogy alkalmazni tudjuk az operátorokat a saját típusainkon. Mivel az operátorok is függvények ezért úgy terheljük túl őket, mint egy függvényt.

A C-nyelvben bekérésre és kiíratásra magas szintű állományleírókat használtunk (stdin, stdout) a C++-ban már adatfolyamok (cout, cin) állnak a rendelkezésünkre, amik a bitenkénti balra és jobbratolás operátorait használják. A típusleíró formátumstringekkel sem kell bajlódnunk a >> operátor automatikusan felismeri az adott típust. A cin alapvetően a „.” -ig olvas egész számokat, ha ezt meg akarjuk változtatni az ignore függvényt kell használunk, ahol beállíthatjuk, hogy mi legyen az olvasási limit. Mivel ezek a kiírások és beolvasások rendszerhívásokat alkalmaznak, amik költsége magas, ezért buffer-eket rendelnek hozzájuk. Ha több cout kiíratás van a programban akkor előbb összegyűjt a szükséges adatokat és egy rendszerhívást használ rájuk. A buffer-ek kiürítésre a flush () függvényt használjuk vagy küldhetünk hibaüzeneteket a cerr objektumból. És ha hibát kapunk a beolvasás ugyan lefut, de nem csinál semmit. A C++-ban már a string-ek is megjelennek, ezek olvasására is használhatjuk a >> operátort, de a szóközt tartalmazó string-et akarunk beolvasni azt a getline () függvényvel lehetjük. Az adatfolyamoknak vannak tagfüggvényei, amelyekkel olvashatjuk, írhatjuk őket, de ez nem az egyetlen mód a megváltoztatásukra. Végezhetjük ezeket manipulátorokkal is, amik speciális objektumok. Ilyen manipulátor pl. az endl (sortörés) vagy az ignorews (szóköz figyelmen kívül hagyása). Szöveg formázására is használhatunk tagfüggvényeket és manipulátorokat. Az állománykezelés is adatfolyamok által működik (ifstream, ofstream). Ezeket osztályok reprezentálnak. A megnyitásukat és bezárásukat végezhetik a konstruktorok és destruktorkor vagy használhatunk rájuk open és close függvények.

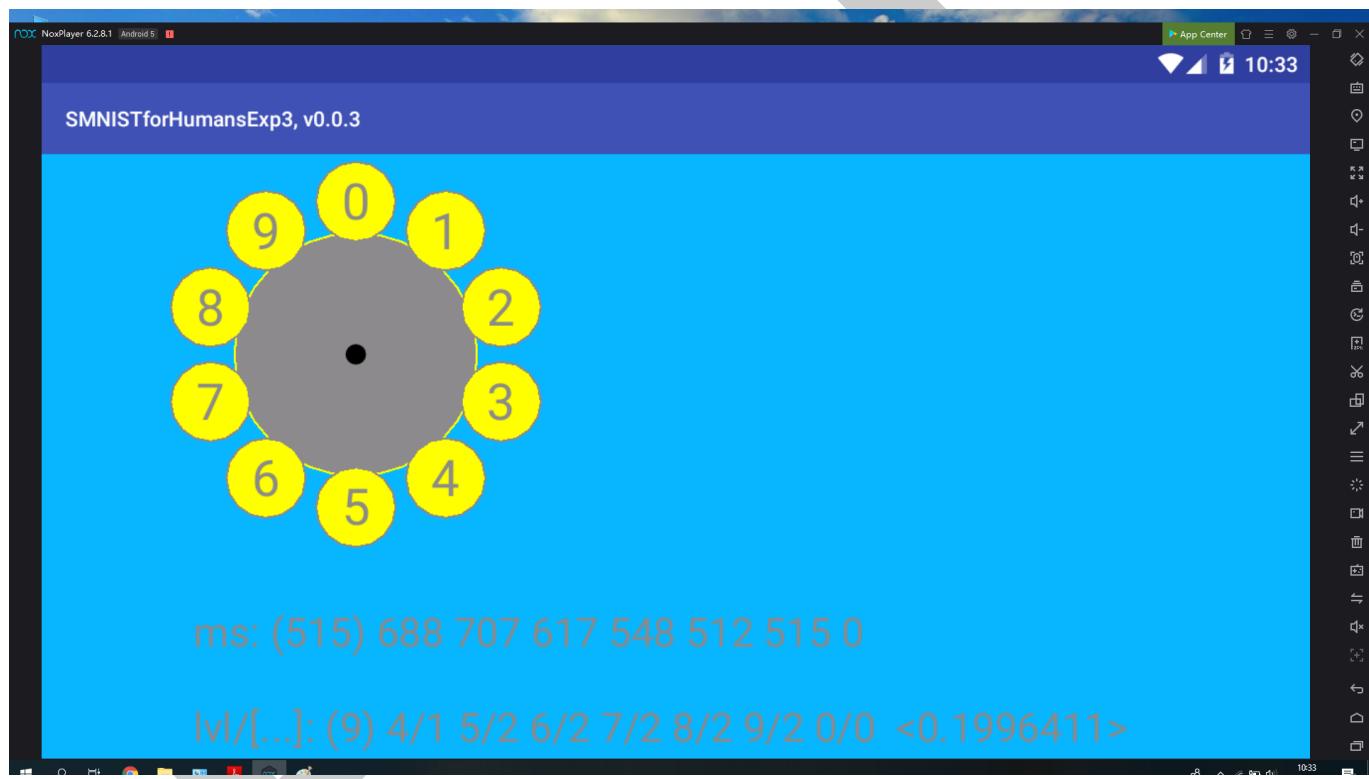
A C++ -ban a hibák kezelésére a kivételkezelést alkalmazzuk. A kivétel érzékelésekor a vezérlés rögtön a kivételkezelőre ugrik. A kivételkezelést a try-catch blokkal oldjuk meg. A try {} -ei között lévő blokk a védett blokk, amiben, ha nem találunk hibát minden utasítása lefut. Ha találunk kivételt, akkor azt throw függvényel eldobjuk és az a catch függvény elkapja, amelynek megegyeznek a paraméterei. A try-catch blokkokat egymásba is ágyazhatjuk így különböző szinteken oldhatjuk meg a kivételkezelést. Ha a throw-nak nem adunk meg paraméter a kivétel újra dobható lesz. A könyv 190. oldalán egy olyan programban alkalmazzuk a try catch párost, ami egy szám reciprokját adja meg, és a bekért szám nem lehet 0-mert, akkor 0-val való osztás történne. A könyv 197. oldalán lévő példában az úgynevezett verem visszacsévélése folyamatot láthatjuk, ami annyit tesz, hogy a kivétel dobása után amíg az elkapás meg nem történik egy hívási láncban haladunk felfelé és a lokális függvények változóit felszabadítjuk.

# 11. fejezet

## Összegzés:

### 11.1. Passzolt feladatok:

2 db az SMNIST for Humans kutatástámogatás értelmében: Mély MNIST és Minecraft-MALMÖ



11.1. ábra. Lvl. 9

### 11.2. Tutoriáltjaim:

Borvíz Róbert: Hangyszimulációk [https://github.com/BorvizRobi/prog\\_1\\_textbook](https://github.com/BorvizRobi/prog_1_textbook)

Borvíz Róbert: A Mandelbrot halmaz:[https://github.com/BorvizRobi/prog\\_1\\_textbook](https://github.com/BorvizRobi/prog_1_textbook)

## 11.3. Tutorok:

Borvíz Róbert: Decimálisból unárisba átváltó Turing-gép [https://github.com/BorvizRobi/prog\\_1\\_textbook](https://github.com/BorvizRobi/prog_1_textbook)

Borvíz Róbrt: C Exor Törő [https://github.com/BorvizRobi/prog\\_1\\_textbook](https://github.com/BorvizRobi/prog_1_textbook)

Racs Tamás: Mozgató szemantika:<https://gitlab.com/cant0r/bhax>

## 11.4. Licensz

A nem saját programok forrásai a hozzájuk tartozó feladatban meg vannak említve.

A saját programokra (amiknél jelenik meg forrás) a következő licensz érvényes:

```
/*
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <https://www.gnu.org/licenses/>.
*/
```

**III. rész**

**Második felvonás**

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

## 12. fejezet

# Helló, Arroway!

### 12.1. OO szemlélet

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

[https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1\\_5.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf)

(16 22 fólia)

Ugynéz írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

**Megoldás forrása:**

[GitLab Java](#)

[GitLab C++](#)

**Tanulságok, tapasztalatok, magyarázat...**

A polártranszformációs algoritmus fontos jellemzője, hogy egy számítási lépéssel két normális eloszlású számot állít elő. Ez azért fontos, mert minden páratlanadik meghívásnál nem kell újraszámolnunk, csak visszaadni az előző lépés másik előre kiszámolt számát.

**Java megoldás:**

A programot Java-nak megfelelően egy PolarGenerator osztályba írjuk. Ennek tagjai a nincsTarolt és tarolt változók, amik az tartalmazzák hogy van-e tárolt szám, és ha van mi az. Az osztály tartalmaz továbbá egy konstruktort, amiben igazra állítjuk, hogy kezdésnek nincsen tárolt szám.

```
public class PolarGenerator {

 boolean nincsTarolt = true;
 double tarolt;

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

Maga a polártranszformációs algoritmus matematikai része a kovetkezo() metódusban zajlik. Ha nincs nincs tárolt szám, akkor lezajlik a szám generálás és a második érték bekerül a tarolt változóba, nincs Tarolt változó hamisra vált, az első értéket pedig kapjuk visszatérési értékként. Ha van tárolt szám akkor a nincs Tarolt-at negáljuk és a metódus visszaadja a tárolt számot, és nem generálunk új számot. A main-ben példányosítunk egy PolarGenerator objektumot, majd egy for ciklusban használjuk a kovetkezo() metódust a számok generálására.

```
public double kovetkezo() {
 if(nincsTarolt) {
 double u1,u2,v1,v2,w;
 do{
 u1 = Math.random();
 u2 = Math.random();
 v1 = 2 * u1 - 1;
 v2 = 2 * u2 -1;
 w = v1 * v1 + v2 * v2;
 }while(w>1);
 double r = Math.sqrt((-2 * Math.log(w)) / w);
 tarolt = r * v2;
 nincsTarolt = !nincsTarolt;
 return r * v1;
 }
 else{
 nincsTarolt = !nincsTarolt;
 return tarolt;
 }
}
public static void main(String[] args){
 PolarGenerator g = new PolarGenerator();
 for(int i =0;i<10;++i){
 System.out.println(g.kovetkezo());
 }
}
```

```
user@ubuntu:~/Asztal/Prog2_source/arroway/oo$ java PolarGenerator
0.3420621181206084
-1.3547707471885693
-0.7718170667385131
-0.1828616159790594
0.48796649235433603
0.2791974377464044
0.5957153241030483
-1.4123413089796657
0.7792452271356513
-0.5777930377181135
```

### C++ megoldás:

A C++ verzióban a PolarGen osztályt egy header fájlba írjuk. Az osztály public részében tartalmaz egy konstruktort ami beállítja, hogy nincs tárolt változó kezdésnek, illetve meghívja a srand() függvényt amire a polártranszformációs algoritmus során lesz szükség, random szám generáláshoz. Tartalmaz egy destruktort illetve, deklarálja a kovetkezo() függvényt. A számtároláshoz szükséges változókat az osztály private

részébe írjuk, mert nincs szükség rá, hogy közvetlenül el legyenek érve, elég csak a kovetkezo() függvényen keresztül.

```
#ifndef POLARGEN_H
#define POLARGEN_H

#include <cstdlib>
#include <cmath>
#include <ctime>

class PolarGen
{
public:
 PolarGen ()
 {
 nincsTarolt=true;
 std::srand(std::time(NULL));
 }
 ~PolarGen()
 {
 }
 double kovetkezo();

private:
 bool nincsTarolt;
 double tarolt;
};

#endif
```

A .cpp fájlban kifejtjük bővebben a kovezkezo() függvényt. Használnunk kell a :: operátort mert, ez egy PolarGen osztályon belüli függvény. Maga a kovetkezo() függvény ugyanúgy működik, mint a Java verzióban, csak C++-os függvéyekkel. A main-ben szintén csak példányosítunk és meghívjuk a kovezkezo() függvényt egy for ciklusban.

```
#include "polargen.h"
#include <iostream>

double PolarGen::kovetkezo()
{
 if(nincsTarolt) {
 double u1,u2,v1,v2,w;
 do{
 u1 = std::rand() / (RAND_MAX + 1.0);
 u2 = std::rand() / (RAND_MAX + 1.0);
 v1 = 2*u1 -1;
 v2 = 2*u2 -1;
 w = v1 * v1 + v2 *v2;
```

```
 }
 while(w>1);

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

 tarolt = r*v2;
 nincsTarolt = !nincsTarolt;

 return r * v1;
 }
else{
 nincsTarolt = !nincsTarolt;
 return tarolt;
}
}

int main(int argc,char **argv)
{
 PolarGen pg;

 for(int i=0;i<10;++i)
 std::cout<<pg.kovetkezo ()<<std::endl;

 return 0;
}
```

```
user@ubuntu:~/Asztal/Prog2_source/arroway/oo/cplus$./polar
-0.638607
-1.75706
-1.55561
1.18982
-0.130017
2.0166
-0.00793054
0.290321
-0.207981
1.33346
```

## 12.2. "Gagyi"

Megoldás forrása:[GitLab](#)

Program, ami lefagy:

```
public class Gagyi2
{
 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);
}
}
```

```
user@ubuntu:~/Asztal/Prog2_source/arroway/gagyi$ java Gagyi2
-129
-129
```

Program, ami nem fagy le:

```
public class Gagyi3
{
 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);
 }
}
```

```
user@ubuntu:~/Asztal/Prog2_source/arroway/gagyi$ java Gagyi3
-128
-128
```

Ha egy Integer változónak értéket adunk, akkor a public static Integer valueOf(int i) metódus szerint kap értéket. Ami annyit tesz, hogy ha -128 és 127 közötti értéket adunk, akkor nem példányosít új objektumot, hanem vannak az Integer cache-ben nekik előre elkészített objektumok. A második esetben azért teljesül a feltétel mert a két érték ugyanazt az Integer cache-ből kiszedett előre legyártott objektumot használja. A első programban pedig azért nem, mert a a -129 nincs benne az cache-ben előre legyártott objektumok skálájában, ezért két új objektumot kapunk a két változónak. Ezek a memóriában máshol helyezkednek el, így az összehasonlításuknál nem kapunk egyezést.

**valueOf**  
public static Integer valueOf(int i)  
Returns an Integer instance representing the specified int value. If a new Integer instance is not required, this method should generally be used in preference to the constructor Integer(int), as this method is likely to yield significantly better space and time performance by caching frequently requested values. This method will always cache values in the range -128 to 127, inclusive, and may cache other values outside of this range.  
Parameters:  
i - an int value.  
Returns:  
an Integer instance representing i.  
Since:  
1.5

12.1. ábra. JDK forrás

## 12.3. Yoda

Megoldás forrása: [GitLab](#)

Tanulságok, tapasztalatok, magyarázat...

A Yoda feltétel a programozásban azt jelenti, hogy a feltétes utasítás kifejezéseit megcseréljük, ezáltal fordítva "beszélünk", mint Yoda a Star Wars-ból. A kifejezés konstans értéke kerül a feltétel bal oldalára.

Az ilyen típusú utasítással elkerülhetünk például bizonyos NullPointerException hibákat.

Az alábbi program a szokásos if utasítás sorrendet használva NullPointerException hibát ad. Ez azért van mert, az equals() metódus azon paramétere amihez hasonlítunk nem lehet null érték.

```
public class Yoda{
 public static void main(String[] args) {
 String myString = null;
 if(myString.equals("foobar")) {}
 }
}
```

```
user@ubuntu:~/Asztal/Prog2_source/arroway/yoda$ java Yoda
Exception in thread "main" java.lang.NullPointerException
at Yoda.main(Yoda.java:4)
```

Yoda feltételt használva viszont nem kapunk hibát, mert az equals() metódus hasonlítandó paramétere viszont lehet null érték.

```
public class Yoda{
 public static void main(String[] args) {
 String myString = null;
 if("foobar".equals(myString)) {}
 }
}
```

```
user@ubuntu:~/Asztal/Prog2_source/arroway/yoda$ javac Yoda.java
user@ubuntu:~/Asztal/Prog2_source/arroway/yoda$ java Yoda
Nincs hibaüzenet
```

**equals**

```
public boolean equals(Object obj)
Indicates whether some other object is "equal to" this one.
The equals method implements an equivalence relation on non-null object references:
• It is reflexive: for any non-null reference value x, x.equals(x) should return true.
• It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
• It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
• It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
For any non-null reference value x, x.equals(null) should return false.
The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true).
Note that it is generally necessary to override the hashCode method whenever this method is overridden, so as to maintain the general contract for the hashCode method, which states that equal objects must have equal hash codes.
Parameters:
obj - the reference object with which to compare.
Returns:
true if this object is the same as the obj argument; false otherwise.
See Also:
hashCode(), HashMap
```

12.2. ábra. JDK forrás:

## 12.4. Kódolás from scratch

Megoldás forrása: [Javát tanítok könyv forrásai](#)

Tanulások, tapasztalatok, magyarázat...

A Bailey-Borwein-Plouffe (BBP) algoritmus a Pi hexadecimális értékeinek kiszámolására alkalmazható, anélkül, hogy ismernénk az előző számjegyeket.

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

A programban változókat hozunk létre a tört elemekkel való számolásra. Ezeknek az értékét a d16Sj() metódus adja. Ezt követően kiszámoljuk az adott Pi értéket és a StrictMath.floor() metódust használjuk kerekítésre.

```
public class PiBBP {
 String d16PiHexaJegyek;
 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);
```

12.3. ábra. A formula (forrás: [https://en.wikipedia.org/wiki/Bailey%E2%80%93Borwein%E2%80%93Plouffe\\_formula](https://en.wikipedia.org/wiki/Bailey%E2%80%93Borwein%E2%80%93Plouffe_formula))

Készítünk egy buffert és a hexajegyeket tartalmazó tömböt. Hozzáfűzzük az új jegyeket a meglévő Pi értékhez az append() metódussal és a bufferból stringet csinálunk a toString() metódussal.

```
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();
```

Metódusok elkészítése a tört értékek kiszámítására és bináris hatványozásra.

```
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);
}
```

```
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;
}
```

Visszaadjuk a kiszámolt hexajegyeket majd a main-ben példányosítunk egy objektumot.

```
public String toString() {
```

```

 return d16PiHexaJegyek;
}
```

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

{

```
user@ubuntu:~/Asztal/Prog2_source/arroway/bbp$ javac PiBBP.java
user@ubuntu:~/Asztal/Prog2_source/arroway/bbp$ java PiBBP
6C65E5308user@ubuntu:~/Asztal/Prog2_source/arroway/bbp$ █
```

DRAFT

## 13. fejezet

# Helló, Berners-Lee!

### 13.1. Python: Bevezetés a mobilprogramozásba

A Python egy platformfüggetlen programozási nyelv, magas szintű objektumorientált programozási nyelv. Egyik mobilprogramozáshoz használt változata a PythonS60 Symbian mobil op. rendszerre. A Python a C-től eltérően interpretes nyelv nem szükséges fordítani futtatás előtt. A nyelv előnye, hogy tartalmaz magas szintű adattípusokat, nincs szükség zárójelekre a kódcsoporthoz tabulátorral van megoldva, illetve a változókat sem kell definiálnunk. Az értelmező a sorokat tokenekre bontja és úgy értelmezi.

A Python minden adat objektum, a változók típusait nem kell nekünk megadni, a futás során értékének megfelelő típust kap. A változótípusokból a legtöbb a C-ben is megtalálható, ezektől eltérő változók a szótár, ennesek és a listák. Ezek egyszerre több, akár eltérő típusú értéket tartalmaznak és szekvenciáknak nevezzük őket. Indexelésük 0-tól kezdődik, de ha negatívan indexelünk a szekvencia végétől kezdődik. A len és del függvényekkel lekérhetjük a hosszukat, és törölhetünk elemeket. A listák rendezettek és dinamikusak, míg a szótárak rendezetlen kulcsokkal azonosított elemeket tartalmaznak. Egy változóhoz értékadáskor hozzárendelhetünk objektumot, típust és függvényt. Az elágazások és ciklusok hasonlóak, mint C-ben, a for ciklushoz használhatunk range(x) és xrange(x) függvényeket, ami egész értékeket tartalmaz 0-tól x-ig. Az elseif Python-os változata pedig az elif. Függvényeket a def kulcsszóval hozhatunk létre. A paraméterek érték szerint adódnak, át kivéve pl., ha listákat szótárakat adunk meg. Lehetőség van az alapértelmezett paraméterek megadására is, illetve megadhatunk paramétereket függvényhíváskor is.

Címkeket a label kulcsszóval lehetünk a szövegbe, itt fontos megemlíteni a goto és comefrom függvényeket. Ha a program futáskor elér a goto függvényhez az adott label-hez ugrik. A comefrom ettől ellentétesen, ha elér a program az adott label-hez a comefrom-ot tartalmazó kódrészhez ugrik. Lehetőségünk van osztály-definiálásra is, ezeknek a példányai objektumok, és tulajdonságai lehetnek függvények és objektumok. Ha egy osztályban valamilyen tulajdonságot megváltoztatunk, akkor az megváltozik az osztály példányainban is, ha csak az nem volt bennük már előtte megváltoztatva. Az attribútum függvényeket, vagy metódusokat globális függvényként lehet definiálni, az első paraméterük mindig a self lesz, ami arra az objektumpéldányra utal, ahol a függvényt meghívjuk. Az `__init__` metódus egy speciális metódus, ami konstruktorként működik. Első paramétere self, az objektum, amit létre akarunk hozni, a többi paraméter pedig a tulajdonságok, amiket hozzá akarunk rendelni az objektumhoz.

A Python tartalmaz mobilfejlesztésre használt modulokat. Ilyenek például az appuifw, amellyel kialakíthatjuk és kezelhetjük a felhasználói felületet. A messaging modul az üzenetkezelésért, a sysinfo modul a telefonnal kapcsolatos információk lekérdezésért felelősek. A kameráért és a hangért felelős modulok a camera és audio modul.

A hibakezelést try-except blokkban oldjuk meg a try blokkba kerül a kód, amiben a hibát keressük, az except blokkban pedig az a kód, ami lefut, ha hibát találunk. A keresendő hibát az except mellé írjuk [] zárójelekbe. Kapcsolhatunk egy else ágat is, az except után vagy esetleg helyette egy finally blokk, amit például fájlok bezárására lehet használni hiba esetén

## 13.2. Java: Java 2 útikalauz 5

A Java nyelv teljesen objektumorientált, egy program kizártlag osztályokból és objektumokból épül fel. A programunkat futtatás előtt fordítani kell, ilyenkor bájtkódokra fordul a program, amit a Java Virtuális Gép interpreteként értelmez és futtat. A nyelv UNICODE karaktereket használ.

Az egyszerű változótípusok hasonlóak, mint a C-ben/C++-ban. A C-hez képest újdonság viszont a String típus, amit a karakterláncok kezelésére hoztak létre. Az azonosítók csak betűvel kezdődhetnek és nem tartalmazhatják a nyelv kulcsszavait. Az egyszerű típusúakhoz valódi értékadással, míg az összetett típusok esetén referencia által rendelünk hozzájuk értéket.

Az egyszerű és összetett típusok inicializálásakor literálokat használunk. Objektumoknál ez a null, ami bármely objektumreferencia helyett használható. A logikai érték lehet igaz vagy hamis, az egész számokat a kezdőkarakterük határozza meg: lehetnek oktális, hexadecimális és decimálisak

Ha egy értékét többször fel szeretnénk használni egy programban és nem akarjuk változtatni az értékét, érdemes konstansként, nem pedig változóként deklarálni. Ezt a final static kulcsszavakkal tehetjük meg. A lebegőpontos számok decimálisak tizedesponttal elválasztva. Karaktereket” szövegeket” jelek közé írunk. A szövegek létrehozásakor String objektum kerül létrehozásra.

A tömbök a C/C++-tól eltérően nem mutatók, hanem valódi típus. Nem primitív típusok, objektumhivatkozást tartalmaznak. Többdimenziós tömbök alapértelmezetten nincsenek a nyelvben, de meg lehet oldani a létrehozásokat pl.: a tömb tartalmazhat tömböket. Bizonyos értékeket felsorolási típusokban is tárolhatunk pl.: hét napjai. Ezek értékét akár tömbökben is elhelyezhetjük.

Megjegyzéseket háromféleképpen írhatunk a kódba. Ezek közül kiemelendő a dokumentációs megjegyzés, amit a javadoc alkalmazással HTML dokumentációvá tehetünk. A megjegyzések minden fajtáját figyelmen kívül hagyja a fordító.

Osztályokat a Class kulcsszóval hozhatunk létre, ezekhez adattagok és metódusok tartoznak, ezeket más nyelvekben változóknak és függvényeknek nevezzük. A Java-ban az objektumokat az new operátorral példányosíthatunk. pl. Osztály objektum1 = new Osztály (). Ilyenkor memóriát foglalunk, ezt a new operátor tárolja. Az osztály változó pedig egy referenciát tartalmaz az objektumról, ami ténylegesen az objektumra mutat nem pedig a memóriában lévő címre. Ellentétben például a C-vel, ahol mutatókat alkalmazunk, amik adott memóriacímre utalnak. Az osztály típusú változók egyben referencia típusú változók is. Ha használjuk a final kulcsszót az osztályváltozó előtt akkor minidig csak ugyanarra az objektumra fog hivatkozni, viszont a tulajdonságait változtathatjuk. A null referenciaérték egyetlen objektumra sem mutat, ez a referencia típusú változók alapértelmezett értéke. Ha egy objektumot törlni szeretnénk azt más nyelvekben általában manuálisan kell megtennünk. Java-ban, ha a rendszer látja, hogy nincs az objektumra hivatkozva, akkor automatikusan törlődik. Ezt a szemétfgyűjtő mechanizmus végzi. Egy hivatkozás akkor szabadul fel, ha változója új értéket kap vagy törlődik.

A kivételek kezelésére a try-catch blokkot használjuk, mint a C-ben. Maga a program a try blokkban van, ha futás közben hibát talál a fordító a try blokkban a catch blokkra ugrik a vezérlés. A catch blokknak dobott hiba is egy objektum típusa a hiba fajtája, adattagjai pedig a hiba részletei.

A Java-ban a műveletek ugyanolyan sorrendben értékelődnek ki (balról jobbra), mint a C-ben és az operátorok is nagyjából megegyeznek. A logikai műveleteknek két fajtája van: a mohó és a lusta kiértékelésű. A lusta változatban az operandusok nem minden kerülnek kiértékelésre pl.: és művelet első tagja hamis.

Típuskonverzióra a nyelvben háromféle lehetőség van. Ha többféle primitív típussal végzünk műveletet akkor automatikus a konverzió automatikus, az eredmény minden olyan típusú lesz amelyik nagyobb tárhelyű. Explicit konverzióra akkor van szükség, ha alaptípusok között a kisebb értelmezési tartományra szeretnénk váltani, illetve objektumok referenciatípusa esetén, ha a statikus és dinamikus típusa nem egyezik meg. Azaz, ha a deklarálásnál megadott típus különbözik annak az objektumnak a típusától amire hivatkozik. A harmadik típus a szövegkonverzió, amikor egy nem String típust String típusra konvertálunk, vagy konkatenációt használunk.

Ahhoz, hogy elérjünk egy adott struktúra egy részelemét minden esetben a „.” -ot kell használni, nincs megkülönböztetett jelölés az osztálytagok elérésére mint C++-ban a „::” operátor. Az utasításoknak két fajtája van: a kifejezés (értékkedás, metódushívások stb.), és a deklarációs-utasítások. Elágazásoknak két fajtáját különböztetjük meg: az egyszerűt (if) és összetettet (switch). Ezek ugyanúgy működnek, mint C-ben/C++-ban. Ciklusok tekintetében a jól ismert fajták (while, do-while, for) itt is megtalálhatók. A bejáró ciklus egy különleges for ciklus, ami adatszerkezetek bejárására használható. A címkek, break, continue utasítások ugyanazok, mint C-ben viszont a Java elhagyta a nem túl biztonságos goto utasítást.

DRAFT

## 14. fejezet

# Helló, Liskov!

### 14.1. Liskov helyettesítés sértése

Megoldás forrása: [GitLab](#)

Tanulások, tapasztalok, magyarázat...

A Liskov elv szerint az objektumorientált programozási nyelvekben, ha T-nek egy altípusa S, akkor amelyik programrészben a T-t használjuk probléma nélkül behelyettesíthetjük az S-t is. A probléma a tulajdonságoknál léphet fel, amit a következő példaprogram jól szemléltet.

Két osztályunk van: a Madar és a Program, a Madar osztályt tekintjük T-nek. Ez az osztály tartalmaz egy repul() függvényt ezt úgy is tekinthetjük, hogy a madarak tudnak repülni. Ezután következnek az Madar alosztályai a Sas és Pingvin osztályok. A Pingvin osztály viszont megséríti a Liskov elvet, mert madár, de repülni nem tud, a programban viszont hiba nélkül tudnánk rá alkalmazni a repul() függvényt. Az ilyen problémákra a jobb tervezés lehet megoldás például ha lenne külön nem repülő és repülő madár osztály.

C++ program:

```
// ez a T az LSP-ben
class Madar {
public:
 virtual void repul() {};
};

class Program {
public:
 void fgv (Madar &madar) {
 madar.repul();
 }
};

class Sas : public Madar
{};

class Pingvin : public Madar
{};
```

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

 Sas sas;
 program.fgv (sas);

 Pingvin pingvin;
 program.fgv (pingvin);

}
```

### Java program:

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

 static class Program{
 public void fgv(Madar madar){
 madar.repul();
 }
 }

 static class Sas extends Madar{}
 static class Pingvin extends Madar{}

 public static void main(String[] args) {

 Program program = new Program();
 Madar madar = new Madar();
 program.fgv(madar);

 Sas sas = new Sas();
 program.fgv(sas);

 Pingvin pingvin = new Pingvin();
 program.fgv(pingvin);
 }
}
```

```
user@ubuntu:~/Asztal/Prog2_source/liskov/sertes$ g++ liskovsert.cpp -o sert
user@ubuntu:~/Asztal/Prog2_source/liskov/sertes$./sert
user@ubuntu:~/Asztal/Prog2_source/liskov/sertes$ █
```

14.1. ábra. A programok hiba nélkül futnak, de a Liskov elvet sértik

```
user@ubuntu:~/Asztal/Prog2_source/liskov/sertes$ javac Madar.java
user@ubuntu:~/Asztal/Prog2_source/liskov/sertes$ java Madar
user@ubuntu:~/Asztal/Prog2_source/liskov/sertes$
```

## 14.2. Szülő-gyerek

Megoldás forrása:[GitLab](#)

Tanulságok, tapasztalatok, magyarázat...

Mind Java-ban és C++-ban az ősön keresztül csak az ős üzenetei közvítíthetők. A példaprogramokban látszik, hogy a gyermekosztályokban definiálunk egy függvényt, amit a szülőosztályból példányosított objektumban akarunk használni, de erre nincs lehetőség, fordításban hibát kapunk.

**Java példa:**

```
class Szulo{

 static class Gyerek extends Szulo{
 void msg(){
 System.out.println("Hello");
 }
 }

 public static void main(String[] args) {
 Szulo szulo = new Szulo();
 szulo.msg();
 }
}
```

```
user@ubuntu:~/Asztal/Prog2_source/liskov/szulo-gyerek$ javac Szulo.java
Szulo.java:16: error: cannot find symbol
 szulo.msg();
 ^
symbol: method msg()
location: variable szulo of type Szulo
1 error
```

**C++ példa:**

```
#include <iostream>

class Szulo{
public:
 int a;

 Szulo(int a){
 a=a;
 }
}
```

```
};

class Gyerek:public Szulo{
public:
 int a;

 Gyerek(int a):Szulo(a){
 a=a;
 }

 int fgv(){
 return a;
 }
};

int main(){
 Szulo* szulo = new Szulo(5);
 szulo->fgv();

 return 0;
}
```

```
user@ubuntu:~/Asztal/Prog2_source/liskov/szulo-gyerek$ g++ szgy.cpp -o szgy
szgy.cpp: In function 'int main()':
szgy.cpp:27:12: error: 'class Szulo' has no member named 'fgv'
 szulo->fgv();
 ^~~
```

### 14.3. Ciklomatikus komplexitás

A ciklomatikus komplexitás Thomas J. McCabe nevéhez fűződik. A kódunk komplexitására ad egy mértéket, annak alapján hogy hány lehetséges elágazás van benne. Kézileg, gráffal kell ábrázolunk a programot és a következő képletet kell használnunk:

$$M = E - N + 2P \text{ ahol:}$$

- E: gráf éleinek száma
- N: gráf csúcsainak száma
- P: összefüggő komponensek száma

Ennek az értéknek a kiszámítására nem csak manuálisan van lehetőség több program áll rendelkezésünkre.

Én a GNU komplexitás mérő programját használtam amivel, C programok komplexitását lehet mérni.

Eredmények az LZW binfa C verzióján:

```

user@ubuntu:~/Asztal/comp$ complexity --histogram --score --thresh=0 'binfa.c'
Complexity Scores
Score | ln-ct | nc-lns| file-name(line): proc-name
 1 6 6 binfa.c(112): szabadit
 1 8 7 binfa.c(17): uj_elem
 2 13 13 binfa.c(94): kiir
 6 51 41 binfa.c(34): main

Complexity Histogram
Score-Range Lin-Ct
 0-9 67 ****
Scored procedure ct: 4
Non-comment line ct: 67
Average line score: 4
25%-ile score: 2 (75% in higher score procs)
50%-ile score: 6 (half in higher score procs)
75%-ile score: 0 (25% in higher score procs)
Highest score: 6 (main() in binfa.c)
user@ubuntu:~/Asztal/comp$

```

## 14.4. Anti OO

A mérést Ubuntu 64 bites virtuális gépen végeztem, 8 GB RAM-al és Intel i5 1.6 Ghz-es processzorral.

Forrás: [GitLab](#)

|        | <b>C</b>   | <b>C++</b> | <b>C#</b> | <b>Java</b> |
|--------|------------|------------|-----------|-------------|
| $10^6$ | 1.713795   | 1.71992    | 1.577164  | 1.539       |
| $10^7$ | 20.055328  | 20.1204    | 18.313944 | 18.043      |
| $10^8$ | 234.730421 | 235.586    | 212.29012 | 207.146     |

14.1. táblázat. Mérési eredmények táblázatba gyűjtve:

Azt hinnénk a C gyorsabb lesz, mint a Java a fordításuk közötti különbség miatt (C rögtön gépi kódra, a Java először bytekódra fordítódik le és fordítja tovább a Java Virtual Machine). Ebben az esetben viszont a Java kódból "kivetettük az objektumorientáltságot, két metóduson kívül, minden kód a main-be került és így a Java bizonyult gyorsabbnak.

**Használt kódok a futási eredményekkel:**

**C**

```

#include <stdio.h>
#include <math.h>
#include <time.h>
/*
 * pi_bbp_bench.c
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu

```

```
*
* A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
* a PiBBPBench osztályt, amit pedig átírtuk C nyelvre.
*
*/

/*
 * 16^n mod k
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
long
n16modk (int n, int k)
{
 long r = 1;

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

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

 t = t / 2;

 if (t < 1)
 break;

 r = (r * r) % k;
 }

 return r;
}

/* {16^d Sj}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
double
d16Sj (int d, int j)
{
 double d16Sj = 0.0;
```

```
int k;

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

/*
 for(k=d+1; k<=2*d; ++k)
 d16Sj += pow(16.0, d-k) / (double) (8*k + j);
*/

return d16Sj - floor (d16Sj);
}

/*
 * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
main ()
{
 double d16Pi = 0.0;

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

 int jegy;
 int d;

 clock_t delta = clock ();

 for (d = 100000000; d < 100000001; ++d)
 {
 d16Pi = 0.0;

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

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

 d16Pi = d16Pi - floor (d16Pi);

 jegy = (int) floor (16.0 * d16Pi);
 }
}
```

```
printf ("%d\n", jegy);
delta = clock () - delta;
printf ("%f\n", (double) delta / CLOCKS_PER_SEC);
}
```

```
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$./pi_bbp
6
1.713795
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ gcc pi_bbp_bench.c -o pi_bbp -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main ()
 ^~~~
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$./pi_bbp
7
20.055328
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ gcc pi_bbp_bench.c -o pi_bbp -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main ()
 ^~~~
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$./pi_bbp
12
234.730421
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$
```

C++

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

long n16modk(int n,int k) {
 long r = 1;

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

 for(;;) {

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

 t = t / 2;

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

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

return r;
}

double d16Sj(int d,int j){

 double d16Sj = 0.0;
 int k;

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

 return d16Sj - floor(d16Sj);
}

main() {
 double d16Pi = 0.0;

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

 int jegy;
 int d;

 clock_t delta = clock ();

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

 d16Pi = 0.0;

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

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

 d16Pi = d16Pi - floor (d16Pi);

 jegy = (int) floor (16.0 * d16Pi);

 }

 std::cout<<jegy<<"\n";
}
```

```
 delta = clock() -delta;
 std::cout<<(double)delta / CLOCKS_PER_SEC<<"\n";
}
```

```
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ g++ Pi.cpp -o Pi
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$./Pi
6
1.71992
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ g++ Pi.cpp -o Pi
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$./Pi
7
20.1204
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ g++ Pi.cpp -o Pi
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$./Pi
12
235.586
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ █
```

## C#

```
/*
 * FileName: PiBBPBench.cs
 * Author: Bátfai Norbert, nbatfai@inf.unideb.hu
 * DIGIT 2005, Javat tanítok
 */
/// <summary>
/// A PiBBPBench C# átírata.
/// </summary>
/// <remark>
/// A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
/// a PiBBPBench osztályt, amit pedig átírtuk C# nyelvre.
///
/// (A PiBBP osztály a BBP (Bailey–Borwein–Plouffe) algoritmust a Pi hexa
/// jegyeinek számolását végző osztály. A könnyebb olvahatóság
/// kedvéért a változó és metódus neveket megpróbáltuk az algoritmust
/// bemutató [BBP ALGORITMUS] David H. Bailey: The BBP Algorithm for Pi.
/// cikk jelöléseihez.)
/// </remark>
public class PiBBPBench {
 /// <remark>
 /// BBP algoritmus a Pi-hez, a [BBP ALGORITMUS] David H. Bailey: The
 /// BBP Algorithm for Pi. alapján a {16^d Sj} részlet kiszámítása.
 /// </remark>
 /// <param>
 /// d a d+1. hexa jegytől számoljuk a hexa jegyeket
 /// </param>
 /// <param>
 /// j Sj indexe
 /// </param>
```

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

 double d16Sj = 0.0d;

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

 /*
 *
 for(int k=d+1; k<=2*d; ++k)
 d16Sj += System.Math.pow(16.0d, d-k) / (double)(8*k + j);
 */

 return d16Sj - System.Math.Floor(d16Sj);
}
/// <summary>
/// Bináris hatványozás mod k, a $16^n \text{ mod } k$ kiszámítása.
/// </summary>
/// <param>
/// n kitevő
/// </param>
/// <param>
/// k modulus
/// </param>
public static long n16modk(int n, int k) {

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

 long r = 1;

 while(true) {

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

 t = t/2;

 if(t < 1)
 break;

 r = (r*r) % k;
 }

 return r;
}
/// <remark>
```

```
/// A [BBP ALGORITHMUS] David H. Bailey: The
/// BBP Algorithm for Pi. alapján a
/// {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
/// kiszámítása, a {} a törtrészt jelöli. A Pi hexa kifejtésében a
/// d+1. hexa jegytől
/// </remark>
public static void Main(System.String[]args) {

 double d16Pi = 0.0d;

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

 int jegy = 0;

 System.DateTime kezd = System.DateTime.Now;

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

 d16Pi = 0.0d;

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

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

 d16Pi = d16Pi - System.Math.Floor(d16Pi);

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

 }

 System.Console.WriteLine(jegy);
 System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
 System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
}

}
```

```
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ mcs -out:Pi.exe PiBBP.cs
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ mono Pi.exe
6
1,577164
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ mcs -out:Pi.exe PiBBP.cs
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ mono Pi.exe
7
18,313944
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ mcs -out:Pi.exe PiBBP.cs
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ mono Pi.exe
12
212,29012
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$
```

### Java:

```
/*
 * PiBBPBench.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
 * ezt az osztályt.
 *
 * (A PiBBP osztály a BBP (Bailey-Borwein-Plouffe) algoritmust a Pi hexa
 * jegyeinek számolását végző osztály. A könnyebb olvahatóság
 * kedvéért a változó és metódus neveket megpróbáltuk az algoritmust
 * bemutató [BBP ALGORITMUS] David H. Bailey: The BBP Algorithm for Pi.
 * cikk jelöléseihez.)
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class PiBBPBench {
 /**
 * BBP algoritmus a Pi-hez, a [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján a {16^d Sj} részlet kiszámítása.
 *
 * @param d a d+1. hexa jegytől számoljuk a hexa jegyeket
 * @param j Sj indexe
 */
 public static double d16Sj(int d, int j) {

 double d16Sj = 0.0d;

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

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

return d16Sj - Math.floor(d16Sj);
}
/***
 * Bináris hatványozás mod k, a 16^n mod k kiszámítása.
 *
 * @param n kitevő
 * @param k modulus
 */
public static long n16modk(int n, int k) {

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

 long r = 1;

 while(true) {

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

 t = t/2;

 if(t < 1)
 break;

 r = (r*r) % k;
 }

 return r;
}
/***
 * A [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján a
 * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
 * kiszámítása, a {} a törtrészt jelöli. A Pi hexa kifejtésében a
 * d+1. hexa jegytől
 */
public static void main(String args[]) {

 double d16Pi = 0.0d;
```

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

int jegy = 0;

long delta = System.currentTimeMillis();

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

 d16Pi = 0.0d;

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

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

 d16Pi = d16Pi - Math.floor(d16Pi);

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

}

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

```
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ javac PiBBPBench.java
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ java PiBBPBench
6
1.539
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ javac PiBBPBench.java
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ java PiBBPBench
7
18.043
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ javac PiBBPBench.java
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ java PiBBPBench
12
207.146
user@ubuntu:~/Asztal/Prog2_source/liskov/antioo$ █
```

# 15. fejezet

## Helló, Mandelbrot!

### 15.1. Reverse engineering UML osztálydiagram

**Megoldás forrása:** -

Tanulságok, tapasztalatok, magyarázat..

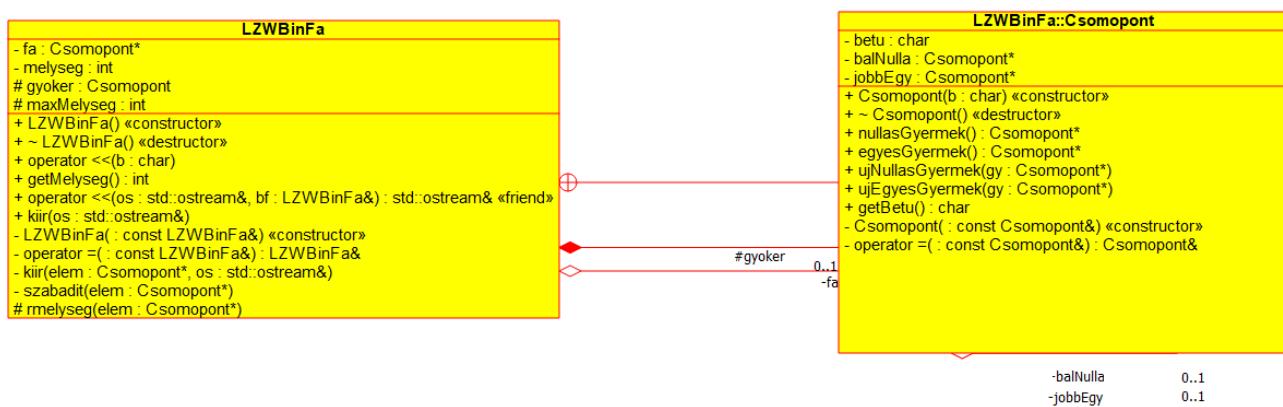
**UML-ről általánosan:**

Az UML egy általános modellező nyelv. Kiválóan használható az objektorientált programozási nyelveken írt programok ábrázolására.

**A feladatról:**

A feladat az volt, hogy a LZW Binfa kódjáról készítsünk egy UML osztálydiagrammot. Az UML diagramok készítésére sok program áll rendelkezésünkre, ehhez a feladathoz Umbrello-t használtam, a későbbiekben pedig Visual Paradigm-et.

**Az LZW Binfa UML osztálydiagramja:**



**Magyarázat:**

A diagramokon a két téglalap a program két osztályát jelzi. A belső téglalalpok közül a felső az osztály attribútumait az alsó pedig a függvényeit ábrázolja. Az ezek előtt látható +,- és # jelek az adott függvények, és változók hozzáférését jelzik:

- +: public
- -: private
- #: protected

A kapcsolatok:

A piros rombusz végű nyíl a két osztály közötti kompozíciót jelöli. Ez azt jelenti hogy az adott objektum nem létezik a szülőobjektum nélkül. Ebben az esetben nincs csomópont ha, nincs gyökér.

A fehér rombusz aggregációs kapcsolatot jelent. Ilyenkor az objektum a szülőobjektum nélkül is létezik.

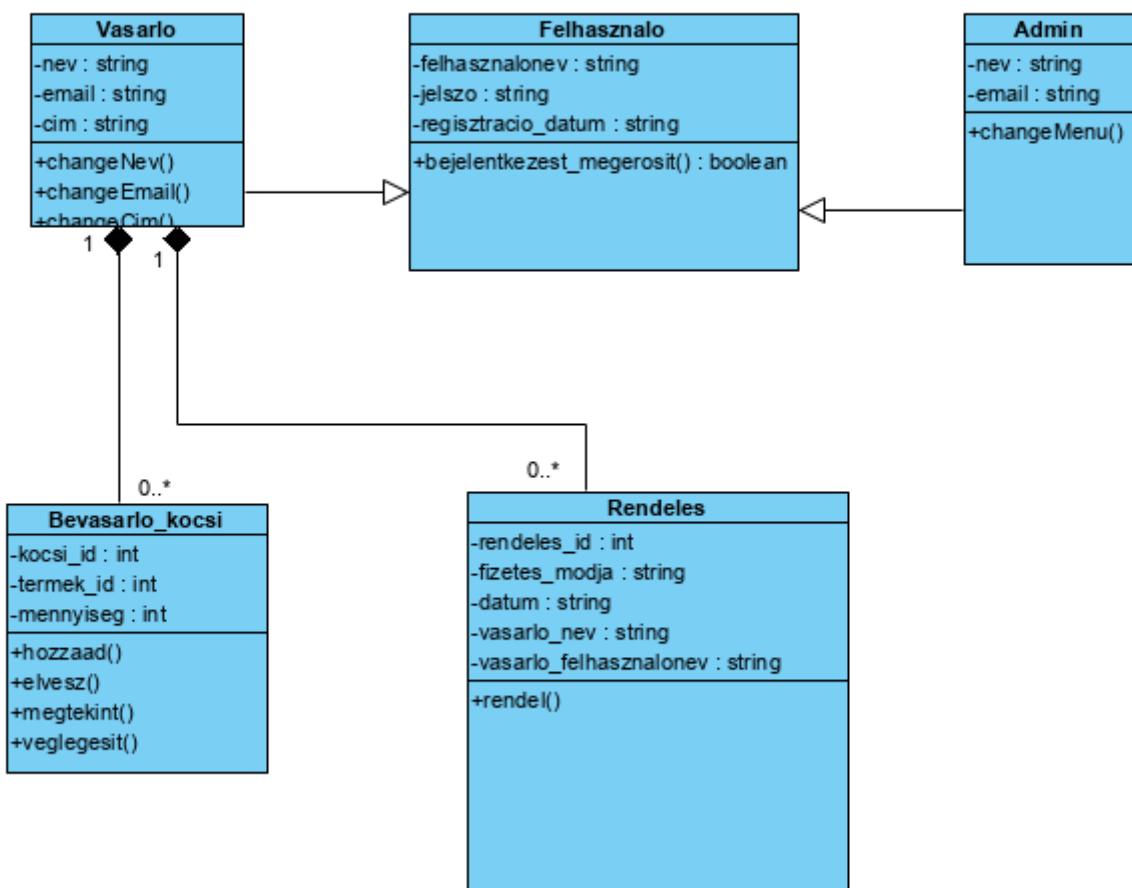
## 15.2. Forward engineering UML osztálydiagram

Megoldás forrása: -

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy készítsünk egy UML osztálydiagramot, majd abból generálunk forrást.

Egy étterem a online rendeléshez használt programjáról csináltam UML diagramot. A felhasználó osztálynak kettő alosztálya van jól látható hogy ez a nyilakkal meg van jelölve. A bevásárló kocsi és rendelés osztályok kompozícióban vannak a vásárló osztállyal. Egy kocsihoz és rendeléshez csak egy vásárló tartozhat. A diagramot Visual Paradigm-ben készítettem, és a kódot is azzal generáltam le.



A kód generálásnál az osztályok külön header és cpp fájlokba kerültek. A header fájlokat meghagytam a .cpp fájlokat viszont beírtam egy fájlba az átláthatóság miatt.

#### Header fájlok:

```
#ifndef ADMIN_H #ifndef VASARLO_H #ifndef FELHASZNALO_H
#define ADMIN_H #define VASARLO_H #define FELHASZNALO_H
class Admin : Felhasznalo { class Vasarlo : Felhasznalo { class Felhasznalo {
private: private: private:
 string nev; string nev; string felhasznalonev;
 string email; string email; string jelszo;
public: string cim; string regisztracio_datum;
void changeMenu(); public: boolean bejelentkezest_megerosit();
}; void changeNev(); };
#endif void changeEmail(); };
 void changeCim();
 };
 #endif

#ifndef BEVASARLO_KOCSI_H #ifndef RENDELES_H
#define BEVASARLO_KOCSI_H #define RENDELES_H
class Bevasarrollo_kocsi { class Rendeles {
private: private:
 int kocsi_id; int rendeles_id;
 int termek_id; string fizetes_modja;
 int mennyiseg; string datum;
public: string vasarlo_nev;
void hozzaad(); string vasarlo_felhasznalonev;
void elvesz(); public: void rendel();
void megtekint(); };
void veglegesit(); #endif
};
#endif
```

A "program":



```
#include "Admin.h"

void Admin::changeMenu() {
}

#include "Bevasarlo_kocsi.h"

void Bevasarlo_kocsi::hozzaad() {
}

void Bevasarlo_kocsi::elvesz() {
}

void Bevasarlo_kocsi::megtekint() {
}

void Bevasarlo_kocsi::veglegesit() {
}

#include "Felhasznalo.h"

bool Felhasznalo::bejelentkezest_megerosít() {
}

#include "Vasarlo.h"

void Vasarlo::changeNev() {
}

void Vasarlo::changeEmail() {
}

void Vasarlo::changeCim() {
}

#include "Rendeles.h"

void Rendeles::rendel() {
}
```

### 15.3. BPMN

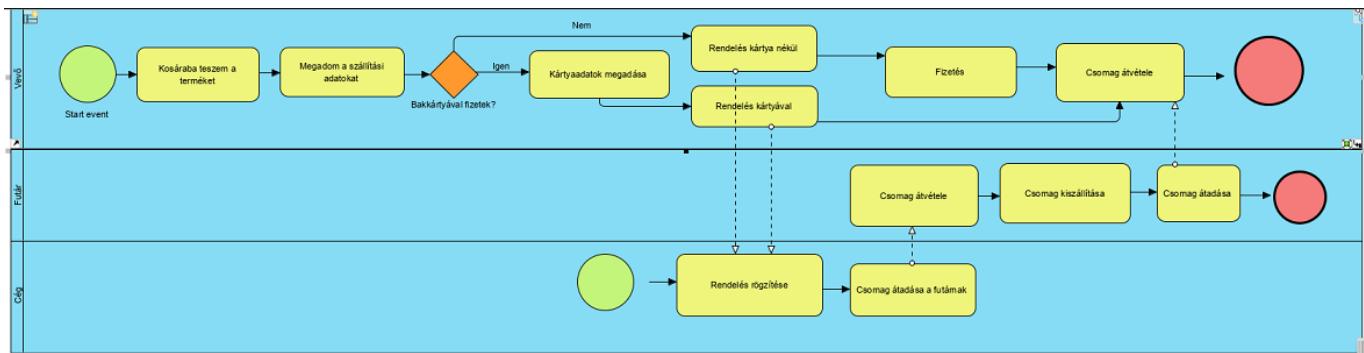
Megoldás forrása: -

Tanulások, tapasztalatok, magyarázat...

A BPMN(Buisness Process Model and Notation)-nel valamilyen üzeleti folyamatot tudunk megjeleníteni grafikus ábrázolással.

A feladat megoldásakor egy hétköznapi folyamatot, egy onlnie vásárlást próbáltam meg ábrázolni.

**BPMN modell:**



### Magyarázat:

A zöld körök a kezdő, a pirosak a végeredményt jelzik, míg a négyzetek az elvégzendő feladatokat. A nyílak az események a végrehajtásának a sorrendjét jelölik, ha ezek szaggatottak, akkor több fél a résztvevő. A rombusszal azt jelezzük, hogy kétféle lehetőség van a továbbhaladásra.

A konkért példában először szükségünk van egy termékre, ez a kezdő folyamat. Majd kiválasztjuk és kosárba tesszük a terméket, megadjuk a szállítási adatokat. Ezután lehetőség van előre fizetni bankkártyával, majd megrendeljük a terméket. Ez a cégnél is beindítja a folyamatotokat rögzítik a rendelést majd átadják a csomagot a futárnak. Ez beindítja a futár folyamatait átveszi, kiszállítja, majd átadja a csomagot és véget ér a folyamat. A vevő, ha szükséges fizet, átveszi a csomagját, majd az ő folyamai és véget érnek.

# 16. fejezet

## Helló, Chomsky!

### 16.1. Encoding

Megoldás forrás:[GitLab](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy a Mandelbort Halmaz nagyító programját fordítsuk és futtassuk úgy, hogy minden a kódban és a fájlnévben meghagyjuk az ékezes karaktereket.

Ha, ezt a szokásos módon próbáljuk meg, akkor kapjuk a hibaüzeneteket, hogy az UTF-8 számára ismeretlen karaktereket tartalmaz a kódunk.

```
user@ubuntu: ~/Asztal/Prog2_source/chomsky/encoding
Fájl Szerkesztés Nézet Keresés Terminál Súgó
// vizsgáljuk egy adott pont iteraciót:
^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
UTF-8
 // vizsgáljuk egy adott pont iteraciót:
^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
UTF-8
 // vizsgáljuk egy adott pont iteraciót:
^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
UTF-8
 // Az egérmutató poziciója
^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
UTF-8
 // Az egérmutató poziciója
^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
UTF-8
 // Az egérmutató poziciója
^
100 errors
user@ubuntu:~/Asztal/Prog2_source/chomsky/encoding$
```

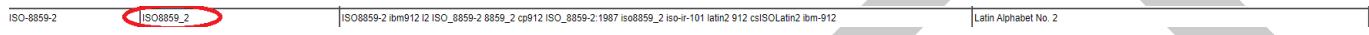
A javac dokumentációban látható, hogy a `-encoding` kapcsolóval lehet jelezni a fordítónak, hogy nem a szokásos kódolással van dolga.

**-encoding encoding**

Set the source file encoding name, such as EUC-JP and UTF-8. If `-encoding` is not specified, the platform default converter is used.

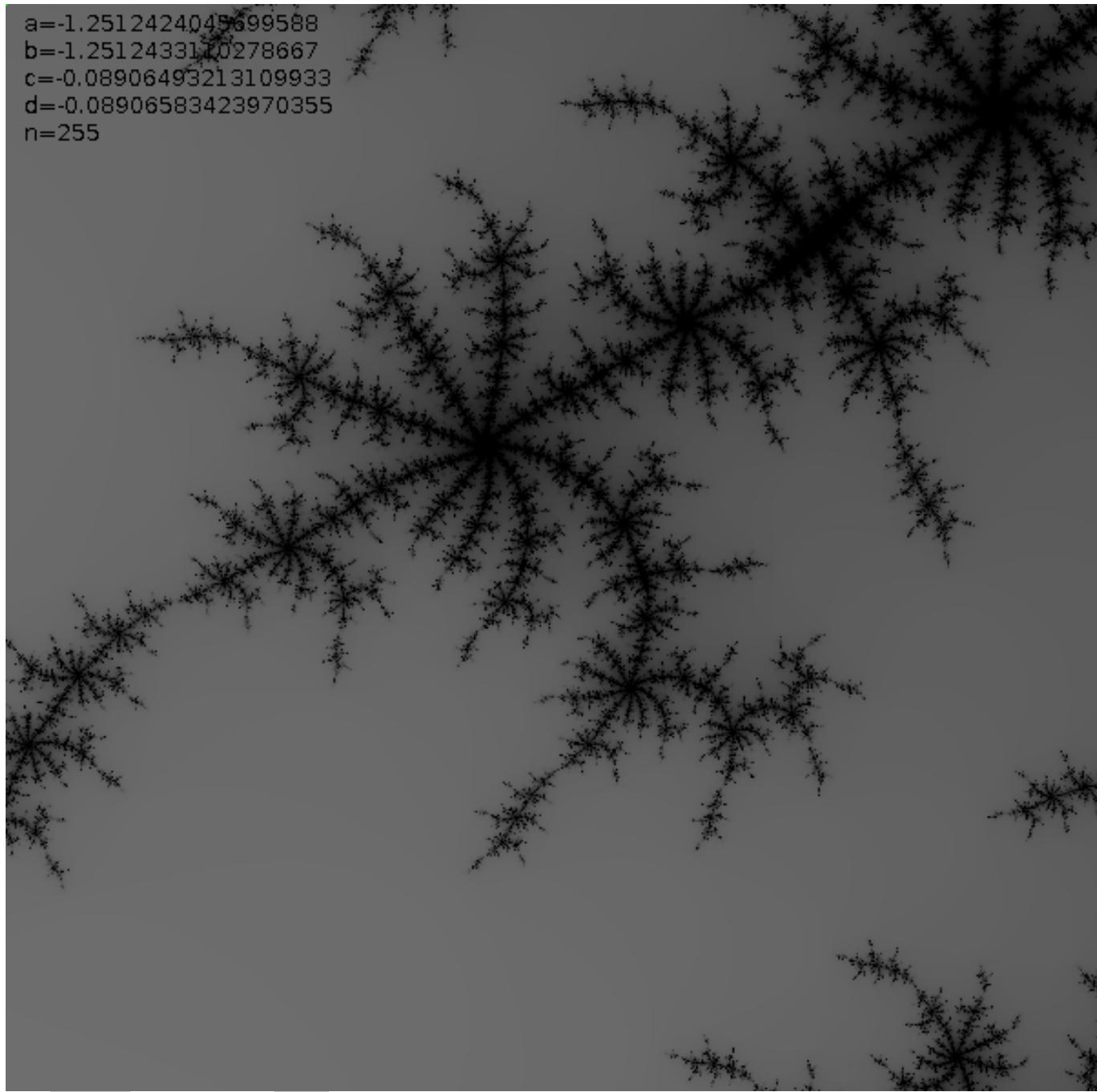
Ezt követően a wikipédián megkerestem, mi a magyar ékezetes karakterek kódolása. Ezek a karakterek az ISO/IEC 8859-2 (Latin-2) karakterkészletében találhatóak meg.

Az Oracle oldalán megnéztem, hogy a java.lang API-ban, ez hogyan van jelölve, majd az encoding kapcsolónak így adtam meg és a program hiba nélkül fordult, futott.



// Az egérmutató pozíciója  
user@ubuntu:~/Asztal/Prog2\_source/chomsky/encoding\$ javac -encoding ISO8859\_2 MandelbrotHalmazNagyító.java  
user@ubuntu:~/Asztal/Prog2\_source/chomsky/encoding\$

DRAFT



16.1. ábra. A program futásokor készített screenshot

## 16.2. Saját Leet Cipher

Megoldás forrása:[GitLab](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat az, volt hogy lexer használata nélkül írunk egy programot ami átkonvertálja a kapott szöveget leet szöveggé.

A leet osztály tartalmaz egy input stringet, ami tartalmazza a bekért szöveget, illetve egy output string vektort, ami a konvertált szöveget tartalmazza.

A konvertálás a cipher függvényben zajlik. Itt az output vektoron referencia szerinti átadást alkalmazunk hogy a későbbiekben visszakaphassuk a megváltozott értékét. Transfrom() és toupper() metódusokkal a bekért stringet nagybetűvé konvertáljuk, hogy ne kelljen különbséget tenni a kis- és nagybetű között.

Ezután végimegyünk az input string minden karakterén és a karakternek mefelelő leet karaktert töltünk az output vektorba. A konstruktorban futtattjuk a cipher függvényt, itt látható, hogy az output vector referenciaival van megadva.

A main()-ben deklaráljuk az input stringet és az output vektort, illetve egy quit stringet és egy quit\_bool-t. Ezekkel a bekérés megszakítását végezzük el. A program addig fut amíg a quit\_bool igaz. minden konvertálás után megkérdezzük (bekérjük a választ a quit stringre), akar-e még a felhasználó konvertálni. Ha igen a quit\_bool igazra állítjuk ha nem kilépünk. Maga a konvertálás rész, pedig a konstruktor meghívásával hajtódiik végre.

A kód:

```
#include <vector>
#include <iostream>
#include <string.h>
#include <stdio.h>
#include<bits/stdc++.h>

using namespace std;

class Leet{

public:
 string input;
 vector<string> output;
 vector<string> cipher(string input, vector<string>* output) {

 transform(input.begin(), input.end(), input.begin(), ::toupper);

 for(int i=0;i<input.length();i++)
 {
 switch (input.at(i)) {
 case 'A' : output->push_back("4");
 break;
 case 'B' : output->push_back("13");
 break;
 case 'C' : output->push_back("(");
 break;
 case 'D' : output->push_back("1 ")");
 break;
 case 'E' : output->push_back("3");
 break;
 case 'F' : output->push_back("1=");
 break;
 case 'G' : output->push_back("6");
 break;
 }
 }
 }
};
```

```
 break;
 case 'H' : output->push_back("|-|");
 break;
 case 'I' : output->push_back("| ");
 break;
 case 'J' : output->push_back(".]");
 break;
 case 'K' : output->push_back("|<");
 break;
 case 'L' : output->push_back("1");
 break;
 case 'M' : output->push_back("|Y|");
 break;
 case 'N' : output->push_back("N");
 break;
 case 'O' : output->push_back("0");
 break;
 case 'P' : output->push_back("P");
 break;
 case 'Q' : output->push_back("Q");
 break;
 case 'R' : output->push_back("|2");
 break;
 case 'S' : output->push_back("5");
 break;
 case 'T' : output->push_back("7");
 break;
 case 'U' : output->push_back("|_|");
 break;
 case 'V' : output->push_back("V");
 break;
 case 'W' : output->push_back("W");
 break;
 case 'X' : output->push_back("}{");
 break;
 case 'Y' : output->push_back("//");
 break;
 case 'Z' : output->push_back("2");
 break;
 case ' ' : output->push_back(" ");
 break;
 case '0' : output->push_back("D");
 break;
 case '1' : output->push_back("I");
 break;
 case '2' : output->push_back("Z");
 break;
 case '3' : output->push_back("E");
 break;
 case '4' : output->push_back("A");
```

```
 break;
 case '5' : output->push_back("S");
 break;
 case '6' : output->push_back("b");
 break;
 case '7' : output->push_back("T");
 break;
 case '8' : output->push_back("B");
 break;
 case '9' : output->push_back("g");
 break;
}

}

return *output;
}

Leet(string input,vector<string> &output) {
 cipher(input,&output);
}

};

int main(){
 vector<string> output;
 string input;
 string quit;
 bool quit_b =true;

 while(quit_b==true){
 quit_b=false;
 cout<<"Add meg a szöveget!\n";
 cin>>input;

 cout<<"A szöveg amit megadtál: "<<input<<"\n";

 Leet test(input,output);
 cout<<"A leet-es szöveg:\n";
 for(int i=0;i<output.size();i++) {
 cout<<output.at(i);
 }
 cout<<"\n";

 cout<<"Folytatni akarod?(i/n)?\n";
 cin>>quit;
 if(quit=="i") {
 output.clear();
 }
 }
}
```

```
 quit_b=true;
 }
 else if(quit=="n") {
 break;
 }

}
}
```

A program futása:

```
user@ubuntu:~/Asztal/Prog2_source/chomsky$ g++ leet.cpp -o leet
user@ubuntu:~/Asztal/Prog2_source/chomsky$./leet
Add meg a szöveget!
Test
A szöveg amit megadtál: Test
A leet-es szöveg:
7357
Folytatni akarod?(i/n)?
i
Add meg a szöveget!
leet
A szöveg amit megadtál: leet
A leet-es szöveg:
1337
Folytatni akarod?(i/n)?
n
user@ubuntu:~/Asztal/Prog2_source/chomsky$
```

### 16.3. Full Screen

Megoldás forrása: [GitLab](#)

Tanulások, tapasztalatok, magyarázat...

A feladat az volt, hogy készítsünk egy Java programot, ami Full Screen-ben fut.

A megoldásra a Bátfai Norbert Tanár Úr által készített Labirintus játékot használtam. A játék működését nem részletezem, mert a feladatban csak a teljesképernyős módon van a hangsúly.

A **LabirintusJáték.java** fájl kódrészleteinek magyarázata:

Az osztály a java Frame osztályát egészíti ki, ami lényegében adja, az ablakos rendszert. A GraphicsDevice osztályból példányostítunk egy objektumot. Ez az osztály azokat a grafikus eszközöket adja meg, amelyek használhatóak az adott grafikus környezethez. Ezeket a getLocalGraphicsEnvironment() metódussal kérjük le, az alapértelmezett képernyőt pedig a getDefaultScreenDevice() metódussal.

```
public class LabirintusJáték extends java.awt.Frame
 implements Runnable {
.
```

```
•
•
java.awt.GraphicsDevice graphicsDevice;
•
•
java.awt.GraphicsEnvironment graphicsEnvironment = java.awt. ←
 GraphicsEnvironment.getLocalGraphicsEnvironment();
•
•
graphicsDevice = graphicsEnvironment.getDefaultScreenDevice();
•
•
teljesKépernyősMód(graphicsDevice);
```

A Full Screen beállítását a teljesKépernyősMód() metódusban végezzük el. A szélesség és magasság változókat kinullázzuk, mert az ablaknak "nincsen kerete". A setResizable() metódusai kikapcsoljuk az átméretehetőséget. Lekérdezzük támogatott-e a Full Screen mód, ha igen, ezt az isFullScreenSupported() metódussal teszzük. Ha támogatott akkor átadjuk a setFullScreenWindow() metódusnak a képernyő tulajdonságait. Ezt a this referenciaival adjuk át. A támogatott felbontásokat egy tömbbe kérjük le. A képernyő tulajdonságait mint a szélesség, magasság, színmélység, bitmélység és frissítési frekvencia változókban tárljuk el, és ezeket a tulajdonságokat majd megjelenítjük a képernyő sarkában. A támogatott felbonásokat egy tömbben tároljuk el. Megnézzük az adott felbontás támogatott-e (itt 1920x1080), ha igen ezt állítjuk be a felbontásnak, ha nem kiírjuk, hogy a felbontás nem lesz jó a játékhoz.

```
public void teljesKépernyősMód(java.awt.GraphicsDevice graphicsDevice) ←
{

 int szélesség = 0;
 int magasság = 0;
 setUndecorated(true);
 setIgnoreRepaint(true);
 setResizable(false);
 boolean fullScreenTamogatott = graphicsDevice.isFullScreenSupported ←
 ();
 if(fullScreenTamogatott) {
 graphicsDevice.setFullScreenWindow(this);
 java.awt.DisplayMode displayMode
 = graphicsDevice.getDisplayMode();
 szélesség = displayMode.getWidth();
 magasság = displayMode.getHeight();
 int színMélység = displayMode.getBitDepth();
 int frissítésiFrekvencia = displayMode.getRefreshRate();
 System.out.println(szélesség
 + "x" + magasság
 + ", " + színMélység
 + ", " + frissítésiFrekvencia);
 java.awt.DisplayMode[] displayModes
 = graphicsDevice.getDisplayModes();
 boolean dm1024x768 = false;
 for(int i=0; i<displayModes.length; ++i) {
```

```
if(displayModes[i].getWidth() == 1920
 & displayModes[i].getHeight() == 1080
 && displayModes[i].getBitDepth() == színMélység
 && displayModes[i].getRefreshRate()
 == frissítésiFrekvencia) {
 graphicsDevice.setDisplayMode(displayModes[i]);
 dm1024x768 = true;
 break;
}

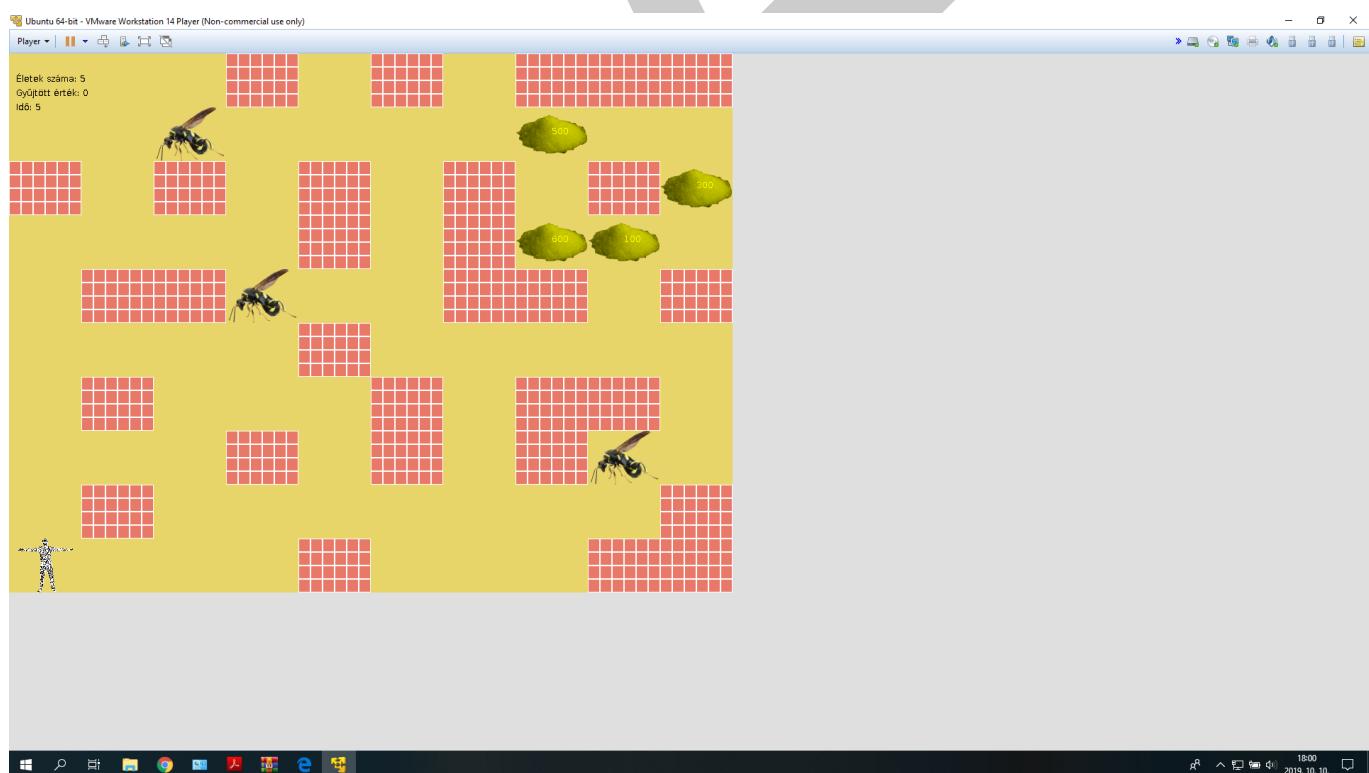
}

if(!dm1024x768)
 System.out.println("Nem megy az 1024x768, de a példa ←
 képméretei ehhez a felbontáshoz vannak állítva.");
```

### A program futtatása:

```
javac javattanitok/labirintus/*.java
javac javattanitok/LabirintusJáték.java
java javattanitok.LabirintusJáték labirintus.txt
```

### Futás 1920x1080-as felbontással:



# 17. fejezet

## Helló, Stroustrup!

### 17.1. JDK Osztályok

Megoldás forrás: [GitLab](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy készítsünk olyan Boost C++ programot, ami megszámolja a JDK összes osztályát.

A Boost olyan C++ könyvtárakat tartalmaz, amik támogatást nyújtanak pélául a lináris algebrához, többszálas programozáshoz és unit testinghez.

Az osztályok számolására készült kód részt egy `read_file()` függvénybe írjuk. Ennek a két paramétere egy `path` objektum, és a egy számláló változó, ami az osztályok számát tárolja. A `path` osztály a Boost-ban az elérési utak tárolására és feldolgozására lett kitalálva. Ezek az útvonalak stringként vannak inicializálva, amit átadunk a `path` konstruktornak.

Egy `if` kifejezéssel megnézzük, hogy a megadott úton lévő `file regular file`-e. Ha igen létrehozunk egy `string`et, ami tartalmazza azt, hogy `.java`, tehát a kiterjesztést, amit keresünk. Ezt a `compare` függvénytel összehasonlítjuk az útvonal fájljának kiterjesztésével. Az útvonalból a `path.string()` függvénytel `string`et készítünk. Majd a `find_last_of()` függvénytel megkeressük hol van benne a legutolsó `/` karakter, és innen készítünk belőle egy `substring`et majd növeljük a számlálót.

Az `else` ágban azt nézzük meg, hogy a megadott útvonal könyvtár-e. Ha igen, egy `for` ciklussal végigmegyünk rajta, ehhez a Boost `directory_iterator()` függvényét használjuk. Itt már csak rekurzívan meghívjuk az előző ágban elkészített függvényt. Így a függvény addig hívódik meg amíg elég "mélyen" nem vagyunk a könyvtárban, hogy megtaláljuk a szükséges fájlokat.

A `main()`-ben néhányt rendszerhívással kicsomagoljuk egy mappába a JDK-t. Az útvonalat deklaráljuk, mint argumentum. Majd meghívjuk a `read_file()` függvényt.

A kód:

```
//g++ jdk_classes.cpp -o jdk_classes -lboost_system -lboost_filesystem -lboost_program_options -std=c++14
//./jdk_classes mappa_nev

#include <iostream>
```

```
#include <string>
#include <fstream>
#include <iomanip>
#include <vector>

#include "boost_1_71_0/boost/filesystem.hpp"

using namespace std;

void read_file (boost::filesystem::path path, int &szamlalo)
{
 if (is_regular_file (path)) {

 string ext (".java");
 if (!ext.compare (boost::filesystem::extension (path))) {

 string actjavaspath = path.string();
 size_t end = actjavaspath.find_last_of ("/");
 string act = actjavaspath.substr (0, end);

 szamlalo++;

 }

 } else if (is_directory (path))
 for (boost::filesystem::directory_entry & entry : boost::filesystem::directory_iterator (path))
 read_file (entry.path(), szamlalo);

}

int main(int argc, char *argv[])
{
 system("mkdir src");
 system("cd src");
 system("unzip src.zip -d src");
 string path="src";
 boost::filesystem::path a(argv[1]);
 int szamlalo=0;
 read_file(a, szamlalo);
 cout<< "Az src.zip-ben található java osztályok száma: "<< szamlalo << endl;
 return 0;
}
```

}

```
user@ubuntu: ~/Asztal/Prog2_source/stroustrup/jdk
Fájl Szerkesztés Nézet Keresés Terminál Súgó
inflating: src/jdk.xml.dom/org/w3c/dom/xpath/package-info.java
inflating: src/jdk.xml.dom/org/w3c/dom/xpath/XPathEvaluator.java
inflating: src/jdk.xml.dom/org/w3c/dom/xpath/XPathException.java
inflating: src/jdk.xml.dom/org/w3c/dom/xpath/XPathExpression.java
inflating: src/jdk.xml.dom/org/w3c/dom/xpath/XPathNamespace.java
inflating: src/jdk.xml.dom/org/w3c/dom/xpath/XPathNSResolver.java
inflating: src/jdk.xml.dom/org/w3c/dom/xpath/XPathResult.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ByteArrayChannel.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/JarFileSystem.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/JarFileSystemProvider.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipCoder.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipConstants.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipDirectoryStream.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileAttributes.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileAttributeView.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileStore.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileSystem.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileSystemProvider.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipInfo.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipPath.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipUtils.java
inflating: src/jdk.zipfs/module-info.java
Az src.zip-ben található java osztályok száma: 17560
user@ubuntu:~/Asztal/Prog2_source/stroustrup/jdk$
```

## 17.2. Másoló-mozgató szemantika és Összegzés

Megoldás forrása:

[GitLab Másoló konstruktur](#)

[GitLab Mozgató konstruktur](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy kódcsipeteken keresztül vessük össze a C++ 11 mozgató és másoló szemantikáját.

A két szemantika személgettésére egy egyszerű String osztályt alkalmaztam. Tartalmaz egy char\*-okat tartalmazó tömböt ami a buffer, illetve egy m\_Size változót, ami ennek a méretét adja meg. A konstruktőrben meghatározzuk a string hosszát strlen() függvényel, majd ennek megfelelően hozunk létre memória foglalást a tömb számára. Memcpy() függvényel másoljuk be a Bufferbe a stringet. Ez a függvény a megadott területre másolja a megadott forrást, a másolókonstruktőrben is ezt a függvényt fogjuk használni. A destruktőrben egyszerűen kitöröljük a buffert. A kiiratásra az ostream operator túlerhelését alkalmazzuk, amit friend-ként deklarálunk hogy a privát tagokhoz is hozzáférhessen.

A String osztály:

```
#include <iostream>
#include <string.h>
```

```
lass String{
private:
 char* m_Buffer;
 unsigned int m_Size;
public:
 String(const char* string)
 {
 m_Size = strlen(string);
 m_Buffer = new char[m_Size + 1];
 memcpy(m_Buffer, string, m_Size);
 m_Buffer[m_Size] = 0;
 }

 ~String() {
 delete[] m_Buffer;
 }

 char& operator[](unsigned int index)
 {
 return m_Buffer[index];
 }

 friend std::ostream& operator<<(std::ostream& stream, const String& string);
};

std::ostream& operator<<(std::ostream& stream, const String& string){
 stream << string.m_Buffer;
 return stream;
}
```

### Másoló szematika:

A másoló szematikához egy másoló konstruktur elegendő. C++-ban a programunkban alapból is van egy másoló konstruktur, de ez csak úgynevezett "shallow copy"-t készít. Ilyenkor az objektum tagjai egy új memóriacímre kerülnek. De a char\* ugyanarra a memóriacímre fog mutatni, mint az eredeti és ebből problémák származhatnak. Elsősorban például, ha megváltoztatjuk az új másolt objektumot, például a 3. karakterét átírjuk 'b'-re, akkor az eredeti objektumban is bekövetkezik a változás.

```
int main(){
 String string = "test";
 String string2 = string;
 string2[2]='b';

 std::cout<< string << std::endl;
 std::cout<< string2 << std::endl;
}
```

```
user@ubuntu:~/Asztal$ g++ copy.cpp -o copy
user@ubuntu:~/Asztal$./copy
tebt
tebt
```

A másik probléma, hogy, mivel a két memóriacím megegyezik, ezáltal mikor meghívódik a destruktur, felszabadítja a memóriaterületet, majd újra felszabadítaná ugyanazt a memóriaterületet a másolt objektum miatt.

Amit a saját másoló konstruktorunkban, akorunk, hogy másolásnál új char\* tömb allokálódjon, ne pedig a két char\* ugyanarra a memóriára mutasson. Ezáltal a másolt stringnek saját memóriacíme lesz, és a pointere is új címre fog mutatni. Ezáltal, ha megváltoztatjuk az egyiket az nincs kihatással a másikra. Ezt a fajta másolást már "deep copy"-nak nevezzük, mert az egész objektumot le fogja másolni és figyelembe veszi a memóriaterületet amire a pointer mutat. A konstruktorban a m\_size változót lehetőségünk van "shallow copy"-val másolni, ugyanis ez egy integer, és nem okoz nagy gondot neki memóriát foglalni. Ezután allokálunk egy új Buffert, akkora méretű mint az m\_size. Majd használjuk a memcpy() függvényt és lemásoljuk a régi objektum bufferét az új objektum bufferébe.

```
String(const String& other) : m_Size(other.m_Size)
{
 std::cout<<"Copy ctor"<<std::endl;
 m_Buffer = new char[m_Size + 1];
 memcpy(m_Buffer, other.m_Buffer, m_Size+1);
}

int main(){
 String string = "test";
 String string2 = string;
 string2[2]='b';

 std::cout<< string << std::endl;
 std::cout<< string2 << std::endl;
```

```
user@ubuntu:~/Asztal$ g++ copy.cpp -o copy
user@ubuntu:~/Asztal$./copy
Copy ctor
test
tebt
```

### Mozgató szematika:

A mozgató szemantika, esetén lehetőségünk van néhány feltétel betartásával arra, hogy az egyik objektum átvegye a másiktól az erőforrásokat. A mozgatás közben például ha egy pointer tagot mozgatunk a mutatott memória átkerül az új taghoz. A mozgatás megértéséhez fontos tudni az lvalue és az rvalue közötti különbséget. Lvalue-nak nevezzük azokat az értékeket, amik olyan objektumot jelölnek aminek van egy biztos helye a memóriában. Az rvalue pedig olyan kifejezés, ami nem jelöl ilyen objektumot. Például int a = 1 egy lvalue, rvalue például egy olyan ideiglenes eredményei kifejezeknek mint példul az (a+1). Ennek nincs lefoglalt memóriabeli helye

```
int a = 1; //lvalue
(a+1) //rvalue
```

A mozgatás lvalue-k esetén veszélyes lehet, de rvalue esetén nem mert miután a konstruktur lefutott az ideiglenes értéket nem tudjuk újra használni. A mozgatásra az rvalue referenciáját használjuk, ami a C++ 11-ben jött be és &&-el jelöljük, "sima" & referenciát pedig lvalue referenciának tekintjük. Ha függvény-paraméterként adunk meg rvalue referenciát, akkor automatikusan le fogja tiltani az lvalue megadását, ez a mozgatókonstruktornál nagyon hasznos, mert lvalue-t nem akarunk mozgatni.

Fontos hogy a függvény futásakor létrehozásra kerül a paraméterként megadott rvalue referenciával magadott típusú változóból egy ideiglenes objektum. Ez a mozgatókonstruktorkban ugye a String-ként szerepel. Itt egyszerűen megtesszük azt, hogy lemásoljuk az adott objektum pointerét, ezesetben a buffert. Majd a régit pedig egyszerűen nullptr-re állítjuk. Ezt azért tehetjük meg, mert a fordító nem érzékeli, hogy módosítás történt, erre csak akkor van lehetőség ha a rvalue referenciát kap a konstruktur paraméterként. Ilyenkora konstruktorkon belül "akármit" megtehetünk, ameddig az szemantikának érvényes.

```
String(String&& other) {
 std::cout<<"Move ctor\n";
 m_Buffer = other.m_Buffer;
 other.m_Buffer = nullptr;
}
```

A mozgató operátor dolga az, hogy a régi objektumhoz tartozó erőforrásokat törölje és az újhoz pedig beszerezze a szükséges erőforrásokat. Ezt megtehetjük manuálisan, a következő módon:

```
String& operator=(String&& other) {
 if(this != &other) {
 delete m_Buffer;

 m_Buffer = other.m_Buffer;
 other = nullptr;
 }
 return *this;
}
```

Ekkor rvalue referenciákat kell megadni az objektumot. Magunknak kell feloldani a Buffer átlal lefogalalt memóriát a delete függvényvel, itt töröljük a lefogalat erőforrásokat. Majd elvégezni a cserét, tehát az új Buffer lesz a régi, a régi pedig nullptr, ez pedig az új erőforrások beszerzése rész.

A másik módszer az a ha használjuk a swap() függvényt. Itt nem rvalue paramétert vár a függvény, mert a mozgató konstruktur fogja inicializálni. Ettől függetlenül rvalue-t kell megadni mert a mozgató konstruktur azt várja. Ekkor a régi memória felszabadítása automatikusan történik. Az régi és új pointert megcseréljük a swap() függvényvel. És az objektumra mutató \*this pointert visszadjuk.

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

```
int main() {
 String string1 = "test";
 std::cout<<"Az eredeti string: "<<string1<<std::endl;
 String string2 = std::move(string1);
 std::cout<<"A másolt string: "<<string2<<std::endl;
```

```
 std::cout<<"Az eredeti string másolás után: "<<string1<<std::endl;
}
```

```
user@ubuntu:~/Asztal$./move
Az eredeti string: test
Move ctor
A másolt string: test
Az eredeti string másolás után: user@ubuntu:~/Asztal$
```

## 18. fejezet

# Helló, Gödel!

### 18.1. Gengszterek

Megoldás forrás:[Bátfai Norbert GitHub](#)

Tanulságok, tapasztalatok, magyarázat...

Lambda kifejezéssel ideiglenes függvényeket írhatunk, a hagyományos függvényírás helyett.

Általános szintaxisa:

```
[captures] (params) { body }
```

Részei:

- []: A capture részben adhatunk meg változókat, objektumokat, amiket majd a kifejezésben használunk. Ezt megtehetjük csak a változók felsorolásával, vagy átadhatunk "mindent". [=] használatakor ez másolással, [&]-val pedig referenciaként történik.
- ( params ): Itt azokat a paramétereket adjuk meg, amiket a függvény használ.
- {body}: Ez a rész pedig maga a függvényben végrehajtódó kód.

A feladat az volt, hogy az OOCWC programban mutassuk be mikét használ a program a Gengszterek rendezésénél Lambda kifejezést.

A programban a Gengszterek egy vektorban vannak elhelyezve, erre alkalmazzuk a sort függvényt. A sort függvénynek a harmadik paramétere maga a függvény, ami az elemek összehasonlítását végzi, a lambda kifejezést itt fogjuk használni, mert saját hasonlító függvényre van szükség. A captures részben megadjuk a this pointerrel magát a gengszereket tartalmazó vektort, illetve egy cop paramétert. Függvényparaméterként használunk két Gengszer objektumot, amiket hasonlítani fogunk. Ezekre hívjuk a dst függvényt, ami azt adja vissza, hogy a gráfban milyen távol vannak a cop-tól. Ennek a távolságnak a mértéke alapján hasonlítjuk a két Gengszer objektumot. Ez lezajlik a vektor minden elemén és a cop-tól való távolság szerint fognak rendeződni növekvő sorrendben.

```
std::sort (gangsters.begin(), gangsters.end(), [this,cop](Gangster x, ←
 Gangster y)
{
```

```

 return dst(cop,x.to) < dts(cop,y.to);
});

```

## 18.2. Alternatív tabella rendezése

Megoldás forrása:[GitLab](#)

Tanulságok, tapasztalatok, magyarázat...

Az alternatív tabella egy a labdarúgró bajnokságokhoz egy olyan pontozási rendszer, ami a hagyományos tabellával ellentétben figyelembe veszi, a két csapat közti erősségi viszonyt.

| Hazai \ Vendég <sup>1</sup> | SIÓ | HON | VAS | DEB | FTC | ETO | KAP | KTE | PÁP | MTK | PAK | SZO | HAL | ÚJP | VID | ZTE |
|-----------------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| BFC Siófok                  |     | 1-3 | 0-0 | 4-1 | 1-1 | 0-1 | 0-0 | 1-2 | 0-1 | 0-0 | 1-3 | 1-1 | 0-0 | 1-1 | 1-1 | 0-4 |
| Budapest Honvéd             | 0-1 |     | 0-0 | 1-0 | 0-1 | 1-1 | 1-0 | 1-2 | 2-4 | 1-2 | 1-0 | 0-0 | 3-1 | 1-4 | 2-2 | 1-0 |
| Vasas                       | 3-0 | 3-2 |     | 1-5 | 1-3 | 2-1 | 1-3 | 1-1 | 1-1 | 1-1 | 2-3 | 3-0 | 2-1 | 1-0 | 0-0 | 2-0 |
| Debreceni VSC               | 2-3 | 2-2 | 3-1 |     | 2-1 | 1-1 | 3-1 | 6-2 | 2-0 | 3-1 | 2-1 | 4-0 | 1-2 | 1-1 | 3-1 | 2-1 |
| Ferencváros                 | 1-2 | 1-3 | 0-1 | 1-1 |     | 3-0 | 1-0 | 2-1 | 3-0 | 3-0 | 2-1 | 1-0 | 2-1 | 1-0 | 0-5 | 4-4 |
| Győri ETO                   | 1-0 | 3-0 | 0-1 | 3-0 | 1-0 |     | 2-0 | 2-1 | 0-1 | 1-1 | 1-1 | 4-2 | 2-2 | 2-1 | 1-1 | 0-1 |
| Kaposvári Rákóczi           | 3-0 | 0-0 | 0-1 | 0-0 | 2-1 | 3-0 |     | 2-1 | 3-2 | 2-1 | 1-2 | 0-1 | 1-0 | 0-1 | 1-4 | 2-1 |
| Kecskeméti TE               | 2-3 | 2-1 | 3-1 | 3-0 | 1-2 | 3-3 | 4-1 |     | 0-1 | 3-0 | 0-1 | 4-2 | 1-0 | 4-3 | 2-4 | 1-0 |
| Lombard Pápa                | 0-1 | 0-1 | 2-1 | 1-1 | 0-5 | 2-1 | 1-1 | 4-1 |     | 0-2 | 1-2 | 0-0 | 5-1 | 3-1 | 1-2 | 4-3 |
| MTK Budapest                | 1-2 | 3-1 | 4-0 | 0-0 | 1-3 | 0-0 | 0-2 | 4-2 | 2-1 |     | 3-4 | 1-0 | 0-3 | 1-0 | 0-3 | 3-4 |
| Paksi FC                    | 0-0 | 0-1 | 3-0 | 2-2 | 3-2 | 2-1 | 2-3 | 2-0 | 4-0 | 1-1 |     | 3-1 | 2-1 | 2-1 | 1-0 | 2-2 |
| Szolnoki MÁV FC             | 0-4 | 0-2 | 0-1 | 1-2 | 2-3 | 0-3 | 1-2 | 2-1 | 2-2 | 2-1 | 3-1 |     | 0-0 | 1-0 | 1-1 | 1-3 |
| Szombathelyi Haladás        | 4-0 | 1-1 | 3-0 | 3-0 | 1-1 | 3-3 | 2-0 | 1-0 | 1-0 | 2-0 | 1-2 | 3-1 |     | 0-2 | 2-0 | 0-0 |
| Újpest                      | 1-1 | 3-1 | 2-2 | 2-2 | 6-0 | 0-0 | 3-2 | 2-1 | 2-1 | 2-1 | 2-3 | 1-0 | 3-1 |     | 1-0 | 4-2 |
| Videoton                    | 2-0 | 0-2 | 3-0 | 2-1 | 1-1 | 2-1 | 3-1 | 2-2 | 4-0 | 2-1 | 2-1 | 3-1 | 3-1 | 1-0 |     | 3-0 |
| Zalaegerszegi TE            | 2-1 | 2-1 | 2-1 | 1-1 | 2-1 | 1-1 | 3-5 | 2-1 | 3-1 | 1-0 | 1-0 | 2-1 | 1-1 | 2-1 | 1-2 |     |

A megadott kereszttábla alapján a következők a pontozások:

- üres: 0
- zöld: 1
- sárga: 2
- piros: 3

A kereszttábla pontjait a Wiki2Matrix programban megadjuk egy Mátrixban, és a program futásakor egy linkmátrixot kapunk, amit megadunk az AlternativTabella programnak. A program pontszámítási része nem volt feladat, csak az hogy miként használja a Comparable<T> interfészet.

A Comparable<T> interfész egyetlen metódust a compareTo(T obj) metódust tartalmazza. A paramétere az adott objektumtípus amit szeretnénk összehasonlítni. A program esetében ez a csapat objektem. A visszatérési értéke egy integer, az alapján hogy milyen a két objektum viszonya.

- Ha az első kisebb mint a második akkor: negatív integer
- Ha egyenlőek akkor: 0
- Ha az első nagyobb mint a második akkor: pozitív integer

A programban ez a következőképpen van jelen: A Csapat osztály implementálja a Comparable interfészt, a compareTo() metódusok pedig a csapat objekumok érték tagját fogja egymáshoz viszonyítani.

```
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;
 }
 }
}
```

Erre az interfészre a következő sorok miatt is szükségünk van.

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList(←
 csapatok);
java.util.Collections.sort(rendezettCsapatok);
```

Az első sorban még nincsen probélma, itt létrehozunk egy listát, fontos megjegyezni hogy ez az adatszerkezet is a Collection interface-t egészíti ki. Ebbe a listába a csapatokat tömbjét fogjuk konvertálni az Arrays.asList() metódus segítségével.

A Comparable<T> interfész a második sor miatt is szükséges. A sort() metódussal listákat rendezhetünk a természetes sorrendjük alapján. Ezt a természetes rendezés a Comparable<T> interfész által létrehozott rendezés, tehát a sort() a Comparable interfészt egészíti ki, ezért kell nekünk is implementálni.

```
static <T extends Comparable<? super T>>
void
```

Returns an immutable map, mapping only the specified key to the specified value.

sort(List<T> list)

Sorts the specified list into ascending order, according to the natural ordering of its elements.

18.1. ábra. sort metódus JDK leírás

```

0.08997865094516633
0.12009327027808978
0.07575767716539045
0.10749660836285888
0.052391857703979944
0.07292288727378253
0.09346964541402003
0.08360553434843401
0.10851105424424129
0.08422951306493812
0.11154330119909847

Csapatok rendezve:

|- Ferencváros
 46
 Debrecen
 0.1200
-
 Vidi
 40
 Újpest
 0.1115
-
 Újpest
 36
 Paksi
 0.1085
-
 Honvéd
 35
 Ferencváros
 0.1074
-
 Debrecen
 34
 Mezőkövesd
 0.0934
-
 MTK
 30
 Honvéd
 0.0899
-
 Paksi
 30
 Puskás Akadémia
 0.0842
-
 Mezőkövesd
 29
 MTK
 0.0836
-
 Puskás
 24
 Diósgyör
 0.0757
-
 Diósgyör
 24
 Kisvárda
 0.0729
-
 Kisvárda
 24
 Haladás
 0.0523
| -
```

18.2. ábra. A program futása a 2019/20-as magyar bajnokság eddigi állása alapján

### 18.3. Gimp Scheme hack

Megoldás forrása:[Bátfai Norbert GitLab](#)

Tanulásgok, tapasztalatok, magyarázat...

**Króm effekt:**

A program első részében deklarálunk néhány függvényt: A color-curve függvényben eltárolunk egy tömbben 8 értéket, amik a GIMP színgörbékének bizonyos értékei. Az elem függvénnyel az kérjük le egy listáról, hogy a megadott elem hányadik. A lisp-ben a változók lehetnek listák tehát több elemük is lehet. A wh függvényben az adott szöveg magasságát és szélességét kérjük le.

Létrehozzuk a chrome függvényt amelynek a paraméterei:

- szöveg
- betűtípus
- betűméret
- szélesség
- magasság
- szín
- színátmenet

A további részekben lépésenként haladunk a szöveg változtatásával amelyek a következők:

- 1: Létrehozzuk egy fekete háttérréteget egy fehér középre toltszöveggel.
- 2: Egy erős gauss elmosódást rakunk a képre.
- 3: A szinek határszintjének alsó és felső határának megváltoztatásával élesítünk az elmosódott képen.
- 4: Újra alakalmazzuk a gauss elomsást.
- 5: Szín szerinti kijelöléssel kijelöljük a fekete területet.
- 6: Létrehozunk majd kijelölünk egy átlásztó réteget.
- 7: Az előbb létrehozott átlásztó réteget kitöljük egy sötétszürkéből világosszürkébe való színátmenettel.
- 8: Bucka leképezést alkalmazunk a második réteget az első réteg használatával.
- 9: A színgörbék módosítjuk, ezeken egy hullám mintát állítunk be.

A program végén elmentjük a GIMP menüjébe a scriptet és adunk neki alapértelmezett paramétereket.

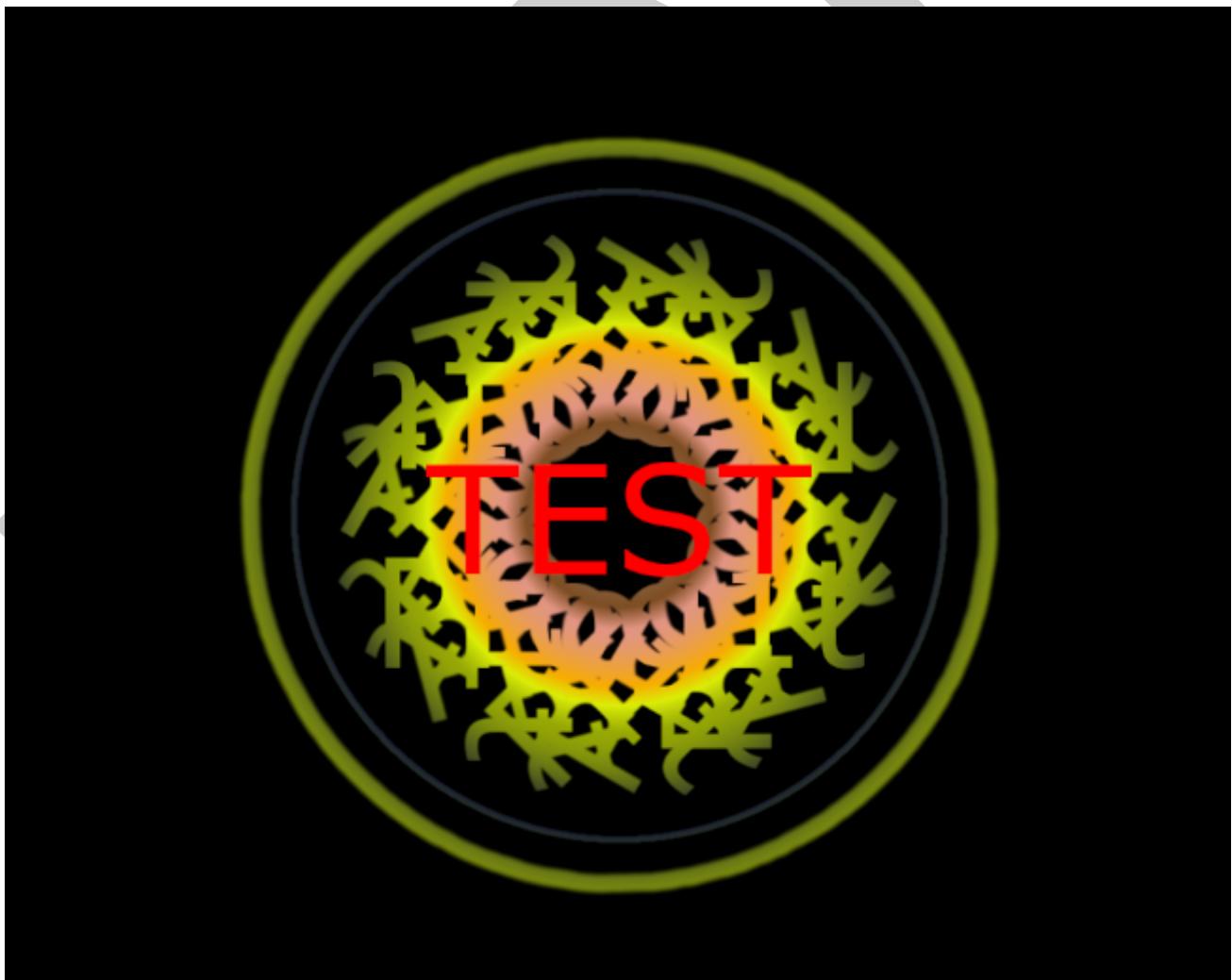


**Mandala:**

A programot néhány függvény deklarálásával kezdjük. Fontos megjegyezni, hogy lisp-ben a változók listák tehát egyszerre több elemük is lehet. Az elem függvénnyel azt kérjük le egy listáról, hogy a megadott elem hányadik.Lekérjük a szöveg szélességét a text-width függvénnyel. Alapból a beépített gimp függvény több értéket ad vissza, ezért használjuk a car függvényt, hogy csak a legelső a paramétert adjuk vissza tehát a szélességét. A text-wh függvénnyel egy hasonló módszerrel egy listába lekérjük az adott szöveg szélességét és magasságát.

A mandala script definiálása 8 paraméterrel, amik a következők: szöveg amiből készül, szöveg középen, betűtípus, betű méret, szélesség, magasság, szöveg színének RGB kódja, színátmennet. A lisp-ben a változókat a let\* függvény segítségével definiáljuk. RGB képet hozunk létre a megadott szélességgel és magassággal. Elkészítjük a háttérréteget illetve a szükséges szövegeket. Beállítunk egy előtérszínt amivel a háttéröt kitöljük, majd értékül adjuk neki a paraméterként kapott színt. Beállítjuk ezen kívül a betűméretet és betűtílust.

- Elforgatjuk a szövegeket úgy hogy a pi adott hatványaival szorozzuk fokszámukat.
- Elipszis kivágásokkal létrehozzuk a kereteket.
- Beállítjuk a színátmennet, majd létrehozzuk a középen elhelyezett szöveget.
- Regisztráljuk a GIMP fájlmenüjébe a scriptet és beállítunk alapértelmezett paramétereit.



## 19. fejezet

# Helló,!

### 19.1. FUTURE tevékenység editor

Megoldás forrása: [GitLab](#)

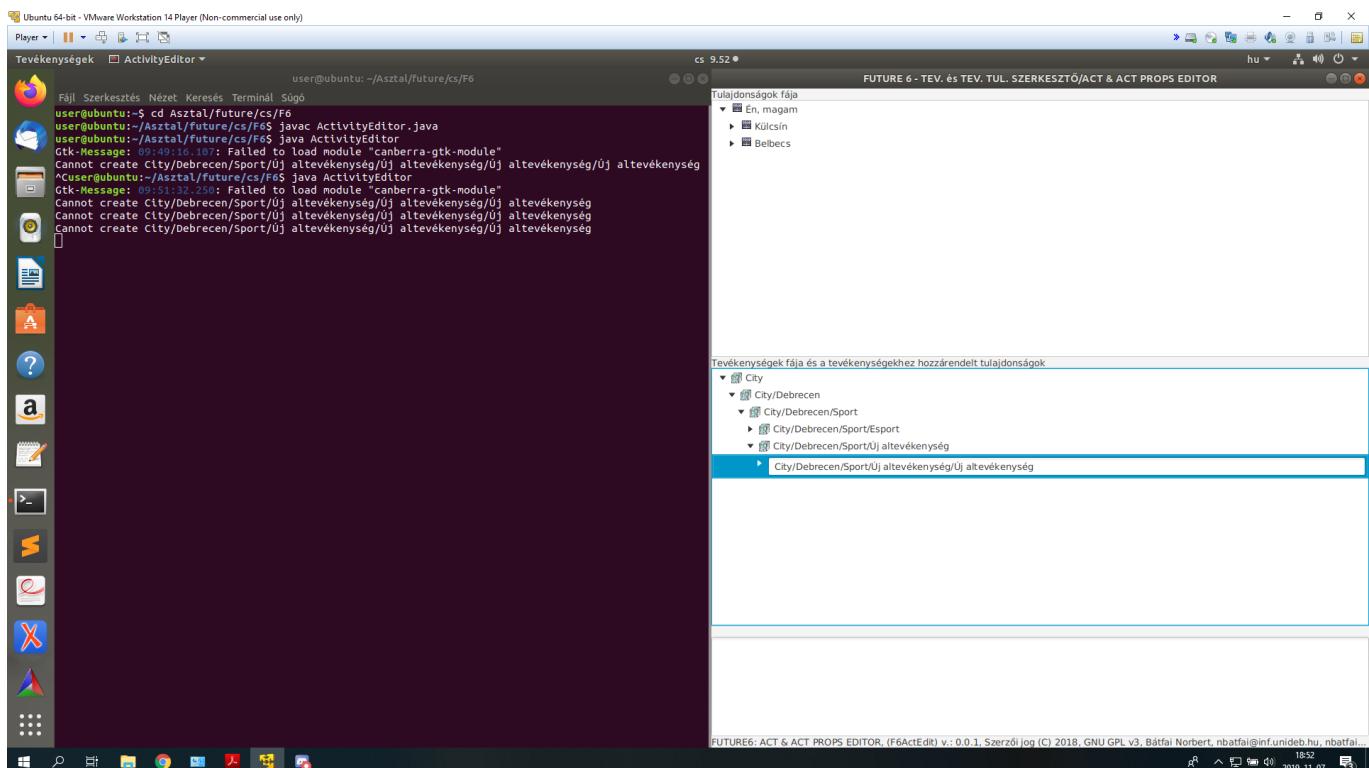
Tanulságok, tapasztalatok, magyarázat:

A feladat az volt, hogy a FUTURE Activity Editorban keressünk bugokat és javítsuk őket ki. Az ActivityEditor futtatásánál fontos megemlíteni, hogy Java 8 felett nem fut, és olyan verzió kell, ami tartalmazza a JavaFX -et. Több verziót is megpróbáltam a 8.0.202-zulufx-el futott hiba nélkül.

Ezt a verziót könnyedén telepíthetjük sdkman segítségével:

```
 sdk install java 8.0.202-zulufx
 sdk use java 8.0.202-zulufx
```

A bug, amit találtam az volt, hogy ha egy tevékenységen létrehozunk egy újt altevékenységet, akkor azután már nem lehetett másikat létrehozni. Ennek az az oka, hogy ilyenkor könyvtárszerkezetet hozunk létre, és két ugyanolyan nevű mappa keletkezne.



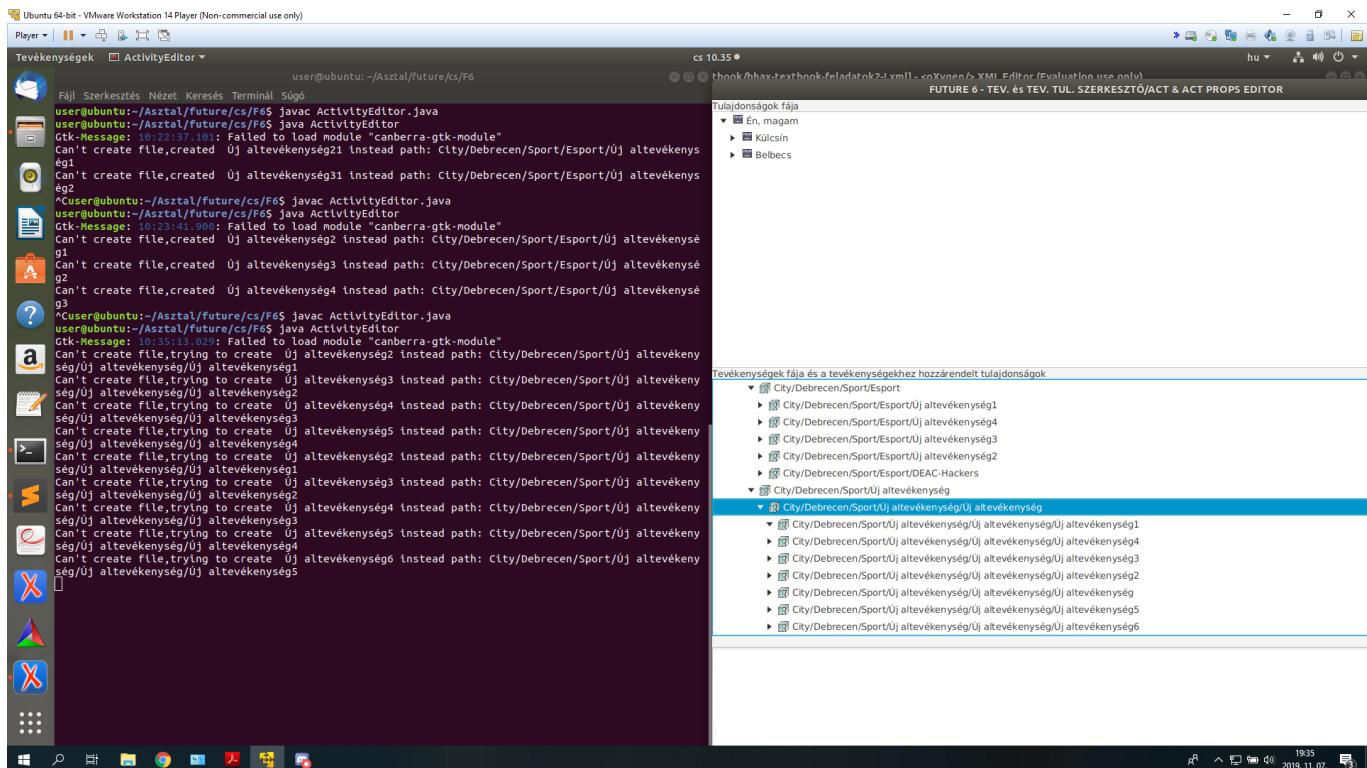
Ezt a következőképpen javítjuk ki a kódban: Deklarálunk egy i ciklusváltozót, és egy végtelen while ciklusba tesszük az Új altevékenység létrehozásának programrészét. Létrehozunk a könyvtárban egy Új Altevékenységet, a nevében a while aktuális indexével. Az if részben azt nézzük meg, hogy létezik-e a könyvtárban már, az altevékenység amit készülünk létrehozni. Ha nem, akkor break-el megállítjuk a ciklust és minden további nélkül létrehozhatjuk az új Altevékenységet. Ha már van ilyen nevű mappa, akkor növelnünk kell a tevékenység indexét tehát a ciklusváltozót. Majd kiírjuk hogy nem sikerült létrehozni a fájlt és újat próbálunk létrehozni. Ezt követően a while ismét lefut növelt i-val. A működés hátránya lehet hogy ahány altevékenység létezik már alapból, annyiszor fog lefutni a while, amíg újat tud létrehozni.

```

int i=1;
while(true) {
 java.io.File f = new java.io.File(file.getPath() + System. ←
 getProperty("file.separator") + "Új altevékenység"+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);
 break;
 } else {
 ++i;
 System.out.println("Can't create file, trying to create ←
 +" Új altevékenység"+i+" instead+" path: "+f. ←
 getPath());
 }
}
);

```



19.2. ábra. Bug javítva

## 19.2. SamuCam

Megoldás forrása: [Bátfai Norbert GitHub](#)

Tanulságok, tapasztalatok, magyarázat:

A feladat az volt, hogy mutassuk be a webkamera kezelését a SamuCam programnak.

A **main.cpp**-ben, itt a `webcamipOption` parancssori argumentumként lesz megadva, ennek kell tartalmaznia a webcam IP-jét, erre akkor van szükség, ha androidos telefon webkamerját használjuk. Ha nem adjuk meg az argumentumot akkor, az alapértétek kerül megadásra.

```
std::string videoStream = parser.value(webcamipOption).toStdString();
SamuLife samulife(videoStream, 176, 144); // (34, 16);
```

A további kódcsipetek, mind a **SamuCam.cpp**-ben találhatóak. Ebben a részletben láthatjuk, hogy a SamuCam konstruktőr tartalmaz egy `openVideoStream()` függvényt, amit rögtön utána ki is fejtünk. A `videoCaptureOpen(0)` függvénnnyel nyitjuk meg a streamet, itt a paramétert átírtam 0-ra mivel így automatikusan az alapértelmezett webcamot fogja megnyitni. A `.set` függvényekkel beállítjuk a videó magasságát, szélességét illetve az FPS-t.

```
SamuCam::SamuCam(std::string videoStream, int width = 176, int height = 144)
 : videoStream(videoStream), width(width), height(height)
{
```

```
 openVideoStream();
}

SamuCam::~SamuCam ()
{
}

void SamuCam::openVideoStream()
{
 videoCapture.open (0);

 videoCapture.set (CV_CAP_PROP_FRAME_WIDTH, width);
 videoCapture.set (CV_CAP_PROP_FRAME_HEIGHT, height);
 videoCapture.set (CV_CAP_PROP_FPS, 10);
}
```

Ebben a részben példányosítunk egy CascadeClassifier objektumot. A kaszkádolás a gépi tanulásnak egy fajtája, ahol a Classifier outputjából tanul a gép elmenti az infót, és a következő Classifierrel ezt felhasználja. Ebben az esetben emberi arcot ír le a classifier, ehhez töltjük le futtatás előtt a [https://github.com/Itseez/opencv/raw/master/data/lbpcascades/lbpcascade\\_frontalface.xml](https://github.com/Itseez/opencv/raw/master/data/lbpcascades/lbpcascade_frontalface.xml) oldalról az xml-t. Ezt egy load függvénytel töltjük be, ha nem találja a program debug üzenetet dob.

```
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;
}
```

Egy while függvényt használunk, amiben a read függvénytel fogujuk olvasni a bemenetet, a programrész végén található msleep(80) függvénytel 80 millisecet késleltetjük a while-t így ilyen időközönként fogujuk olvasni a bemenetet. Ezt a frame tömbbe fogjuk tárolni, ami egy többdimenziós tömb. Ha látjuk, hogy van bemenet tehát a frame tömb nem üres akkor, ezt a resize függvénytel átméretezzük és az INTER\_CUBIC függvénytel interpoláljuk.

Létrehozunk egy faces vektort és grayFrame tömböt. A cvtColor függvénytel a frame függvényt átalíjtuk szürkeárnyalatosra és, ez a verzió a grayFrame tömbben lesz tárolva. Ennek a tömbnek az equalizeHist függvénytel fogjuk megnövelni a kontrasztját. Ezt követően használjuk a faceClassifier-t a detectMultiScale függvény a bemeneten keres különböző méretű objektumokat, jelen esetben arcokat. Ha talál arcot létrehoz egy QImage-et majd emit-et küld egy faceChanged signalt a SamuBrain osztálynak küldünk tovább. Az arcra készítünk egy keretet a rectangle függvénytel és küldünk signalt SamuLife rész számára.

```
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,
 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);

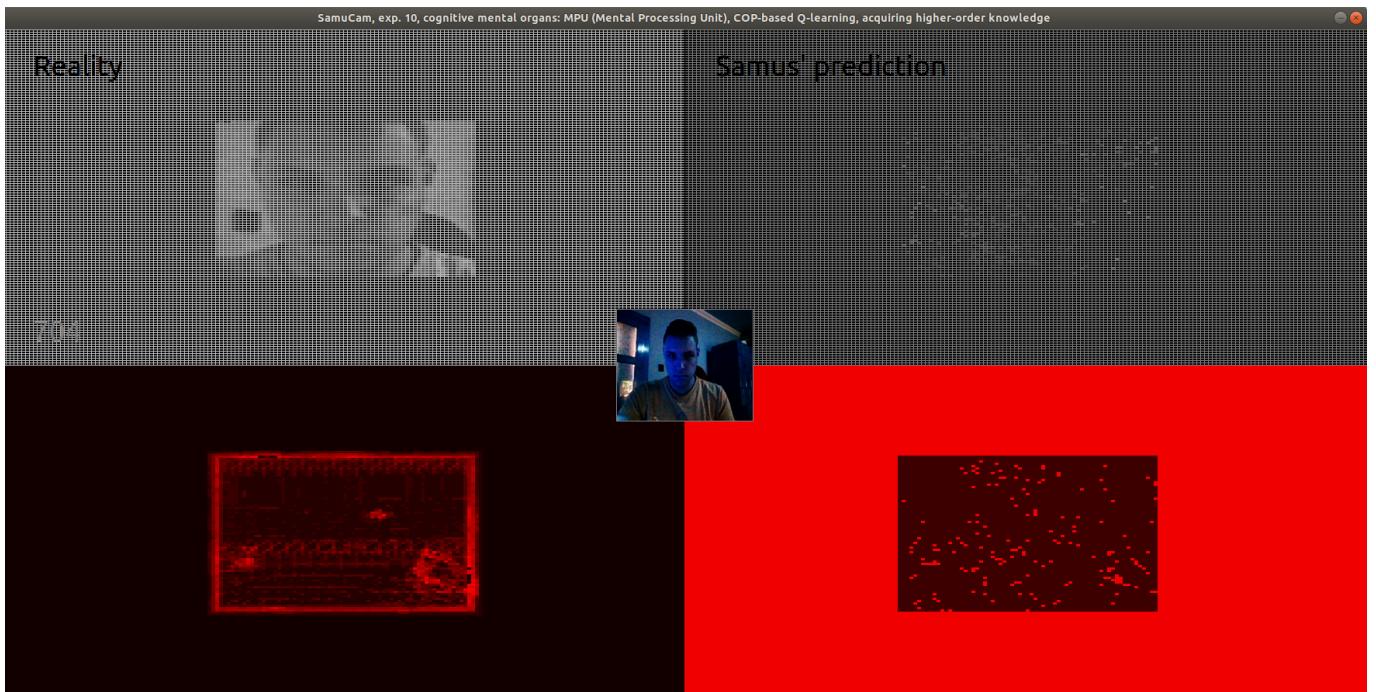
 cv::Point x (faces[0].x-1, faces[0].y-1);
 cv::Point y (faces[0].x + faces[0].width+2, faces[0].y + faces[0].height+2);
 cv::rectangle (frame, x, y, cv::Scalar (240, 230, 200));
}

emit faceChanged (face);
}

QImage* webcam = new QImage (frame.data,
 frame.cols,
 frame.rows,
 frame.step,
 QImage::Format_RGB888);

emit webcamChanged (webcam);
}

QThread::msleep (80);
}
```



### 19.3. BrainB Slot-signal mechanizmus

Megoldás forrása:[Bátfai Norbert GitHub](#)

Tanulságok, tapasztalatok, magyarázat...

#### Slot-signal mechanizmus általánosan:

A QT slot-signal mechanizmus az objektumok közötti kommunikációt teszi egyszeűbbé. Más GUI rendszerekben az objektumok közötti kommunikáció úgynevezett callback-ekkel történik, ami függvénypointerek által működik.

A QT-ban bevezették az Signal-slot mechanizmust, a callback-ek alternatívájára. A slot egy függvény, ami akkor hívódik meg ha kap signalt. A callback-ekhez képest nagy előnyük, hogy típus biztonságosak. A signal és slot típusának meg kell egyeznie, ha nem így van a jelez a fordító. Viszont akármennyi paraméterük lehet és akármilyen típusúak lehetnek. Ha a singal-hoz kapcsolt objektum állapota megváltozik, akkor lefut a kapcsolt slot, ha több akkor ezek egymás után futnak le.

#### BrainB példák:

A **BrainWin.cpp**-ben ezt a mechanizmust használjuk a Hero-k frissítésére, illetve a Statok lekérésére. Láthatjuk hogy a Signal tartalmazza a heroesChanged függvényt, ami ha végbemegy meghívódik a kapcsolt Slot ami pedig az updateHeroes függvényt tartalmazza.

```
connect (brainBThread, SIGNAL (heroesChanged (QImage, int, int)) ,
 this, SLOT (updateHeroes (QImage, int, int)));
connect (brainBThread, SIGNAL (endAndStats (int)),
 this, SLOT (endAndStats (int)));
```

Ugyanez az endAndStats függvény esetén, ami akkor fut le ha a **BrainThread.cpp** fájlban a run() függvényben a lejár a futási idő. Maga az endAndStats függvény a játék végeztével kiírja debug üzenetként, hogy, milyen könyvtárban található az eredményünkről szóló txt.

Maga az endAndStats függvény a **BrainWin.cpp**-ben a játék végeztével kiírja debug üzenetként, hogy, milyen könyvtárban található az eredményünkről szóló txt.

```
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();
}
```



19.3. ábra. Felélesztve

## 20. fejezet

# Helló,Lauda!

### 20.1. Port Scan

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat:[GitLab](#)

A programban az argumentumként megadott IP címen található 1024 alatti TCP kapukkal létesítünk kapcsolatot. Ehhez létrehozunk egy socket objektumot a java.net csomagból. Az első paramétere lesz argumentumként megadott IP cím, a második pedig az adott portszám 1024-ig.

`Socket(InetAddress address, int port)`  
Creates a stream socket and connects it to the specified port number at the specified IP address.

20.1. ábra. Socket dokumentáció

Kivétel akkor keletkezik, ha az adott porton van szerver folyamat a futás során. Ha nincs kivétel csak a close() metódussal bezárjuk a socketet. A kivételkezelés szerepe itt az, hogy ha nem kezelnénk azt, hogy az adott porton nincs szerver folyamat, egyszerűen leállna a program és nem tudnánk végigfuttatni a scan-t.

A program:

```
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);

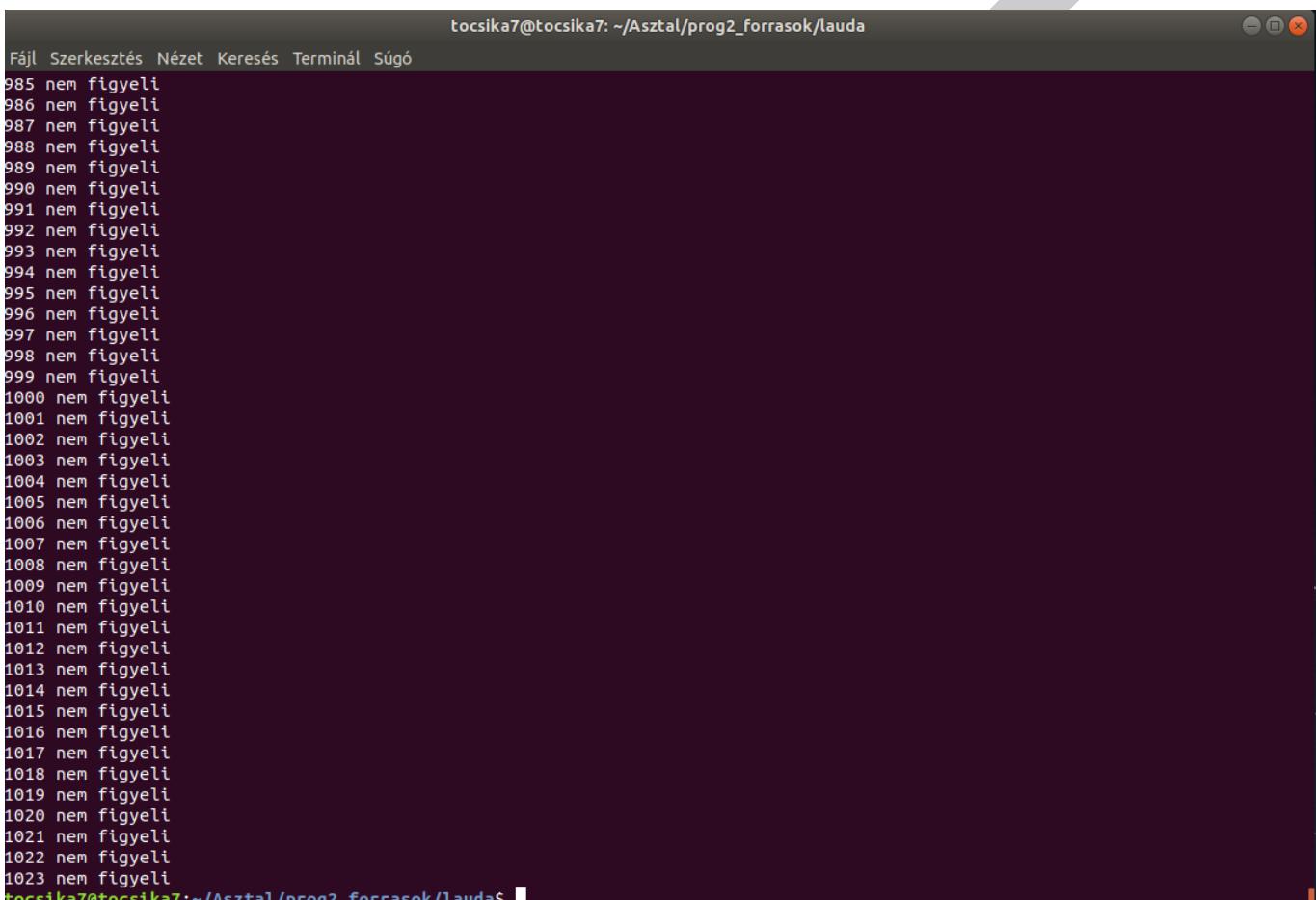
 System.out.println(i + " figyeli");

 socket.close();
 } catch (IOException e) {
 System.out.println("Nincs szerver "+i);
 }
 }
}
```

```
 } catch (Exception e) {

 System.out.println(i + " nem figyeli");

 }
 }
}
```



```
tocsika7@tocsika7: ~/Asztal/prog2_forrasok/lauda
Fájl Szerkesztés Nézet Keresés Terminál Súgó
985 nem figyeli
986 nem figyeli
987 nem figyeli
988 nem figyeli
989 nem figyeli
990 nem figyeli
991 nem figyeli
992 nem figyeli
993 nem figyeli
994 nem figyeli
995 nem figyeli
996 nem figyeli
997 nem figyeli
998 nem figyeli
999 nem figyeli
1000 nem figyeli
1001 nem figyeli
1002 nem figyeli
1003 nem figyeli
1004 nem figyeli
1005 nem figyeli
1006 nem figyeli
1007 nem figyeli
1008 nem figyeli
1009 nem figyeli
1010 nem figyeli
1011 nem figyeli
1012 nem figyeli
1013 nem figyeli
1014 nem figyeli
1015 nem figyeli
1016 nem figyeli
1017 nem figyeli
1018 nem figyeli
1019 nem figyeli
1020 nem figyeli
1021 nem figyeli
1022 nem figyeli
1023 nem figyeli
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda$
```

20.2. ábra. Futás:

## 20.2. AOP

Megoldás forrása:[GitLab](#)

Tanulságok, tapasztalatok, magyarázat:

A feladat az volt, hogy írunk bele egy átszövő vonatkoztatást az LZWBinfa Java átíratába. Ehhez használnunk kell valamilyen aspektus orientált paradigmát. Az aspektus orientált programozás, egy olyan paradigma, amivel lehetőségünk arra, hogy a kódot egy külső forrásból módosítsuk úgynevezetett "pointcutok" segítségével. A programkódban join point-ok mutatják azt, hogy mely részei a programnak vannak saját, illetve külön modulban. A pointcut pedig ezeket a join point-okat jelöli a program végrehajtásában. Ez

nyilván akkor hasznos, ha egy kész programon akarunk változatásokat végreghajtani és nem akarunk egy esetlegesen hibás kódot beleírni.

A feladat végreghajtásához AspectJ-t használtam, ami a Java-hoz a legelterjedtebb aspektus orientált kiegészítő. Feladatként azt választottam, hogy a kiir() függvény inorder helyett postorder módon fogja kiírni a binfát. A kódot egy .aj fájlban írjak, ez az AspectJ kiterjesztése. A program egy privileged aspectben van, erre azért van szükség hogy a private tagokat is elérje. Ebben az esetben például a mélységet, ami a Csomopont osztályban egy private tag. Az around() al helyettesíthetjük azt a függvényhívást amit utána írnunk, a függvényhívást pointcuttal adjuk meg, ebben az esetben target és call pointcuttal. A target olyan join pointokra fog match-álni amit megadunk paraméterként, itt f-et adunk meg ami az LZWBinfा objektum. A call és az args együtt adják meg a függvényt amit helyettesíteni akarunk, a call-nál megadjuk a paraméterek típusát az argsnál pedig a nevét és erre keres majd egyezést a program. Itt a kiir metódust akarjuk helyettesíteni, aminek a paraméterei a Csomopont típusú elem illetve az Java output stream. Innentől már csak annyi a teendő hogy a metódusban kicseréljük a kiir() metódushívásait a postorder-nek megfelelően, tehát először az egyes majd a nullás gyermekek és végül pedig a gyökér.

```
privileged aspect BinfaAspect {

 void around (LZWBinfा. Csomopont elem, java.io.PrintWriter os , LZWBinfा ←
 f) :
 target (f) && call (void kiir (LZWBinfा. Csomopont, java.io.PrintWriter)) ←
 && args (elem,os)
 {
 if (elem != null) {
 ++f.melyseg;
 f.kiir (elem.egyesGyermek(), os);
 f.kiir (elem.nullasGyermek(), os);
 for (int i = 0; i < f.melyseg; ++i) {

 os.print ("---");
 }
 os.print (elem.getBetű ());
 os.print ("(");
 os.print (f.melyseg - 1);
 os.println (")");
 --f.melyseg;
 }
 }
}
```

```
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda/aop$ ajc LZWBinfा. java BinfaAspect.aj
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda/aop$ java -cp ./aspectjrt-1.6.8.jar:. LZW
Binfा input.txt -o output.txt
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda/aop$ █
```

20.3. ábra. Futtatás

```
tocsika7@tocsika7: ~/Asztal/prog2_forrasok/lauda/aop$./lauda
Fájl Szerkesztés Nézet Keresés Terminál Súgó
-----0(3)
-----1(2)
-----0(3)
----1(1)
----0(2)
----0(3)
----0(4)
---/(0)
---1(3)
---0(4)
---0(5)
--1(2)
--0(3)
--1(5)
--0(4)
--0(1)
--1(4)
--0(5)
--0(6)
--1(3)
--0(2)
--1(5)
--1(4)
--0(5)
--0(3)
--0(4)
--1(6)
--0(5)
depth = 6
mean = 4.875
var = 0.9910312089651149
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda/aop$
```

```
tocsika7@tocsika7: ~/Asztal/prog2_forrasok/lauda/aop$./lauda
Fájl Szerkesztés Nézet Keresés Terminál Súgó
-----0(3)
-----1(2)
-----0(4)
-----0(3)
-----0(2)
----1(1)
----0(5)
----0(4)
----1(3)
----1(5)
----0(4)
----0(3)
----1(2)
-----0(6)
-----0(5)
-----1(4)
-----1(3)
-----1(5)
-----0(5)
-----1(4)
-----1(6)
-----0(5)
-----0(4)
-----0(3)
-----0(2)
-----0(1)
---/(0)
depth = 6
mean = 4.875
var = 0.9910312089651149
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda/aop$
```

20.4. ábra. Az eredeti inorder és az AspectJ-s postorder kiiratás

## 20.3. JUnit teszt

Megoldás forrása: [GitLab](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy az LZWBinfa Java átíratához a mélységet és szórást kiszámoló függvénykhez írunk JUnit testet. Tesztelés során lehetőségünk van arra, hogy ellenőrizzük, hogy a programunk bizonos helyzetekben jól működik-e. Nem teljes tesztet végzünk a kódon, hanem Unit testet, ami annyiban tér el, hogy csak a kód bizonyos kisebb részeit például függvényeket vagy osztályokat tesztel. A feladatban a Java JUnit framework-jét használjuk a teszteléshez.

A teszteléshez a [progpater](#)-en található értékeket használtam. A kódban először importáljuk a szükséges library-keket, készítünk egy Binfa objektumot. Egy Stringben tároljuk el a Test inputot, amit a toCharArray() metódussal először egyenként berakunk egy char-ba és onnan továbbítjuk az egyBitFeldolg() metódusnak, mert az csak char-onként tudja kezelni a bemenetet. Ezt követően az assertEquals() metódussal fogjuk tesztelni a mélységet és a szórást. Itt paraméterként először beállítjuk az értéket amit várunk, ezt követően a tényleges értéket, amit kapni fogunk a getMelyseg() és getSzoras() metódusokból. Harmadik paraméternek pedig egy delta értéket állítunk be, ami lényegében az első két érték közötti maximális különbség. Az assertEquals() azt nézi meg hogy az első két paraméter eltéréssel megegyezik-e.

A kód:

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;
```

```
public class LZWBinFaTest {
 LZWBinFa binfa = new LZWBinFa();

 @Test
 public void BinfaTest(){

 String input ="01111001001001000111";

 for(char input_char : input.toCharArray()){
 binfa.egyBitFeldolg(input_char);
 }
 assertEquals(4,binfa.getMelyseg(),0.0);
 assertEquals(0.957427,binfa.getSzoras(),0.0001);
 }
}
```

```
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda/junit$ java -cp .:junit-4.1
3-rc-1.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore LZWBinFaTest
JUnit version 4.13-rc-1
.
Time: 0,005

OK (1 test)
tocsika7@tocsika7:~/Asztal/prog2_forrasok/lauda/junit$ █
```

20.5. ábra. Sikeres teszt

## 21. fejezet

# Helló,Calvin!

### 21.1. MNIST

Megoldás forrása: [TensorFlow GitLab](#)

Tanulságok, tapasztalatok, magyarázat:

A feladat az volt, hogy próbáljuk ki az MNIST-et, illetve készítsünk hozzá egy saját 8-ast tesztelésre. A MNIST egy olyan program ami adatbázisból való tanítás után felismeri a kézzel írott egyjegyű számokat. A programhoz Python-t és az ehhez szükséges Tensorflow library-t használjuk. A Tensorflow az adatfolyam programozáshoz kínál nekünk lehetőségeket. Az adatfolyam programozásban az egyes prgoramrészletek, mint irányított gráfok értelmezzük és ezek között pedig adatfolyamok áramlanak.

A programban dekalrálunk egy függvényt a saját 8-ast ábrázoló képünk beolvasására. A file változóba a read\_file() függvénnyel olvassuk be a képet, majd a decode\_png() függvénnyel, ezt dekódoljuk unit8-as vagy 16-os tensorról.

```
def readimg():
 file = tf.io.read_file("sajat8a.png")
 img = tf.image.decode_png(file, channels=1)
 return img
```

A main függvényben beolvassuk az MNIST adabázis képeit majd létrehozunk egy modellt. A modellben az x egy placeholder, ami segítségével gráfot hozhatunk létre. A W és b Variable-ként vannak deklarálva, ami itt olyan tensorokat jelent amibe műveleteket végezhetünk a futás során. Fontos megemlíteni, hogy a b a biast tartalmazza. Az y-ban vesszük az a és W mátrixszorzatát a matmul() függvény segítségével, itt a súlyokat állítjuk be, de ehhez még hozzá kell adni a biast tehát a b-t.

```
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
```

A cross\_entropy-t a tartalmazza a y logits és y labels közötti cross entropy-t amit a reduce\_mean() függvényel átlagolunk. Tanításnál ezt az átlagot próbáljuk minimalizálni a gradient descent algoritmus segítségével. Ez a programban a train.GradientDescentOptimizer.minimize() függvényként jelenik meg.

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(←
 logits=y, labels=y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(←
 cross_entropy)
```

Ezután elkezdjük a tanítást az MNIST képeivel, majd teszteljük az adatázs 42. elemére ami egy 4-es, illetve a saját kezűleg rajzolt 8-asra is.

```
-- A halozat tesztelese
-- Pontosság: 0.9193

-- A MNIST 42. tesztképenek felismerése, mutatom a szamot, a továbblepeszhez csukd be az ablakat
-- Ezt a halozat ennek ismeri fel: 4

-- A saját kezi 8-asom felismerése, mutatom a szamot, a továbblepeszhez csukd be az ablakat
WARNING:tensorflow:From mnist.py:45: The name tf.read_file is deprecated. Please use tf.io.read_file instead.

[1114 18:13:41.150559 140441638950720 deprecation_wrapper.py:119] From mnist.py:45: The name tf.read_file is deprecated. Please use tf.io.read_file instead.

-- Ezt a halozat ennek ismeri fel: 8

tocsika7@tocsika7:~/Asztal/prog2_forrasok/calvin$ █
```

21.1. ábra. A program sikeresen felismeri a saját 8-ast

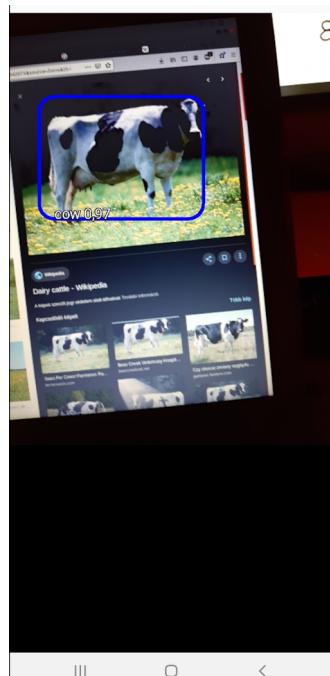
## 21.2. TensorFlow objektum detektáló

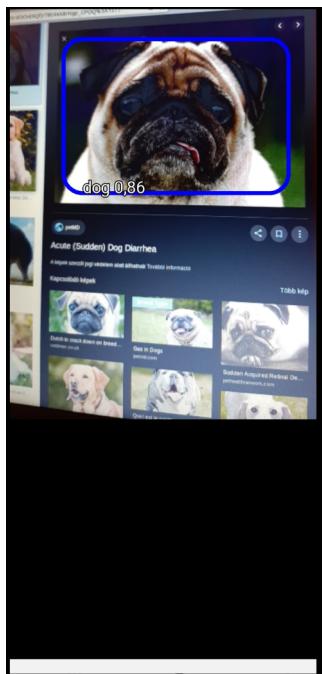
Megoldás forrása: [TensorFlow GitHub](#)

Tanulságok, tapasztalatok, magyarázat:

A feladat az volt, hogy próbáljuk ki a TensorFlow objektum detektáló appját androidra. Ehhez először le kell klónozni a TensorFlow android-os [repóját](#). Majd Android Studioba beimportálni és ott build-eltől egy .apk fájlt amit aztán telepíthetünk az Androidos telefonra. Ha nem szeretnénk magunknak build-eltől van lehetőség már kész appot is letölteni. A program viszonylag jól működött állatokat, informatikai eszközöket gond nélkül felismert.

Kipróbálva:





## IV. rész

### Irodalomjegyzék

DRAFT

## 21.3. Általános

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

## 21.4. C

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

## 21.5. C++

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

## 21.6. Lisp

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

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

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