

# Univerzális programozás

---

**Írd meg a saját programozás tankönyvedet!**

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

**COLLABORATORS**

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert	2019. április 11.	

**REVISION HISTORY**

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna <a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

## Ajánlás

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

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

# Tartalomjegyzék

<b>I. Bevezetés</b>	<b>1</b>
<b>1. Vízió</b>	<b>2</b>
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
<b>II. Tematikus feladatok</b>	<b>3</b>
<b>2. Helló, Turing!</b>	<b>5</b>
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	11
2.6. Helló, Google!	12
2.7. 100 éves a Brun tétel	14
2.8. A Monty Hall probléma	14
<b>3. Helló, Chomsky!</b>	<b>15</b>
3.1. Decimálisból unárisba átváltó Turing gép	15
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	15
3.3. Hivatkozási nyelv	16
3.4. Saját lexikális elemző	16
3.5. l33t.1	17
3.6. A források olvasása	19
3.7. Logikus	20
3.8. Deklaráció	21

<b>4. Helló, Caesar!</b>	<b>23</b>
4.1. int *** háromszögmátrix	23
4.2. C EXOR titkosító	24
4.3. Java EXOR titkosító	25
4.4. C EXOR törő	27
4.5. Neurális OR, AND és EXOR kapu	30
4.6. Hiba-visszaterjesztéses perceptron	32
<b>5. Helló, Mandelbrot!</b>	<b>33</b>
5.1. A Mandelbrot halmaz	33
5.2. A Mandelbrot halmaz a std::complex osztállyal	34
5.3. Biomorfok	35
5.4. A Mandelbrot halmaz CUDA megvalósítása	37
5.5. Mandelbrot nagyító és utazó C++ nyelven	40
5.6. Mandelbrot nagyító és utazó Java nyelven	43
<b>6. Helló, Welch!</b>	<b>47</b>
6.1. Első osztályom	47
6.2. LZW	49
6.3. Fabejárás	54
6.4. Tag a gyökér	55
6.5. Mutató a gyökér	67
6.6. Mozgató szemantika	67
<b>7. Helló, Conway!</b>	<b>69</b>
7.1. Hangyaszimulációk	69
7.2. Java életjáték	69
7.3. Qt C++ életjáték	77
7.4. BrainB Benchmark	77
<b>8. Helló, Schwarzenegger!</b>	<b>79</b>
8.1. Szoftmax Py MNIST	79
8.2. Szoftmax R MNIST	79
8.3. Mély MNIST	79
8.4. Deep dream	79
8.5. Robotpszichológia	80

<b>9. Helló, Chaitin!</b>	<b>81</b>
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .	81
9.2. Weizenbaum Eliza programja . . . . .	81
9.3. Gimp Scheme Script-fu: króm effekt . . . . .	81
9.4. Gimp Scheme Script-fu: név mandala . . . . .	81
9.5. Lambda . . . . .	82
9.6. Omega . . . . .	82
<b>10. Helló, Gutenberg!</b>	<b>83</b>
10.1. Juhász István: Magas szintű programozási nyelvek 1 . . . . .	83
10.2. Kernigan-Ritchie: A C programozási nyelv . . . . .	83
10.3. Benedek-Levendovszky: Szoftverfejlesztés C++ nyelven . . . . .	84
<b>III. Második felvonás</b>	<b>85</b>
<b>11. Helló, Arroway!</b>	<b>87</b>
11.1. A BPP algoritmus Java megvalósítása . . . . .	87
11.2. Java osztályok a Pi-ben . . . . .	87
<b>IV. Irodalomjegyzék</b>	<b>88</b>
11.3. Általános . . . . .	89
11.4. C . . . . .	89
11.5. C++ . . . . .	89
11.6. Lisp . . . . .	89

# Előszó

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

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

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

## Hogyan forgasd

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

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

## Milyen nyelven nyomjuk?

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

## Hogyan nyomjuk?

Ránts 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 találsz az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

## **I. rész**

### **Bevezetés**

# 1. fejezet

## Vízió

### 1.1. Mi a programozás?

### 1.2. Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [[KERNIGHANRITCHIE](#)]
- [[BMECPP](#)]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

### 1.3. Milyen filmeket nézzek meg?

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

## **II. rész**

### **Tematikus feladatok**

**Bátf41 Haxor Stream**

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

---

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:

Végtelen ciklus 1 mag 100%

```
#include <unistd.h>

int main ()
{
    for (;;){} //for ciklus kilépési feltétel nélkül

    return 0;
}
```

Végtelen ciklus minden mag 100%

```
#include <stdbool.h> //while ciklusban található true miatt kell

//gcc vegtelenossz.c -fopenmp -o vegossz

int main()
{
    #pragma omp parallel //Kiosszuk a feladatot a magok között
    while (true) //while ciklus kilépési feltétel nélkül
    {
        ;
    }
}
```

```
    return 0;
}
```

```
}
```

Végtelen ciklus proci 0%

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

```
int main ()
```

```
{
```

```
    for (;;) //for ciklus kilépési feltétel nélkül
```

```
        usleep (1); //az unistd.h -ban található usleep parancs "alvóba" teszi ↔  
        a processzort
```

```
    return 0;
```

```
}
```

```
}
```

Megoldás forrása:

Mindhárom esetben végtelen ciklusokat alkalmaztunk, mely lehet for vagy while is. Az érdekesebb része a több magra való feladat kiosztás jelentette "#pragma omp paralell", ezentúl ezen parancs segítségével meggyorsíthatjuk a programok futási idejét.

## 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 építő Lefagy2 már nem tartalmaz feltételezett, 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 használata nélkül!

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

Csere

```
#include <stdio.h>

int main()
{
    int a = 0;  /a inicializáció
    int b = 0;  /b inicializáció
    printf("Adja meg az a számot: ");    //kiaratás consolera printf függvény ←
        segítségével
    scanf("%d" , &a );                    //olvasás a consolebol a scanf ←
        fuggveny segitsegevel, "%d" = kapott ertek integer legyen,
    printf("Adja meg a b számot: ");
    scanf("%d" , &b);
    b = b-a;
    a = a+b;
    b = a-b;
    printf("a=%d%s",a,"\\n");    // "\\n" = sortörés
    printf("b=%d%s",b,"\\n");
}
```

Megoldás forrása:

Megcseréltünk két változó értékét bármiféle logikai utasítás vagy kifejezés nélkül, kipróbálhattuk a "printf" "scanf" függvényeket.

## 2.4. Labdapattogás

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

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

Megoldás forrása:

Labdaif

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

//gcc labdaif.c -o labda -lncurses      -igy kell futtatni kozolban

int
main ( void )    //void = nem ad vissza értéket
{
    WINDOW *ablak;      //ablak -ra mutató pointer
    ablak = initscr ();  // inicializáljuk a curses ablakot

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );    //jelenlegi kurzor koordinátákat ↔
        beleteszi az my, mx változókba

        mvprintw ( y, x, "O" );        // kiírja a formáott outputot a ↔
        curses ablakba

        refresh ();
        usleep ( 100000 );    // ennyi mikrosecig alszik a programunk
        clear();              //letakarítja az ablakot

        x = x + xnov;          //minden ciklusban hozzáad 1-et "xnov, ynov" ↔
        hozzáad a kurzor koordinátaíhoz
        y = y + ynov;

        if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
            xnov = xnov * -1;
        }
        if ( x<=0 ) { // elerte-e a bal oldalt?
            xnov = xnov * -1;
        }
        if ( y<=0 ) { // elerte-e a tetejet?
            ynov = ynov * -1;
        }
    }
}
```

```
    }
    if ( y>=my-1 ) { // elerte-e a aljat?
        ynov = ynov * -1;
    }

}

return 0;

}
```

Labdanoif

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

#define SZEL 78          //definialjuk SZEL, MAG konstansokat
#define MAG 22

int putX(int x,int y)    //putX függvény
{
    int i;

    for(i=0;i<x;i++)
        printf("\n");    //sortörés

    for(i=0;i<y;i++)
        printf(" ");      //üres karakter

    printf("X\n");        //X majd sortörés

    return 0;
}

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

    while(1)             //végtelen ciklus
    {
        system("clear"); //ablak letakarítása
        putX(abs(MAG-(x++%(MAG*2))),abs(SZEL-(y++%(SZEL*2)))); //abs = ↔
        //abszolútérték Meghívjuk a putX függvényt
        usleep(50000);   //alvás
    }

    return 0;
}
```

```
}
```

Mindkét esetben a kurzort pattogtattunk. Az első esetben egyszerű if-es feltétellel amely azt nézte elérte-e a határokat a kurzorunk. Második esetben már if-es feltétel nélkül pattogtattuk a kurzort függvényt használva.

## 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:

Szohossz

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

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

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

    printf ("Calibrating delay loop..");      //printf függvény kiírás
    fflush (stdout);      //az OS bufferelése miatt kell, így fixen ←
        megjelenik a konzolon a printfünk

    while ((loops_per_sec <= 1))
    {
        ticks = clock ();      //visszadja a program kezdete óta eltelt ←
            tick -ek számát
        delay (loops_per_sec);      //delay függvény meghívása
        ticks = clock () - ticks;

        printf ("%llu %llu\n", ticks, loops_per_sec);

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

            printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
```

```
(loops_per_sec / 5000) % 100);

return 0;
}
}

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 2.6. Helló, Google!

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

Pagerank

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

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

double tavolsag(double pagerank[], double pagerank_temp[], int db)    // ←
    tavolsag függvényünk
{
    double tav = 0.0;
    int i;
    for (i=0; i<db; i++)
    {
        if ((pagerank[i] - pagerank_temp[i])<0)
        {
            tav += (-1*(pagerank[i] - pagerank_temp[i]));
        }
        else
        {
            tav += (pagerank[i] - pagerank_temp[i]);
        }
    }
}
```

```
}
return tav;          //visszaadja a távolságot
}

int main(void)
{
double L[4][4] = {          //4x4 -es mátrix feltöltése
{0.0, 0.0, 1.0 / 3.0, 0.0},
{1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
{0.0, 1.0 / 2.0, 0.0, 0.0},
{0.0, 0.0, 1.0 / 3.0, 0.0}
};

double PR[4] = {0.0, 0.0, 0.0, 0.0};          /x4 -es tömb feltöltése, ez ←
        fogja a végeredményt tárolni
double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};    //Az ←
        oldalak jósága számositva

long int i,j,h;    //definiálunk
i=0; j=0; h=5;    //értéket adunk

for (;;)          //végtelen for ciklus
{
for(i=0;i<4;i++)
PR[i] = PRv[i];    //PR[i] feltöltése PRv[i] értékeivel
for (i=0;i<4;i++)
{
double temp=0;
for (j=0;j<4;j++)
temp+=L[i][j]*PR[j];    //temp=L[i][j]*PR[j]
PRv[i]=temp;
}

if ( tavolsag(PR,PRv, 4) < 0.00001)    //if feltétele tavolsag függvény
break;    //ha feltétel teljesül kilé a ciklusból
}
kiir (PR,4);    //meghívjuk a kiir függvényt
return 0;
}
```

Megoldás videó:

Megoldás forrása:

A Google ezzel az algoritmussal futott be a piacra. Ez az algoritmus kiszámolja az oldalak jóságát, ez alapján helyezi sorrendbe őket. Olyan algoritmus mely a hiperlinkekkel összekötött oldalakhoz hozzárendel egy számot, ezt a számot befolyásolja, hogy hány db oldal mutat az adott oldalra és figyelembe veszi az adott oldalra mutató oldalak jóságát is.

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

A Brun-tétel kimondja hogy az egymást hogy az ikerprímszámok reciprokaiból képzett sor összege véges vagy végtelen sor konvergencia ami azt jelenti hogy ezek a törtek összeadva egy határt adnak amivel pontosan vagy azt át nem lépve növekednek Ez a szám a Brun-konstans

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

A Monty-Hall probléma lényege az hogy van 3 ajtónk. Ezek közül csak egy mögött van nyeremény a többi kettő mögött pedig semmi sincs. Ha választottunk egy ajtót a maradék két ajtó közül azt nyitják ki ami mögött nincs nyeremény. Ekkor jön a döntés meg éri e másik ajtót választani ? A számítógépes szimulációk alapján igenis megéri változtatnunk a döntésünkön.

## 3. fejezet

# Helló, Chomsky!

### 3.1. Decimálisból unárisba átváltó Turing gép

Unáris azaz egyes számrendszer minden számot megfelelő db szimbólummal ábrázol. Ilyen például amikor vonalakkal írjuk fel a számokat egy vonal egyet ér. Az alább látható programon látható, hogy is működik ez a gyakorlatban.

```
include <iostream>
using namespace std;

int main ()
{
    int z;
    cout << "Adj meg egy számot\n";
    cin >> z;
    cout << "Unáris alakja: ";
    for (int i = 0; i < z; i++)
    {
        cout << "|";
    }
}
```

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

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

Változók(N): A, B, C Konstansok(T): a, b, c Kezdő elem(S): S Szabályok(H):  $S \rightarrow AaBcC$ ;  $AaB \rightarrow aa$ ;  $cC \rightarrow bbC$ ;  $C \rightarrow cc$ ;  $Aa \rightarrow aaB$ ;  $Ac \rightarrow bc$ ;  $BB \rightarrow abbA$  S  $(S \rightarrow AaBcC)$  S  $(S \rightarrow AaBcC)$  AaBcC (AaB  $\rightarrow aa$ ) AaBcC (Aa  $\rightarrow aaB$ ) aacC (cC  $\rightarrow bbC$ ) aaBBcC (BB  $\rightarrow abbA$ ) aabbC (C  $\rightarrow cc$ ) aaabbAcC(Ac  $\rightarrow bc$ ) aabbcc aaabbbcC(C  $\rightarrow cc$ ) aaabbbccc

Változók(N): X, Y, Z Konstansok(T): a, b, c Kezdő elem(S): S Szabályok(H):  $S \rightarrow XabYcZ$ ;  $Xab \rightarrow aaX$ ;  $XY \rightarrow bbY$ ;  $YcZ \rightarrow cc$ ;  $Yc \rightarrow bX$ ;  $XZ \rightarrow bYcZ$ ;  $Xa \rightarrow aaa$  S  $(S \rightarrow XabYcZ)$  S  $(S \rightarrow XabYcZ)$  XabYcZ (Xab



->aaX) XabYcZ (Yc-->bX) aaXYcZ (XY-->bbY) XabbXZ (XZ-->bYcZc) aabbYcZ (YcZ-->cc) XabbbYcZc (YcZ-->cc) aabbcc Xabbbccc (Xa-->aaa) aaabbbccc

### 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.

A BNF(Backus-Naur-forma) formális nyelvek leírására használatos eszköz. Természetes nyelvek és programozási nyelvek nyelvtanát adják meg például a segítségével.

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

Ez a programrészlet c89-al hibát ír ki a ciklusfejben deklarált változó miatt, míg c99-et alkalmazva probléma nélkül lefordul. //gcc else.c -o else -std=c89

### 3.4. Saját lexikális elemző

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

Itt a lexer programot használjuk, ugyanis nekünk csak egy .l kiterjesztésű fájlt kell írni ami segítségével a lexer program készíti a c kódunkat. % jelek közzé tesszük azokat a programrészeket amelyeket ténylegesen látni szeretnénk a generált programban.

```
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit    [0-9]
%%
{digit}* (\. {digit}+)?    {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

## 3.5. l33t.l

Lexelj össze egy l33t ciphert!

```
/*
Forditas:
$ lex -o l337d1c7.c l337d1c7.l

Futtatas:
$ gcc l337d1c7.c -o l337d1c7 -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', {"1", "1", "|", "|_" }},
{'m', {"m", "44", "(V)", "\\|/" }},
{'n', {"n", "\\|", "/\\/", "/V" }},
{'o', {"0", "0", "()", "[]" }},
{'p', {"p", "/o", "|D", "|o" }},
{'q', {"q", "9", "O_", "(,)" }},
{'r', {"r", "12", "12", "|2" }},
{'s', {"s", "5", "$", "$" }},
{'t', {"t", "7", "7", "'|'" }},
{'u', {"u", "|_|", "(_)", "[_]" }},
{'v', {"v", "\\/", "\\/", "\\/" }},
{'w', {"w", "VV", "\\//\\/", "(/\\)" }},
{'x', {"x", "%", ")(", ")( " }},
{'y', {"y", "", "", "" }},
{'z', {"z", "2", "7_", ">_" }},
```

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

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

```
%}
```

```
%%
```

```
. {
```

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

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

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

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

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

        found = 1;
        break;
    }

}

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

}

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

A program lényege az, hogy a bemenetre kapott szöveg karaktereit egy ahhoz hasonló karakterre, jelölésre cserélje az előre megadott tömb alapján.

### 3.6. A források olvasása

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

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

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



#### Bugok

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

i.

```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezelő);
//Ha a SIGINT jel kezelését nem lehetett figyelmen kívül hagyni, akkor ←
    a jel kezelését a jelkezelő végezze.
```

ii.

```
for(i=0; i<5; ++i)
//For ciklus 5 alkalommal fut le
```

iii.

```
for(i=0; i<5; i++)
//Ugyanaz mint előbb nem lényeg hogy növeljük az i-t
```

iv.

```
for(i=0; i<5; tomb[i] = i++)
//A tomb elemeinek az i index értékét adja meg a második elemtől ←
kezdve az 5.-ig.
```

v.

```
for(i=0; i<n && (*d++ = *s++); ++i)
//Ez azért lesz bugos mivel az operandus jobb oldalán értékadás áll.
```

vi.

```
printf("%d %d", f(a, ++a), f(++a, a));
//Kiírja a két függvény által kapott számot.
```

vii.

```
printf("%d %d", f(a), a);
//Előzőhöz hasonlóan kiírja az a-t és az f által kapott értéket ←
amennyiben az szám.
```

viii.

```
printf("%d %d", f(&a), a);
//Az mint az előbb csupán a függvénynek az a címét adjuk át
```

### 3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

$$\$(\backslash\text{forall } x \backslash\text{exists } y ((x < y) \backslash\text{wedge}(y \text{ \textit{prím}})))\$$$

$$\$(\backslash\text{forall } x \backslash\text{exists } y ((x < y) \backslash\text{wedge}(y \text{ \textit{prím}})) \backslash\text{wedge}(Ssy \text{ \textit{prím}})) \leftrightarrow$$

$$)\$$$

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

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/MatLog\\_LaTeX](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX)

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

Sorrendben: - Végtelensok prímszám van. - véges sok prímszám van - véges sok prímszám van

### 3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a;
```

- egészre mutató mutató

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

- egész referenciája

```
int &r = a;
```

- egészek tömbje

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

- egészek tömbjének referenciája (nem az első elemé)

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

- egészre mutató mutatók tömbje

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

- egészre mutató mutatót visszaadó függvény

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

- egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

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

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

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;  
//int típusú változó
```

- ```
int *b = &a;  
//int a-ra mutato pointer b
```
- ```
int &r = a;  
//int a referenciája
```
- ```
int c[5];  
//5 elemű int tömb
```
- ```
int (&tr)[5] = c;  
//előbb deklarált tömb referenciája (nem az első elemé)
```
- ```
int *d[5];  
//mutató tömb, ami 5 db egészre mutató pointert tartalmaz
```
- ```
int *h ();  
//függvény ami egészre mutató pointert ad vissza
```
- ```
int *(*l) ();  
//egészre mutató mutatót (*l) visszaadó függvényre mutató mutató *(*l)
```
- ```
int (*v (int c)) (int a, int b)  
//egészre visszaadó és két egészre kapó (a és b) függvényre mutató ←  
mutatót (*v) visszaadó, egészre kapó függvény
```
- ```
int (*(z) (int)) (int, int);  
//függvénymutató (*(z) egy egészre visszaadó (int) és két egészre kapó ←  
függvényre mutató mutatót (*z) visszaadó, egészre kapó függvényre
```

## 4. fejezet

# Helló, Caesar!

### 4.1. int \*\*\* háromszögmátrix

Haromszog

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

int
main ()
{
    int nr = 3;
    double **tm;

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

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

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

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

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

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



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

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

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

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

    free (tm);

    return 0;
}
```

A fenti kód háromszög mátrixot hoz létre. 2 tulajdonsága van főátlója felett csak nullák szerepelnek és négyzetes. Ezt a mátrixot egy egy egészre mutató mutató mutatójával fogjuk létrehozni. A malloc függvény legoglalja a dinamikus memóriában a paraméterként kapott értéket, majd a free függvénnyel fel is tudjuk szabadítani azt.

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

```
EXOR titkosító
$ more e.c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#define MAX_KULCS 100
#define BUFFER_MERET 256
```

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

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

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

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

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

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

        }

        write (1, buffer, olvasott_bajtok);

    }
}
$ gcc e.c -o e -std=c99
$ ./e 56789012 <tiszta.txt >titkos.szoveg
```

Megoldás videó:

Megoldás forrása:

Az EXOR titkosító lényegében a logikai vagyra, azaz a XOR műveletre utal, mely bitenként összehasonlítja mindkét operandust, és mindig 1-et ad vissza, kivéve, amikor az összehasonlított 2 Bit megegyezik, mert akkor nullát. Tehát két operandusra van szükségünk, ez jelen esetben a titkosítandó bemenet, és a titkosításhoz használt kulcs. KULCS,BUFFER méretének maximumát konstansban tároljuk. A mainnek argumentumként adjuk át a kulcsot. "strlen" segítségével megkapjuk a konzolon beadott kulcsunk hosszát, majd a "strncpy" -val átmásoljuk az argv[1] -ben tárolt kulcsot karakterenként átmásoljuk a kulcs tömbbe. A while ciklusban a standard inputról a bufferba olvassuk a biteket amíg el nem érjük a buffer maximumát majd a kulccsal exorozunk. A kapott titkosított szöveget standard outputra írja.

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

```
$ more Titkosito.java
public class Titkosito {

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

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

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

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

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

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

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

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}

public class Titkosito {

    public Titkosito(String kulcsSzöveg,
        java.io.InputStream bejövőCsatorna,
        java.io.OutputStream kimenőCsatorna)
```

```
        throws java.io.IOException {

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

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

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

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

        }

        kimenőCsatorna.write(buffer, 0, olvasottBájtok);

    }

}

public static void main(String[] args) {

    try {

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

    } catch(java.io.IOException e) {

        e.printStackTrace();

    }

}

}
```

Megoldás videó:

Megoldás forrása:

Az előbb leírt C programmal megegyező a funkciója a különbség a nyelvben rejlik. A JAVA objektum orientált programozási nyelv. A szöveget amit titkosít a konzolról kapja.

## 4.4. C EXOR törő

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

```
$ more t.c
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE

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

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

    return (double) titkos_meret / sz;
}

int
tisztalehet (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");
}

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

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

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

    return tiszta_lehet (titkos, titkos_meret);
}

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

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

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

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

                                        if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
                                            printf
                                                ("Kulcs: [%c%c%c%c%c%c%c%c]\\nTiszta szoveg: [%s]\\n",
```

```
        ii, ji, ki, li, mi, ni, oi, pi, titkos);

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

    return 0;
}
```

Megoldás videó:

Megoldás forrása:

Ebben a feladatban az előző feladatokban látott titkosított szövegek feltörésére írunk programot. Konstansként meghatározzuk a kulcs méretét így ennél nagyobb kulccsal titkosított fájlokat nem tudunk feltörni vele. A program vizsgálja hogy az átlagos szóhossz meg van e, illetve megbézi hogy tartalmazza e a gyakori magyar szavakat. Az exor függvény ugyanazt csinálja mint titkosításnál, mivel ha valamit kétszer EXOR-ozunk, akkor visszakapjuk az eredeti szöveget. A kulcs keresése közben az összes lehetőséget átnézi amíg meg nem találja a megfelelő kulcsot.

## 4.5. Neurális OR, AND és EXOR kapu

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

library(neuralnet)

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

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

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

plot(nn.or)

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

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

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

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

plot(nn.orand)

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

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])
```

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)

Ebben a feladatban neuronokat használtunk. A neuronokat felhasználva neurális hálót hozunk létre ami képes tanulni. A lényeg, hogy a neuron akkor fog tüzelni, ha a bemenetek súlyozott összege meghaladnak egy küszöböt. Az aktivációs függvény adja meg a kimenet értékét. Az exor-nál kissé bonyolultabb a dolog. Létre kell hozni rejtett neuronokat, melyek segítenek a tanulásban.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

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

Ez az algoritmus megtanítja a számítógépnek a bináris osztályozást. Itt is neuronokat használunk, ami egy bemenetet kap majd egy adott értéket elérve jelez. A perceptronra tekinthetünk úgy mint egy adatfelismerő "gépre". Ennek a "gépnek" a feladata, hogy véges számú kísérletből megtanulja osztályozni az egyesekből és nullákból álló bemeneti mintázatokat. Ezek után a bemenetek súlyozott összegzését elvégzi, amelyet nem lineáris leképezés követ.

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

A Mandelbrot halmaz a komplex számokat tartalmazó halmaz. A Mandelbrot halmaz a fraktálok közé tartozik. A fraktálok végtelenül komplex alakzatok. Két fő tulajdonságuk, hogy a szélei szakadozottak, nem egyenletesek, a mások pedig az, hogy nagyon hasonlítanak egymásra. A programmal ezt az ábrát fogjuk elkészíteni, png++ headerre szükségünk lesz `//sudo apt-get install libpng++ -dev //`. A program létrehoz egy üres png-t, melybe elkészítjük a Mandelbrot halmazt. A kép pontjain végigmenve ha elemei a halmaznak, akkor az adott pixelt megszinezük.

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

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

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

    double a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = 600, magassag = 600, iteraciosHatar = 1000;
    </programlisting>
    Első lépésként megadjuk a függvény értékkészletét és
    értelmezéstartományát. Majd meghatározzuk a létrehozandó
    kép méretét, és az iterációs határt.
    <programlisting language="c++"><![CDATA[
        png::image <png::rgb_pixel> kep (szelesseg, magassag);
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double reC, imC, reZ, imZ, ujureZ, ujimZ;

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

```
for (int k=0; k<szelesseg; ++k) {
    reC = a+k*dx;
    imC = d-j*dy;
    reZ = 0;
    imZ = 0;
    iteracio = 0;
    while (reZ*reZ + imZ*imZ < 4 && iteracio < iteraciosHatar) {
        // z_{n+1} = z_n * z_n + c
        ujureZ = reZ*reZ - imZ*imZ + reC;
        ujimZ = 2*reZ*imZ + imC;
        reZ = ujureZ;
        imZ = ujimZ;

        ++iteracio;
    }

    kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,
                                         255-iteracio%256, 255-iteracio%256));
}
std::cout << "." << std::flush;

    kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,
                                         255-iteracio%256, 255-iteracio%256));

    kep.write (argv[1])
}
```

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

Az előző feladathoz képest annyi a különbség, hogy míg az előző feladatban a külön vátozó volt a komplex szám kezelésére valós és képzetes rész, itt a komplex osztály segítségével valósítjuk meg ezt. Itt is két egybeágyazott ciklussal járjuk át a pixeleket.

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
```

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

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

## 5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbgRzY76E>

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

Ebben a feladatban a Julia halmazokkal fogunk foglalkozni. Minden Julia halmaz eleme a Mandelbrot halmaznak. A Julia halmaz egyik változója konstans. A program egyezik a Mandelbrot programmal csupán itt a cc konstans értékét és a küszöbszámot a usertők kérjük be.

A program eleje teljesen megegyezik a Mandelbrot halmazos programunkkal, azzal a kivétellel, hogy most a felhasználótól kérjük be a cc konstans értékét, és a küszöbszámot. Ezek az adatok megtalálhatóak a cikkben, minden biomorfhoz külön-külön.

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
```

```
double ymax = 1.3;
double reC = .285, imC = 0;
double R = 10.0;

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

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

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

    std::complex<double> cc ( reC, imC );
1  for x = xmin to xmax by s do
2      for y = ymin to ymax by s do
3          z = x + yi
4          ic = 0
5          for i = 1 to K do
6              z = f(z) + c
7              if |z| > R then
8                  ic = i
9                  break
10         PrintDotAt(x, y) with color ic

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

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

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

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

```
        z_n = std::pow(z_n, z_n) + std::pow(z_n,6) + cc;
        //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
        if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
        {
            iteracio = i;
            break;
        }
    }

    kep.set_pixel ( x, y,
                    png::rgb_pixel ( (iteracio*60)%255, (iteracio ←
                                     *100)%255, (iteracio*40)%255 ));
}

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

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

A Mandelbrot halmaz tovább tuningolása, immár NVIDIA CUDA technológiát felhasználva többszörösre tudjuk növelni a program sebességét, azaz a kép generálását. 600x600-as gridet hozunk létre melyekhez egyenként tartozik egy szál, azaz párhuzamosítjuk a programunkat. A CUDA használatához nvidia GPU-ra van szükség, és telepíteni kell a nvidia-cuda-toolkit-et.

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

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

#define MERET 600
#define ITER_HAT 32000

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

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

    // a számítás
    float dx = (b - a) / szelesseg;
```

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

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

    ++iteracio;
}
return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{

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

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

}
*/

__global__ void
mandelkernel (int *kepadat)
```

```
{

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

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

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

}

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

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

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

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

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

}

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

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

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

int kepadat[MERET][MERET];
```



```
    cudamandel (kepadat);

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

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

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

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

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

}

}
```

## 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!

Itt is Mandelbrot-halmazt kell készíteni, most viszont képesnek kell lennünk belenagyítani, és azokat külön ábrán részletesen meg lehessen vizsgálni. Szükségünk lesz a libqt könyvtárra melyet a `sudo apt-get install libqt4-dev` paranccsal telepíthetünk. A 4 szükséges fájlunk egy mappában kell lennie, ezután futtatjuk a `qmake -project` parancsot. Ez létre hoz egy .pro kiterjesztésű fájlt ahova be kell írni a QT += widgets sort Ezután futtatni kell a `qmake *.pro` paranccsal Ezután lesz a mappában egy Makefile, ezt kell majd használni. Ki adjuk a `make` parancsot, mely létrehoz egy bináris fájlt. Ezt pedig a szokásos módon futtatjuk. Ahhoz, hogy részletesebb képet kapj a ránagyított területről, az "n" billentyűt kell lenyomnod, mely kiszámolja a z-eket a megadott területen.

```
// frakablak.cpp
```

```
//  
// Mandelbrot halmaz nagyító  
#include "frakablak.h"  
FrakAblak::FrakAblak(double a, double b, double c, double d,  
                    int szelesseg, int iteraciosHatar, QWidget *parent)  
    : QMainWindow(parent)  
{  
    setWindowTitle("Mandelbrot halmaz");  
    szamitasFut = true;  
    x = y = mx = my = 0;  
    this->a = a;  
    this->b = b;  
    this->c = c;  
    this->d = d;  
    this->szelesseg = szelesseg;  
    this->iteraciosHatar = iteraciosHatar;  
    magassag = (int)(szelesseg * ((d-c)/(b-a)));  
    setFixedSize(QSize(szelesseg, magassag));  
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);  
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←  
        iteraciosHatar, this);  
    mandelbrot->start();  
}  
FrakAblak::~FrakAblak()  
{  
    delete fraktal;  
    delete mandelbrot;  
}  
void FrakAblak::paintEvent(QPaintEvent*) {  
    QPainter qpainter(this);  
    qpainter.drawImage(0, 0, *fraktal);  
    if(!szamitasFut) {  
        qpainter.setPen(QPen(Qt::white, 1));  
        qpainter.drawRect(x, y, mx, my);  
    }  
    qpainter.end();  
}  
void FrakAblak::mousePressEvent(QMouseEvent* event) {  
    // A nagyítandó kijelölt területet bal felső sarka:  
    x = event->x();  
    y = event->y();  
    mx = 0;  
    my = 0;  
    update();  
}  
void FrakAblak::mouseMoveEvent(QMouseEvent* event) {  
    // A nagyítandó kijelölt terület szélessége és magassága:  
    mx = event->x() - x;  
    my = mx; // négyzet alakú  
    update();  
}
```

```
}  
void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {  
    if(szamitasFut)  
        return;  
    szamitasFut = true;  
    double dx = (b-a)/szelesseg;  
    double dy = (d-c)/magassag;  
    double a = this->a+x*dx;  
    double b = this->a+x*dx+mx*dx;  
    double c = this->d-y*dy-my*dy;  
    double d = this->d-y*dy;  
    this->a = a;  
    this->b = b;  
    this->c = c;  
    this->d = d;  
    delete mandelbrot;  
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←  
        iteraciosHatar, this);  
    mandelbrot->start();  
    update();  
}  
void FrakAblak::keyPressEvent(QKeyEvent *event)  
{  
    if(szamitasFut)  
        return;  
    if (event->key() == Qt::Key_N)  
        iteraciosHatar *= 2;  
    szamitasFut = true;  
    delete mandelbrot;  
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←  
        iteraciosHatar, this);  
    mandelbrot->start();  
}  
void FrakAblak::vissza(int magassag, int *sor, int meret)  
{  
    for(int i=0; i<meret; ++i) {  
        QRgb szin = qRgb(0, 255-sor[i], 0);  
        fraktal->setPixel(i, magassag, szin);  
    }  
    update();  
}  
void FrakAblak::vissza(void)  
{  
    szamitasFut = false;  
    x = y = mx = my = 0;  
}
```

## 5.6. Mandelbrot nagyító és utazó Java nyelven

```
[
    /*
    * MandelbrotHalmazNagyító.java
    *
    * DIGIT 2005, Javat tanítok
    * Bátfai Norbert, nbatfai@inf.unideb.hu
    *
    */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /** A nagyítandó kijelölt területet bal felső sarka. */
    private int x, y;
    /** A nagyítandó kijelölt terület szélessége és magassága. */
    private int mx, my;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nygítani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     *
     * @param a a [a,b]x[c,d] tartomány a koordinátája.
     * @param b a [a,b]x[c,d] tartomány b koordinátája.
     * @param c a [a,b]x[c,d] tartomány c koordinátája.
     * @param d a [a,b]x[c,d] tartomány d koordinátája.
     * @param szélesség a halmazt tartalmazó tömb szélessége.
     * @param iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
        int szélesség, int iterációsHatár) {
        // Az űs osztály konstruktorának hívása
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        // Egér kattintó események feldolgozása:
        addMouseListener(new java.awt.event.MouseAdapter() {
            // Egér kattintással jelöljük ki a nagyítandó területet
            // bal felső sarkát:
            public void mousePressed(java.awt.event.MouseEvent m) {
                // A nagyítandó kijelölt területet bal felső sarka:
                x = m.getX();
                y = m.getY();
                mx = 0;
                my = 0;
                repaint();
            }
        });
    }
}
```

```
    }  
    // Vonszolva kijelölünk egy területet...  
    // Ha felengedjük, akkor a kijelölt terület  
    // újraszámítása indul:  
    public void mouseReleased(java.awt.event.MouseEvent m) {  
        double dx = (MandelbrotHalmazNagyító.this.b  
            - MandelbrotHalmazNagyító.this.a)  
            /MandelbrotHalmazNagyító.this.szélesség;  
        double dy = (MandelbrotHalmazNagyító.this.d  
            - MandelbrotHalmazNagyító.this.c)  
            /MandelbrotHalmazNagyító.this.magasság;  
        // Az új Mandelbrot nagyító objektum elkészítése:  
        new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+ ←  
            x*dx,  
            MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,  
            MandelbrotHalmazNagyító.this.d-y*dy-my*dy,  
            MandelbrotHalmazNagyító.this.d-y*dy,  
            600,  
            MandelbrotHalmazNagyító.this.iterációsHatár);  
    }  
});  
// Egér mozgás események feldolgozása:  
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {  
    // Vonszolással jelöljük ki a négyzetet:  
    public void mouseDragged(java.awt.event.MouseEvent m) {  
        // A nagyítandó kijelölt terület szélessége és magassága:  
        mx = m.getX() - x;  
        my = m.getY() - y;  
        repaint();  
    }  
});  
}  
/**  
 * Pillanatfelvételek készítése.  
 */  
public void pillanatfelvétel() {  
    // Az elementendő kép elkészítése:  
    java.awt.image.BufferedImage mentKép =  
        new java.awt.image.BufferedImage(szélesség, magasság,  
            java.awt.image.BufferedImage.TYPE_INT_RGB);  
    java.awt.Graphics g = mentKép.getGraphics();  
    g.drawImage(kép, 0, 0, this);  
    g.setColor(java.awt.Color.BLUE);  
    g.drawString("a=" + a, 10, 15);  
    g.drawString("b=" + b, 10, 30);  
    g.drawString("c=" + c, 10, 45);  
    g.drawString("d=" + d, 10, 60);  
    g.drawString("n=" + iterációsHatár, 10, 75);  
    if(számításFut) {  
        g.setColor(java.awt.Color.RED);
```

```
        g.drawLine(0, sor, getWidth(), sor);
    }
    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
    g.dispose();
    // A pillanatfelvétel képfájl nevének képzése:
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrotHalmazNagyitas_");
    sb.append(++pillanatfelvételSzámláló);
    sb.append("_");
    // A fájl nevébe bele vesszük, hogy melyik tartományban
    // találtuk a halmazt:
    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
    sb.append("_");
    sb.append(d);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(mentKép, "png",
            new java.io.File(sb.toString()));
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
}
/**
 * A nagyítandó kijelölt területet jelző négyzet kirajzolása.
 */
public void paint(java.awt.Graphics g) {
    // A Mandelbrot halmaz kirajzolása
    g.drawImage(kép, 0, 0, this);
    // Ha éppen fut a számítás, akkor egy vörös
    // vonallal jelöljük, hogy melyik sorban tart:
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    // A jelző négyzet kirajzolása:
    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
}
/**
 * Példányosít egy Mandelbrot halmazt nagyító obektumot.
 */
public static void main(String[] args) {
    // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
```

```
// tartományában keressük egy 600x600-as hálóval és az  
// aktuális nagyítási pontossággal:  
new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);  
}  
}
```

Az előbbiekben látott feladat JAVA implementációja. A forrásban található egy kis bug mivel ha nagyítunk azt egy új ablakban nyílik meg, ami attól függ mekkora területet jelöltünk ki nagyításra.

## 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:

```
class PolarGen {  
  
    public:  
  
        PolarGen(); //konstruktor  
  
        ~PolarGen(){} //destruktor  
  
        double kovetkezo(); //random lekérés  
  
    private:  
  
        bool nincsTarolt;  
        double tarolt; //random értéke  
  
};  
PolarGen::PolarGen() { //a konstruktor kifejtése  
    nincsTarolt = false;  
    std::srand (std::time(NULL)); //random inicializálás  
};  
  
double PolarGen::kovetkezo() { //random lekérő függvény kifejtése  
    if (nincsTarolt)  
    {  
        double u1, u2, v1, v2, w;
```



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

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

tarolt = r * v2;
nincsTarolt = !nincsTarolt;

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

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

int main()
{
    PolarGen rnd;

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

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

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
        if(nincsTarolt)
        {
            double u1, u2, v1, v2, w;
```

```
        do{
            u1 = Math.random();
            u2 = Math.random();
            v1 = 2* u1 -1;
            v2 = 2* u2 -1;
            w = v1*v1 + v2*v2;
        } while (w>1);

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

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

Láthatóan a java forrás sokkal letisztultabb könnyebben értelmezhető és jóval rövidebb. A javában az egész forrás a class része, ebben van a main is, de azt nem tekintjük a class részének.

## 6.2. LZW

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

Egy olyan algoritmus melynek az a lényege, hogy a bemeneten olvasott 1 és 0 -ákból egy bináris fát épít. A fát úgy építi fel hogy mindig megnézi van e egyes vagy nullás gyermeke az adott csomopontnak. Ha nincs létrehoz egyet és visszaugrik a gyökérre, ha van akkor a 0-ás vagy 1-es gyerekre lép, addig halad lefele a fában amíg nem talál olyan csomópontot melynek nincs a keresett gyermeke majd létrehozza azt.

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

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

BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

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

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

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

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

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

```

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

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

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

Katt a továbbra a teljes forrásért:

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

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

/* Ághosszak szórásának kiszámítása */
atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
szorasosszeg = 0.0;

rszoras (gyoker);

double szoras = 0.0;
```

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

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

    szabadit (gyoker);
}

// a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
// http://sourceforge.net/projects/javacska/
// az atlag() hivasakor is inicializalni kell oket, a
// a rekurziv bejaras hasznalja
int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

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

        if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

// a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
// http://sourceforge.net/projects/javacska/
// az atlag() hivasakor is inicializalni kell oket, a
// a rekurziv bejaras hasznalja
double szorasosszeg = 0.0, atlag = 0.0;

void
rszoras (BINFA_PTR fa)
{
```

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

    if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
    {
        ++atlagdb;
        szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
    }
}

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

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

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

```
szabadit (elem->bal_nulla);  
free (elem);  
}  
}
```

## 6.3. Fabejárás

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

Előző feladatban látott programot módosítjuk a bejárás sorrendjével. Posztorder bejárásnál elsőnek a baloldali gyereket majd a jobb oldali gyereket azután a gyökéren keresztül kerül feldolgozásra. Inorder bejárásnál amit a programunk alpból használ elsőnek feldolgozzuk a baloldali gyermeket majd a gyökérelemet majd a jobb oldali gyereket. A preorder bejárás során a gyökérelemet dolgozza fel elsőnek, majd a bal azután pedig a jobb oldali gyereket. A program lényegében ugyanaz csak a kiír függvényt kell átírni. Kezdjük a postorder bejárással.

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

### preorder bejárás

```
void  
kiir (BINFA_PTR elem)  
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        if (melyseg > max_melyseg)  
            max_melyseg = melyseg;  
        for (int i = 0; i < melyseg; ++i)
```

```
printf ("---");
printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
    ,
    melyseg-1);
kiir (elem->jobb_egy);
// ez a postorder bejáráshoz képest
// 1-el nagyobb mélység, ezért -1
kiir (elem->bal_nulla);
--melyseg;
}
}
```

Itt a for ciklus került legelőre, tehát a paraméterként átadott elemet dolgozzuk fel, és csak ezután a bal, majd a jobb oldali gyermeket.

## 6.4. Tag a gyökér

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

Az LZW algoritmus átírása c++-ba. Bejönnek az osztályok melyek a c forrásban látott struktúráknak fejlettebb verziója. C++ ban már képes az osztály függvényeket is kezelni.

```
// z3a2.cpp
//
// Együtt támadjuk meg: http://progpater.blog.hu/2011/04/14/ ←
// együtt_tamadjuk_meg
// LZW fa építő 3. C++ átírata a C változatból (+mélység, átlag és szórás)
// Programozó Páternoszter
//
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail ←
// .com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
```



```
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1,    http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2,    csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3,    http://progpater.blog.hu/2011/03/05/ ←
//            labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// 0.0.4,    z.cpp: a C verzióból svn: bevezetes/C/ziv/z.c átírjuk C++-ra
//            http://progpater.blog.hu/2011/03/31/ ←
//            imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol
// 0.0.5,    z2.cpp: az fgv(*mut)-ok helyett fgv(&ref)
// 0.0.6,    z3.cpp: Csomopont beágyazva
//            http://progpater.blog.hu/2011/04/01/ ←
//            imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2
// 0.0.6.1   z3a2.c: LZWBinFa már nem barátja a Csomopont-nak, mert annak ←
//            tagjait nem használja direktben
// 0.0.6.2   Kis kommentezést teszünk bele 1. lépésként (hogy a kicsit ←
//            lemaradt hallgatóknak is
//            könnyebb legyen, jól megtűzdeljük további olvasmányokkal)
//            http://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg
//            (majd a 2. lépésben "beletemesszük a d.c-t", majd s 3. lépésben a ←
//            parancssor sor argok feldolgozását)

#include <iostream> // mert olvassuk a std::cin, írjuk a std::cout ←
//            csatornákat
#include <cmath> // mert vonunk gyököt a szóráshoz: std::sqrt
#include <fstream> // fájlból olvasunk, írunk majd

/* Az LZWBinFa osztályban absztraháljuk az LZW algoritmus bináris fa ←
//            építését. Az osztály
//            definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ←
//            ez lesz a
//            beágyazott Csomopont osztály. Miért ágyazzuk be? Mert külön nem szánunk ←
//            neki szerepet, ezzel
//            is jelezzük, hogy csak a fa részeként számolunk vele.*/

class LZWBinFa
{
public:
```

```
/* Szemben a bináris keresőfánkkal (BinFa osztály)
http://progpater.blog.hu/2011/04/12/ ↵
    imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3
itt (LZWBinFa osztály) a fa gyökere nem pointer, hanem a '/' betűt ↵
    tartalmazó objektum,
lásd majd a védett tagok között lent: Csomopont gyoker;
A fa viszont már pointer, mindig az épülő LZW-fánk azon csomópontjára ↵
    mutat, amit az
input feldolgozása során az LZW algoritmus logikája diktál:
http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
Ez a konstruktor annyit csinál, hogy a fa mutatót ráállítja a gyökre ↵
    . (Mert ugye
labopon, blogon, előadásban tisztáztuk, hogy a tartalmazott tagok, ↵
    most "Csomopont gyoker"
konstruktor elöbb lefut, mint a tagot tartalmazó LZWBinFa osztály ↵
    konstruktor, éppen a
következő, azaz a fa=&gyoker OK.)
*/
LZWBinFa (): fa(&gyoker) {}

/* Tagfüggvényként túlterheljük a << operátort, ezzel a célunk, hogy ↵
    felkeltsük a
hallgató érdeklődését, mert ekkor így nyomhatjuk a fába az inputot: ↵
    binFa << b; ahol a b
egy '0' vagy '1'-es betű.
Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett ↵
    paraméterként"
kapott ) példány, azaz annak a fának amibe éppen be akarjuk nyomni a b ↵
    betűt a tagjai
(pl.: "fa", "gyoker") használhatóak a függvényben.

A függvénybe programoztuk az LZW fa építésének algoritmusát tk.:
http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

a b formális param az a betű, amit éppen be kell nyomni a fába: */
void operator<<(char b)
{
    // Mit kell betenni éppen, '0'-t?
    if (b == '0')
    {
        /* Van '0'-s gyermeke az aktuális csomópontnak?
        megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
        if (!fa->nullasGyermeke ()) // ha nincs, hát akkor csinálunk
        {
            // elkészítjük, azaz példányosítunk a '0' betű akt. ↵
            parammal
            Csomopont *uj = new Csomopont ('0');
            // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
            // jegyezze már be magának, hogy nullás gyereke mostantól ↵
            van
        }
    }
}
```

```
        // küldjük is Neki a gyerek címét:
        fa->ujNullasGyermek (uj);
        // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
        fa = &gyoker;
    }
    else // ha van, arra rálépünk
    {
        // azaz a "fa" pointer már majd a szóban forgó gyermekre ↵
        mutat:
        fa = fa->nullasGyermek ();
    }
}
// Mit kell betenni éppen, vagy '1'-et?
else
{
    if (!fa->egyGyermek ())
    {
        Csomopont *uj = new Csomopont ('1');
        fa->ujEgyGyermek (uj);
        fa = &gyoker;
    }
    else
    {
        fa = fa->egyGyermek ();
    }
}
}
/* A bejárással kapcsolatos függvényeink (túlterhelt kiir-ók, atlag, ↵
    ratlag stb.) rekurzívak,
    tk. a rekurzív fabejárást valósítják meg (lásd a 3. előadás "Fabejárás ↵
    " c. fóliáját és társait)

    (Ha a rekurzív függvénnnyel általában gondod van => K&R könyv megfelel ↵
    ő része: a 3. ea. izometrikus
    részében ezt "letáncoltuk" :) és külön idéztük a K&R álláspontját :)
    */
void kiir (void)
{
    // Sokkal elegánsabb lenne (és más, a bevezetésben nem kibontandó ↵
    reentráns kérdések miatt is, mert
    // ugye ha most két helyről hívják meg az objektum ilyen ↵
    függvényeit, tehát ha kétszer kezd futni az
    // objektum kiir() fgv.-e pl., az komoly hiba, mert elromlana a ↵
    mélység... tehát a mostani megoldásunk
    // nem reentráns) ha nem használnánk a C verzióban globális ↵
    változókat, a C++ változatban példánytagot a
    // mélység kezelésére: http://progpater.blog.hu/2011/03/05/ ↵
    there_is_no_spoon
    melyseg = 0;
    // ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát, ↵
```

```
        akkor a
        // sztenderd out-ra nyomjuk
        kiir (&gyoker, std::cout);
    }
    void szabadit (void)
    {
        szabadit (gyoker.egyGyermek());
        szabadit (gyoker.nullasGyermek());
        // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a ←
        szabad tárban (halmon).
    }

    /* A változatosság kedvéért ezeket az osztálydefiníció (class LZWBinFa ←
    {...};) után definiáljuk,
    hogy kénytelen légy az LZWBinFa és a :: hatókör operátorral minősítve ←
    definiálni :) l. lentebb */
    int getMelyseg (void);
    double getAtlag (void);
    double getSzoras (void);

    /* Vágyunk, hogy a felépített LZW fát ki tudjuk nyomni ilyenformán: std ←
    ::cout << binFa;
    de mivel a << operátor is a sztenderd névtérben van, de a using ←
    namespace std-t elvből
    nem használjuk bevezető kurzusban, így ez a konstrukció csak az ←
    argfüggő névfeloldás miatt
    fordul le (B&L könyv 185. o. teteje) ám itt nem az a lényeg, hanem, ←
    hogy a cout ostream
    osztálybeli, így abban az osztályban kéne módosítani, hogy tudjon ←
    kiírni LZWBinFa osztálybelieket...
    e helyett a globális << operátort terheljük túl, */
    friend std::ostream& operator<< (std::ostream& os, LZWBinFa& bf)
    {
        bf.kiir(os);
        return os;
    }
    void kiir (std::ostream& os)
    {
        melyseg = 0;
        kiir (&gyoker, os);
    }

private:
    class Csomopont
    {
    public:
        /* A paraméter nélküli konstruktor az elepértelmezett '/' "gyökér- ←
        betűvel" hozza
        létre a csomópontot, illet hívunk a fából, aki tagként tartalmazza ←
        a gyökeret.
```

```
Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu" ←
tagba, a két
gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is ←
megteszi. */
Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
~Csomopont () {};
// Aktuális csomópont, mondd meg nékem, ki a bal oldali gyermeked
// (a C verzió logikájával műxik ez is: ha nincs, akkor a null megy ←
vissza)
Csomopont *nullasGyermekek () const {
    return balNulla;
}
// Aktuális csomópont, mondd meg nékem, ki a jobb oldali gyermeked?
Csomopont *egyenesGyermekek () const {
    return jobbEgy;
}
// Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a bal ←
oldali gyerekek!
void ujNullasGyermekek (Csomopont * gy) {
    balNulla = gy;
}
// Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a jobb ←
oldali gyerekek!
void ujEgyenesGyermekek (Csomopont * gy) {
    jobbEgy = gy;
}
// Aktuális csomópont: Te milyen betűt hordozol?
// (a const kulcsszóval jelezzük, hogy nem bántjuk a példányt)
char getBetu() const {
    return betu;
}

private:
// friend class LZWBinFa; /* mert ebben a változatban az LZWBinFa ←
metódusai nem közvetlenül
// a Csomopont tagjaival dolgoznak, hanem beállító/lekérdező ←
üzenetekkel érik el azokat */

// Milyen betűt hordoz a csomópont
char betu;
// Melyik másik csomópont a bal oldali gyermeke? (a C változatból " ←
örökölt" logika:
// ha nincs ilyen csermeke, akkor balNulla == null) igaz
Csomopont *balNulla;
Csomopont *jobbEgy;
// nem másolható a csomópont (ökörszabály: ha van valamely a ←
szabad tárban,
// letiltjuk a másoló konstruktort, meg a másoló értékadást)
Csomopont (const Csomopont &);
Csomopont & operator=(const Csomopont &);
```

```
};

/* Mindig a fa "LZW algoritmus logikája szerinti aktuális" ←
   csomópontjára mutat */
Csomopont *fa;
// technikai
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
// szokásosan: nocopyable
LZWBinFa (const LZWBinFa &);
LZWBinFa & operator=(const LZWBinFa &);

/* Kiírja a csomópontot az os csatornára. A rekurzió kapcsán lásd a ←
   korábbi K&R-es utalást...*/
void kiir (Csomopont* elem, std::ostream& os)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    // leállítása
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermeke(), os);
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu() << "(" << melyseg - 1 << ")" << std::endl ←
        ;
        kiir (elem->nullasGyermeke(), os);
        --melyseg;
    }
}

void szabadit (Csomopont * elem)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    // leállítása
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermeke());
        szabadit (elem->nullasGyermeke());
        // ha a csomópont mindkét gyermekét felszabadítottuk
        // azután szabadítjuk magát a csomópontot:
        delete elem;
    }
}

protected: // ha esetleg egyszer majd kiterjesztjük az osztályt, mert
// akarunk benne valami újdonságot csinálni, vagy meglévő tevékenységet ←
// máshogy... stb.
// akkor ezek látszanak majd a gyerek osztályban is
```

```
/* A fában tagként benne van egy csomópont, ez erősen ki van tüntetve, ↵
   Ő a gyökér: */
Csomopont gyoker;
int maxMelyseg;
double atlag, szoras;

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

};

// Néhány függvényt az osztálydefiníció után definiálunk, hogy lássunk ↵
// ilyet is ... :)
// Nem erőltetjük viszont a külön fájlba szedést, mert a ↵
// sablonosztályosított tovább
// fejlesztésben az linkelési gondot okozna, de ez a téma már kivezet a ↵
// laborteljesítés
// szükséges feladatából: http://progpater.blog.hu/2011/04/12/ ↵
// imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3

// Egyébként a melyseg, atlag és szoras fgv.-ek a kiir fgv.-el teljesen egy ↵
// kaptafa.

int LZWBinFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (&gyoker);
    return maxMelyseg-1;
}
double LZWBinFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double)atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (&gyoker);

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

```
        return szoras;
    }
    void LZWBinFa::rmelyseg (Csomopont* elem)
    {
        if (elem != NULL)
        {
            ++melyseg;
            if (melyseg > maxMelyseg)
                maxMelyseg = melyseg;
            rmelyseg (elem->egyenesGyermek());
            // ez a postorder bejáráshoz képest
            // 1-el nagyobb mélység, ezért -1
            rmelyseg (elem->nullasGyermek());
            --melyseg;
        }
    }
    void
    LZWBinFa::ratlag (Csomopont* elem)
    {
        if (elem != NULL)
        {
            ++melyseg;
            ratlag (elem->egyenesGyermek());
            ratlag (elem->nullasGyermek());
            --melyseg;
            if (elem->egyenesGyermek() == NULL && elem->nullasGyermek() == NULL)
            {
                ++atlagdb;
                atlagosszeg += melyseg;
            }
        }
    }
    void
    LZWBinFa::rszoras (Csomopont* elem)
    {
        if (elem != NULL)
        {
            ++melyseg;
            rszoras (elem->egyenesGyermek());
            rszoras (elem->nullasGyermek());
            --melyseg;
            if (elem->egyenesGyermek() == NULL && elem->nullasGyermek() == NULL)
            {
                ++atlagdb;
                szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
            }
        }
    }
}
```



```
// teszt pl.: http://progpater.blog.hu/2011/03/05/ ↵
    labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// [norbi@sgu ~]$ echo "01111001001001000111" | ./z3a2
// -----1(3)
// -----1(2)
// -----1(1)
// -----0(2)
// -----0(3)
// -----0(4)
// ---/(0)
// -----1(2)
// -----0(1)
// -----0(2)
// depth = 4
// mean = 2.75
// var = 0.957427
// a laborvédeéshez majd ezt a tesztelést használjuk:
// http://

/* Ez volt eddig a main, de most komplexebb kell, mert explicite bejövő, ↵
    kimenő fájlokkal kell dolgozni
int
main ()
{
    char b;
    LZWBinFa binFa;

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

    //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ↵
        verziókban de, hogy izgalmasabb legyen
    // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

    std::cout << binFa; // ehhez kell a globális operator<< túlterhelése, ↵
        lásd fentebb

    std::cout << "depth = " << binFa.getMelyseg () << std::endl;
    std::cout << "mean = " << binFa.getAtlag () << std::endl;
    std::cout << "var = " << binFa.getSzoras () << std::endl;

    binFa.szabadit ();

    return 0;
}
*/

/* A parancssor arg. kezelést egyszerűen bedolgozzuk a 2. hullám kapcsolódó ↵
```

```
feladatából:
http://progpater.blog.hu/2011/03/12/hey_mikey_he_likes_it_ready_for_more_3
de mivel nekünk sokkal egyszerűbb is elég, alig hagyunk meg belőle valamit ↵
...
*/

void usage(void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    // http://progpater.blog.hu/2011/03/12/ ↵
    // hey_mikey_he_likes_it_ready_for_more_3
    // alapján a parancssor argok ottani elegáns feldolgozásából kb. ennyi ↵
    // marad:
    // "*(++argv)+1"...

    // a kiírás szerint ./lzwtree in_file -o out_file alakra kell mennie, ↵
    // ez 4 db arg:
    if (argc != 4) {
        // ha nem annyit kapott a program, akkor felhomályosítjuk erről a ↵
        // jüzettr:
        usage();
        // és jelezzük az operációs rendszer felé, hogy valami gáz volt...
        return -1;
    }

    // "Megjegyezzük" a bemenő fájl nevét
    char *inFile = ++argv;

    // a -o kapcsoló jön?
    if (*(++argv)+1) != 'o') {
        usage();
        return -2;
    }

    // ha igen, akkor az 5. előadásból kimásoljuk a fájlkezelés C++ ↵
    // változatát:
    std::fstream beFile (inFile, std::ios_base::in);
    std::fstream kiFile (*argv, std::ios_base::out);

    unsigned char b; // ide olvassik majd a bejövő fájl bájtjait
    LZWBinFa binFa; // s nyomjuk majd be az LZW fa objektumunkba

    // a bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen ↵
    // írjuk, hogy meg tudjuk
    // majd nézni... :) 1. az említett 5. ea. C -> C++ gyökettes átírási ↵
```

példáit

```
while (beFile.read ((char *) &b, sizeof (unsigned char))) {
// egyszerűen a korábbi d.c kódját bemásoljuk
// laboron többször lerajzoltuk ezt a bit-tologatást:
// a b-ben lévő bájt bitjeit egyenként megnézzük
    for (int i = 0; i < 8; ++i)
    {
        // maszkolunk
        int egy_e = b & 0x80;
        // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ←
        megmondja melyik:
        if ((egy_e >> 7) == 1)
        // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW fa ←
        objektumunkba
            binFa << '1';
        else
        // különben meg a '0' betűt:
            binFa << '0';
        b <<= 1;
    }

}

//std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ←
// verziókban de, hogy izgalmasabb legyen
// a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

kiFile << binFa; // ehhez kell a globális operator<< túlterhelése, lásd ←
// fentebb
// (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, ←
// mégis megy, hurrá)

kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;

binFa.szabadit ();

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

return 0;
}
```

A fentebbi programrészletben részleteiben el van magyarázva a program funkciója. Röviden a kapott inputon maskolással megkapjuk a nullákat vagy egyeseket melyekből felépítjük a binfánkat.

## 6.5. Mutató a gyökér

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

Az előző feladatban a gyökér tagja volt az osztálynak. Most át kell írni pointerre. Elsőnek át kell írni a gyökert mutatóra // Csomópont \*gyoker //. Így lefuttatva rengeteg hibát fog kidobni, most ezeket kell kijavítanunk. Elsőnek a konstruktorokban kell módosítani. Ezután már csak mindenhol törölni kell a memóriacímre való utalást a gyökér elől.

```
LZWBinFa ()
{
    gyoker = new Csomopont ( '/' );
    fa = gyoker;
}

~LZWBinFa ()
{
    szabadit (gyoker->egyenesGyermekek ());
    szabadit (gyoker->nullasGyermekek ());
    delete(gyoker);
}
```

## 6.6. Mozgató szemantika

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

A feladathoz ahhoz a programhoz nyúlunk vissza, amely esetén a gyökét tag volt. Ezen elvégezve az apróbb módosításokat, már készen is van a mozgatókonstruktor.

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

    gyoker.ujEgyenesGyermekek ( regi.gyoker.egyenesGyermekek() );
    gyoker.ujNullasGyermekek ( regi.gyoker.nullasGyermekek() );

    regi.gyoker.ujEgyenesGyermekek ( nullptr );
    regi.gyoker.ujNullasGyermekek ( nullptr );
}

LZWBinFa& operator = (LZWBinFa && regi)
{
    if (this == &regi)
        return *this;

    gyoker.ujEgyenesGyermekek ( regi.gyoker.egyenesGyermekek() );
    gyoker.ujNullasGyermekek ( regi.gyoker.nullasGyermekek() );

    regi.gyoker.ujEgyenesGyermekek ( nullptr );
}
```

```
    regi.gyoker.ujNullasGyermek ( nullptr );  
  
    return *this;  
}
```

Ehhez meg kell csinálnunk a mozgató konstruktort és a mozgató értékadást. Mind a kettőnek hasonló a szintaktikája, ami annyit tesz, hogy a paraméterként átadott fa gyökerének az elemeit átadjuk az üres fának, majd a mozgatott fa elemeit kinullázuk.

```
LZWBinFa binFa2 = std::move(binFa);  
  
kiFile << binFa2;  
kiFile << "depth = " << binFa2.getMelyseg () << std::endl;  
kiFile << "mean = " << binFa2.getAtlag () << std::endl;  
kiFile << "var = " << binFa2.getSzoras () << std::endl;
```

Végezetül a move függvénnyel átmozgatjuk az egyes elemeket, és végül kiírjuk az új fát. Láthatjuk ha ezután az eredeti binFa-t is ki akarnánk iratni, akkor nem tudnánk, mert azt kinulláztuk.

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyaszimulációk

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

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

Megoldás forrása: <https://github.com/davik0000/kod/tree/master/Hangya>

A program több fájlból tevődik össze, fontos, hogy ezek egy helyen legyenek. Ez a feladat a hangyák viselkedésének lemodellezéséről szól. A természetben jól látható, hogy a hangyák egy egyenest követve mozognak. Az osztályok objektumok tagváltozóit és metódusait tartalmazza pl a hangyának van síkbeli pozíciója, haladási iránya. Az ants vektor fogja a hangyákat tárolni. Az antthread.h header file AntThread osztályában található a párolgás mértéke (evaporation) és a feromonok számát (nbrPheromone) tároló változókat. Láthatjuk a hangyák cellán belüli maximális előfordulásának számát tároló változót is (cellAntMax). Private az új irányt meghatározó függvényünk, illetve a moveAnts függvényünk ami a hangyák mozgásáért felelős. A Qt keretrendszer feltelepítése, illetve a program fordítása után futtatható is.

### 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

A játék John Horton Conway Angol matematikus nevéhez fűződik az életjáték nevű program. A felhasználónak csak annyi beleszólása van hogy meghatározza a kezdő pozíciót illetve alakzatot. A játéktér négyzetláncos felület minden négyzetet 8 másik négyzet vesz körül. Ha egy generációban egy sejtnek kettő vagy három élő szomszédja van, akkor a sejt élni fog a következő generációban is, minden más esetben a sejt kihal. Ha egy üres cellának pontosan három élő sejt van a szomszédjában, akkor ott új sejt születik.

```
/*  
 * Sejtautomata.java  
 */
```

```
* DIGIT 2005, Javat tanítok
* Bátfai Norbert, nbatfai@inf.unideb.hu
*
*/
/**
 * Sejtautomata osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class Sejtautomata extends java.awt.Frame implements Runnable {
    /** Egy sejt lehet élő */
    public static final boolean ÉLŐ = true;
    /** vagy halott */
    public static final boolean HALOTT = false;
    /** Két rácsot használunk majd, az egyik a sejttér állapotát
     * a t_n, a másik a t_n+1 időpillanatban jellemzi. */
    protected boolean [][][] rácsok = new boolean [2][][];
    /** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
     * [2][][]-ból az első dimenziót használni, mert vagy az egyikre
     * állítjuk, vagy a másikra. */
    protected boolean [][] rács;
    /** Megmutatja melyik rács az aktuális: [rácsIndex][][] */
    protected int rácsIndex = 0;
    /** Pixelben egy cella adatai. */
    protected int cellaSzélesség = 20;
    protected int cellaMagasság = 20;
    /** A sejttér nagysága, azaz hányszor hány cella van? */
    protected int szélesség = 20;
    protected int magasság = 10;
    /** A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
     közötti valós idő. */
    protected int várakozás = 1000;
    // Pillanatfelvétel készítéséhez
    private java.awt.Robot robot;
    /** Készítsünk pillanatfelvételt? */
    private boolean pillanatfelvétel = false;
    /** A pillanatfelvételek számozásához. */
    private static int pillanatfelvételSzámláló = 0;
    /**
     * Létrehoz egy <code>Sejtautomata</code> objektumot.
     *
     * @param      szélesség      a sejttér szélessége.
     * @param      magasság      a sejttér szélessége.
     */
    public Sejtautomata(int szélesség, int magasság) {
        this.szélesség = szélesség;
        this.magasság = magasság;
        // A két rács elkészítése
        rácsok[0] = new boolean[magasság][szélesség];
```

```
rácsok[1] = new boolean[magasság][szélesség];
rácsIndex = 0;
rács = rácsok[rácsIndex];
// A kiinduló rács minden cellája HALOTT
for(int i=0; i<rács.length; ++i)
    for(int j=0; j<rács[0].length; ++j)
        rács[i][j] = HALOTT;
// A kiinduló rácsra "élőlényeket" helyezünk
//sikló(rács, 2, 2);
siklóKilövő(rács, 5, 60);
// Az ablak bezárásakor kilépünk a programból.
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent e) {
        setVisible(false);
        System.exit(0);
    }
});
// A billentyűzetről érkező események feldolgozása
addKeyListener(new java.awt.event.KeyAdapter() {
    // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük
    public void keyPressed(java.awt.event.KeyEvent e) {
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
            // Felezzük a cella méreteit:
            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            // Duplázzuk a cella méreteit:
            cellaSzélesség *= 2;
            cellaMagasság *= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel = !pillanatfelvétel;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
            várakozás /= 2;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
            várakozás *= 2;
        repaint();
    }
});
// Egér kattintó események feldolgozása:
addMouseListener(new java.awt.event.MouseAdapter() {
    // Egér kattintással jelöljük ki a nagyítandó területet
    // bal felső sarkát vagy ugyancsak egér kattintással
    // vizsgáljuk egy adott pont iterációit:
    public void mousePressed(java.awt.event.MouseEvent m) {
```



```
        // Az egérmutató pozíciója
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});
// Cellaméretetek kezdetben
cellaSzélesség = 10;
cellaMagasság = 10;
// Pillanatfelvétel készítéséhez:
try {
    robot = new java.awt.Robot(
        java.awt.GraphicsEnvironment.
            getLocalGraphicsEnvironment().
            getDefaultScreenDevice());
} catch (java.awt.AWTException e) {
    e.printStackTrace();
}
// A program ablakának adatai:
setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
        magasság*cellaMagasság);
setVisible(true);
// A sejttér életrekeltése:
new Thread(this).start();
}
/** A sejttér kirajzolása. */
public void paint(java.awt.Graphics g) {
    // Az aktuális
    boolean [][] rács = rácsok[rácsIndex];
    // rácsot rajzoljuk ki:
    for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon
        for(int j=0; j<rács[0].length; ++j) { // s az oszlopok
            // Sejt cella kirajzolása
            if(rács[i][j] == ÉLŐ)
                g.setColor(java.awt.Color.BLACK);
            else
                g.setColor(java.awt.Color.WHITE);
        }
    }
}
```

```
        g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                    cellaSzélesség, cellaMagasság);
        // Rács kirajzolása
        g.setColor(java.awt.Color.LIGHT_GRAY);
        g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                    cellaSzélesség, cellaMagasság);
    }
}

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

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

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

```
        else if(s >= magasság)
            s = 0;

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

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

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

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

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

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

    while(true) {
        try {
```

```
        Thread.sleep(várakozás);
    } catch (InterruptedException e) {}

    időFejlődés();
    repaint();
}

/**
 * A sejttérbe "élőlényeket" helyezünk, ez a "sikló".
 * Adott irányban halad, másolja magát a sejttérben.
 * Az élőlény ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 172. oldal.)
 *
 * @param rács      a sejttér ahová ezt az állatkát helyezzük
 * @param x          a befoglaló téglal bal felső sarkának oszlopa
 * @param y          a befoglaló téglal bal felső sarkának sora
 */
public void sikló(boolean [][] rács, int x, int y) {

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

}

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

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

```
rács[y+ 3][x+ 13] = ÉLŐ;

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

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

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

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

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

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

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

rács[y+ 11][x+ 25] = ÉLŐ;

}

/** Pillanatfelvételek készítése. */
```

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

// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
    paint(g);
}

/**
 * Példányosít egy Conway-féle életjáték szabályos
 * sejtter obektumot.
 */
public static void main(String[] args) {
    // 100 oszlop, 75 sor mérettel:
    new Sejtautomata(100, 75);
}
}
```

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/davik0000/prog1/blob/master/eletjatek>

Ugyanaz az életjáték amit fentebb is láthatunk csupán a c++ implementációja. Itt is ugyanazok a szabályok élnek mint a fentebb látható példában. Itt több header fájlunk van, fontos hogy minden egy helyen legyen.

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/esport-talent-search>

A BrainB Benchmark szoftver célja a játékos képességeinek felmérése olyan pillanatokban amikor a képernyők effektek sokassága egyszerre is megjelenhet. A programban egy karaktert kell figyelniünk akkor jó a

mérés ha sikerül a doboz közepén lévő kék ponton tartani a kurzort. Egyre több doboz kezd el mozogni a képernyőn és ha a játékos 1200ms -en túl tovább elveszti a dobozt ,azaz nincs rajta a kurzora akkor vége a játéknak. Ha a játékos eéveszti a négyzetet a négyzetek lelassulnak. Ezzel a programmal kilehet szűrni a potenciális e-sportolókat.

DRAFT

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.4. Deep dream

Keras



Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 9. fejezet

# Helló, Chaitin!

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

Megoldás videó:

Megoldás forrása:

### 9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

## 9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9.6. Omega

Megoldás videó:

Megoldás forrása:

DRAFT

## 10. fejezet

# Helló, Gutenberg!

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

A számítógépek programozására kialakult nyelvnek 3 fajtáját különböztetjük meg: gépi nyelv, assembly szintű nyelv, magas szintű nyelv. Jelenleg magas szintű nyelvvel foglalkozunk mint amilyen a C is. Minden nyelvnek megvan a saját nyelvi szabályrendszere. Ezek folyamatosan fejlődnek az igényeknek megfelelően. A programban található nyelvi elemek az alábbi részekből állnak össze: -lexikális egységek, -szintaktikai egységek, -utasítások, -programegységek, -fordítási egységek, -program. A kifejezések szintaktikai eszközök. Feladatuk hogy a program egy adott pontján már ismert értékekből új értékeket kapjunk meg. Két összetevője van : -érték, -típus. Egy kifejezés formálisan 3 összetevőből áll: -operandus, -operátor, -kerek zárójelek. Van egyoperandusú művelet illetve kétoperandusú vagy háromoperandusú művelet attól függően hogy hány db operandussal végezzük a műveletet. A kifejezéseknek három alakja lehet : -prefix, -infix, -postfix. Az utasítások segítségével adjuk meg az algoritmusok egyes lépéseit, illetve a fordítóprogramunk ezzel generálja a tárgyprogramot. Két csoport létezik : -deklarációs, -végrehajtható. Értékadó utasítás : Segítségével beállíthatjuk vagy módosíthatjuk egy változónak értékét a program futásának bármelyik pillanatában. Ugró utasítás : GOTO címke. Elágaztató utasítások : if else. Többirányú elágaztató utasítások : case. Ciklusszervező utasítások : for, while ciklus általános felépítése : fej, mag, vég . Az ismétlésre vonatkozó információk a fejben vagy a végben szerepelnek. A mag tartalmazza a végrehajtandó parancsokat.

### 10.2. Kernigan-Ritchie: A C programozási nyelv

Az itt található programrészletek a Kernigan-Ritchie: A C programozási nyelv című könyvből vannak kiemelve. A szabványps könyvtárban találhatunk olyan függvényeket amellyel képesek vagyunk egyszerre egy karaktert írni vagy olvasni. getchar() minden hívásakor beolvas egy karaktert és a karakter lesz a visszatérési értéke. A putchar() fg. valamilyen kimenetre kiír egy karaktert.

```
/* A bemenet átmásolása a kimenetre. 1. változat*/
main()
{
    int c ;
    c = getchar();
    while (c != EOF)
```

```
{ putchar(c);  
  c = getchar();  
}  
}
```

Használat előtt a változóinkat deklarálnunk kell, enélkül hibát fogunk kapni. A deklaráció meghatároz egy típust, amit a változó neve követ. Ha a deklarációnál értéket is adunk meg akkor onnantól kezdve inicializálásról beszélünk. A kifejezések utasítássá válnak abban az esetben ha pontosvessző követi őket. A c nyelvben a pontosvessző utasításlezáró jel azaz terminátor

### 10.3. Benedek-Levendovszky: Szoftverfejlesztés C++ nyelven

A c++ objektumorientált nyelv. Jelenleg is a legmodernebb nyelvek közé tartozik, széles körben elterjedt. Találkozhatunk egy új típussal a bool -al, mely felveheti a true illetve a false értékeket. A c++ nyelvben akkor a legjobb változót deklarálni amikor egyből fel is használjuk. C++ -ban lehetőség nyílik a függvények túlterhelésére. Azonos nevű de különböző argumentumlistájú függvényeket is létre tudunk hozni. A visszatérési értékük nem térhetnek el. Az argumentumlistához alapértelmezett értékeket is rendelhetünk. Megjelenik a referencia, dereferencia. Kialakult az Objektumorientáltság ami hatalmas szabadságot és átláthatóságot ad a programozók kezébe. A valós világ dolgainak tulajdonságai alapján a dolgokat osztályokba tudják zárni. Ezeknek a dolgoknak osztályon belül meg lehet adni a tulajdonságait illetve a funkcióikat. Ezzel megvalósítható az effektív adatrejtés is : private, public, protected. Ahányszor példányosítunk osztályt annyszor foglalunk helyet a tagjainak. Adatrejtésre azért van szükségünk, hogy ne érjék el közvetlenül a tagváltozóinkat az osztályban és azzal illegális értékeket állítsanak be neki. Ahhoz hogy elrejtünk valamit elég csupán a private kulcsszót elé tennünk. A classok tagjai alapértelmezetten privátok míg a structoknak publikusak alapértelmezetten a tagjaik. A konstruktorok lehetőség adnak arra, hogy az osztályok inicializálódni tudjanak létrejöttükkor. Ezeknek azonosítója megegyezik az osztály nevével. C++ sablonok arra használatosak, hogy osztályok és függvények deklarációjakor néhány adatelemnek felparaméterezzük a típusát ezáltal létrehozva egy dinamikus dolgot.

## **III. rész**

### **Második felvonás**

**Bátf41 Haxor Stream**

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

---

DRAFT

---

# 11. fejezet

## Helló, Arroway!

### 11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 11.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...



## **IV. rész**

### **Irodalomjegyzék**

DRAFT

## 11.3. Általános

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

## 11.4. C

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

## 11.5. C++

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

## 11.6. 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 UDPROG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

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