

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	Tóth, Balázs	2019. szeptember 30.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.1.0	2019-02-26	Feladatok elkezdése, chapter 1	Balázs Tóth
0.1.1	2019-03-03	Feladatok elkezdése, chapter 2	Balázs Tóth
0.1.2	2019-03-11	Feladatok elkezdése, chapter 3	Balázs Tóth
0.1.3	2019-03-18	Feladatok elkezdése, chapter 4	Balázs Tóth

REVISION HISTORY

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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1.0.7	2019-10-2	Liskov chapter befejezése	Balázs Tóth
1.0.8	2019-10-9	Mandelbrot chapter elkezdése	Balázs Tóth

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun tétel	14
2.8. A Monty Hall probléma	15
3. Helló, Chomsky!	17
3.1. Decimálisból unárisba átváltó Turing gép	17
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	17
3.3. Hivatkozási nyelv	18
3.4. Saját lexikális elemző	19
3.5. Leetspeak	20
3.6. A források olvasása	22
3.7. Logikus	23
3.8. Deklaráció	24

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

9. Helló, Schwarzenegger!	92
9.1. Szoftmax Py MNIST	92
9.2. Mély MNIST	95
9.3. Minecraft-MALMÖ	100
10. Helló, Chaitin!	101
10.1. Iteratív és rekurzív faktoriális Lisp-ben	101
10.2. Gimp Scheme Script-fu: króm effekt	101
10.3. Gimp Scheme Script-fu: név mandala	101
III. Második felvonás	107
11. Helló, Berners-Lee!	109
11.1. Olvasónaplók	109
12. Helló, Arroway!	112
12.1. OO szemlélet	112
12.2. "Gagyí"	114
12.3. Yoda	115
12.4. Homokozó	116
12.5. Kódolás from scratch	116
13. Helló, Liskov!	118
13.1. Liskov helyettesítés sértése	118
13.2. Szülő-gyerek	118
13.3. Anti OO	118
13.4. Hello, Android!	118
13.5. Ciklomatikus komplexitás	119
13.6. deprecated - Hello, Android!	119
13.7. Hello, SMNIST for Humans!	119
14. Helló, Mandelbrot!	120
14.1. Reverse engineering UML osztálydiagram	120
14.2. Forward engineering UML osztálydiagram	120
14.3. Egy Esettan	120
14.4. BPMN	120
14.5. TEX uml	120
14.6. BPEL Helló, Világ! - egy visszhang folyamat	121

IV. Irodalomjegyzék	122
14.7. Általános	123
14.8. C	123
14.9. C++	123
14.10Lisp	123

DRAFT

Ábrák jegyzéke

3.1. Az átváltó Turing gép	17
3.2. A környezetfüggő grammatikák	18
4.1. A double ** háromszögmátrix a memóriában	30
5.1. A Mandelbrot halmaz a komplex síkon	43
7.1. A hangyaszimuláció UML diagram	74
7.2. A hangyák akcióban	75
7.3. Az életjáték futás közben	84
12.1. A JDK kód	112
12.2. A gagyi futás	115
12.3. A yoda futás	116
12.4. A piBBP futás	117

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?

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

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

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

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



A DocBook XML 5.1 új neked?

Ez esetben forgasd a <https://tdg.docbook.org/tdg/5.1/> könyvet, a végén 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: <https://github.com/ghjbku/DE/blob/master/video.flv>

Megoldás forrása:

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

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

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

Megoldás videó:

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

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

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

2.4. Labdapattogás

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

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

Megoldás forrása:

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

```
//Labdapattogás if nélkül (mentorálva Gila Attila által)
#include<stdio.h>
#include<math.h>

#define szel 80
#define mag 24

int putX(x,y)
{
    int ix,iy;

    for(ix=0;ix<x;ix++)
        printf("\n");

    for(iy=0;iy<y;iy++)
        printf(" ");

    printf("O\n");
}
```

```
return 0;
}

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

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

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

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

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

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

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

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

int main(){

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

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

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

return 0;
}
```

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

Tanulságok, tapasztalatok, magyarázat...

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

2.6. Helló, Google!

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

Megoldás videó:

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

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

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

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

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



```
{0.0, 0.0, 1.0 / 3.0, 0.0},
{1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
{0.0, 1.0 / 2.0, 0.0, 0.0},
{0.0, 0.0, 1.0 / 3.0, 0.0}
};

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

A pagerank algoritmust a google fejlesztette ki azzal a céllal, hogy a weboldalak minőségét rangsorolja.

A feladat komplexitása miatt a soronkénti értelemzést választottam.Kezdjük is el.

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

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

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

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

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

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

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

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

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

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

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

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

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

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

```
library(matlab)

stp <- function(x) {

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

x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

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

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

```
library(matlab)

stp <- function(x) {

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

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

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

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

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

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

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

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

for (i in 1:kiserletek_szama) {

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

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

  }else{

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

  }

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

```
}

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

for (i in 1:kiserletek_szama) {

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

}

valtoztatesnyer = which(kiserlet==valtoztat)

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

Tanulságok, tapasztalatok, magyarázat...

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

3. fejezet

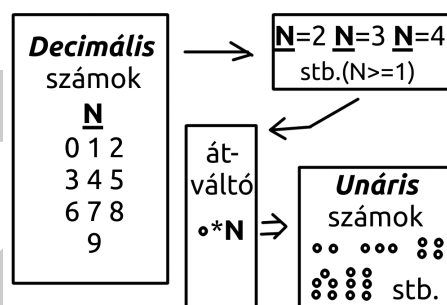
Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:



3.1. ábra. Az átváltó Turing gép

Tanulságok, tapasztalatok, magyarázat...

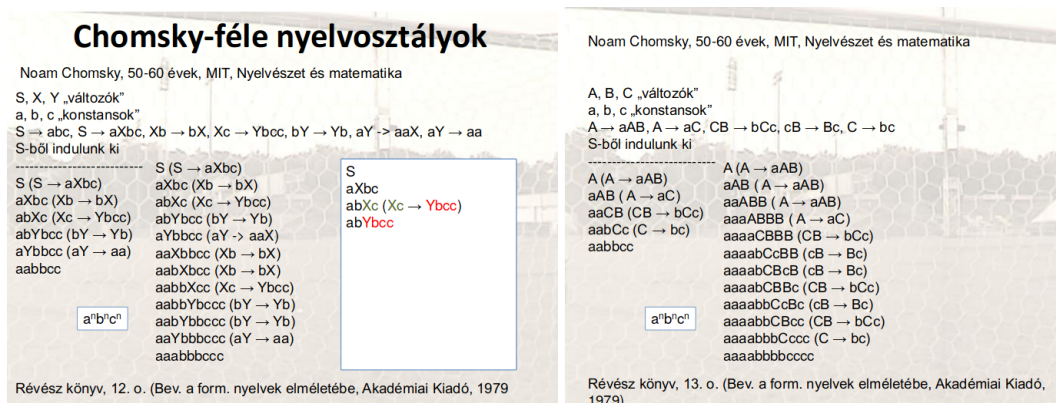
Az átváltó bekér egy Decimális számot, legyen ez egy tetszőleges szám, és nevezzük el N-nek. Ezután egy képlettel átváltja azt Unáris számrendszerbe. Ez a képlet a következő: **unar=egysegelem*N** majd "kirajzolja" az eredményt.

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

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

Megoldás videó:

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



3.2. ábra. A környezetfüggő grammatikák

Tanulságok, tapasztalatok, magyarázat...

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

Környezetfüggő(hossznemcsökkentő)

$P_1XP_2 \rightarrow P_1QP_2$, P_1, P_2 eleme $(VN \cup VT)^*$, $X \in VN$ belülről, $Q \in (VN \cup VT)^+$ belülről, kivéve $S \rightarrow \epsilon$, de akkor S nem lehet jobb oldali egyetlen szabályban sem, tehát Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

3.4. Saját lexikális elemző

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

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1)

```
//lex fájl
//fordítás c-re : lex -o output.c lexfajl.1 (szükséges hozzá a flex ↔
)
//jelen esetben: lex -o realnumbers.c realnumbers.1

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.1)

```
%{
//a fordítás megegyezik az előző feladatával: lex -o output.c lexfájl.1
//majd gcc output.c -o output
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>

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

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

    {'a', {"4", "4", "@", "/-\\\"}},
    {'b', {"b", "8", "|3", "|\"}},
    {'c', {"c", "(", "<", "{"}},
    {'d', {"d", "|)", "|]", "|\"}},
    {'e', {"3", "3", "3", "3\"}},
    {'f', {"f", "|=", "ph", "|#\"}},
    {'g', {"g", "6", "[", "[+"}},
    {'h', {"h", "4", "|-|", "[-\"}},
    {'i', {"1", "1", "|", "!"}},
    {'j', {"j", "7", "_|", "_/"}},
    {'k', {"k", "|<", "1<", "|{"}},
    {'l', {"l", "1", "|", "|_"}},
    {'m', {"m", "44", "(V)", "\\|\"}},
    {'n', {"n", "\\|\\|", "/\\\"}},
    {'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;
}
```

Tanulságok, tapasztalatok, magyarázat...

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

3.6. A források olvasása

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

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

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



Bugok

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

i.

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

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

ii.

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

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

iii.

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

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

iv.

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

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

v.

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

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

vi.

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

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

vii.

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

visszaad kettő számot, melyekből az egyik egy függvény visszatérési értéke

viii.

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

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

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

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

3.8. Deklaráció

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

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

  
egy szám típusú **a** változót
- ```
int *b = &a;
```


egy szám típusú **b** **pointert**, aminek az értéke **a** **memóriában foglalt helye**
- ```
int &r = a;
```

  
integer típusú **r** **értéke a** lesz
- ```
int c[5];
```


létrehoz egy 5 számnak helyet adó **c** tömböt
- ```
int (&tr)[5] = c;
```

  
létrehoz egy **tr** **tömb referenciát**, melynek az értéke **c**
- ```
int *d[5];
```


egy egészekre mutató **d** **tömb pointer**

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

### **h funkcióra mutató pointer**

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

egy pointer, ami az **l függvényre mutató pointerre mutat**

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

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

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

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

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:

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

```
#include <stdio.h>

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

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

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

int
```

```
main ()
{
    int (*f) (int, int);

    f = sum;

    printf ("%d\n", f (2, 3));

    int (*(g) (int)) (int, int);

    g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

```
#include <stdio.h>

typedef int (*F) (int, int);
typedef int (*(G) (int)) (int, int);

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

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

F sumormul (int c)
{
    if (c)
        return mul;
    else
        return sum;
}

int
main ()
{
    F f = sum;
```

```
printf ("%d\n", f (2, 3));  
  
G g = sumormul;  
  
f = *g (42);  
  
printf ("%d\n", f (2, 3));  
  
return 0;  
}
```


4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Írj egy olyan malloc és free párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárbán!

Megoldás videó: <https://youtu.be/1MRTuKwRsB0>, <https://youtu.be/RKbX5-EWpzA>.

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c)

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

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

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

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

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

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

```
}

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

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

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

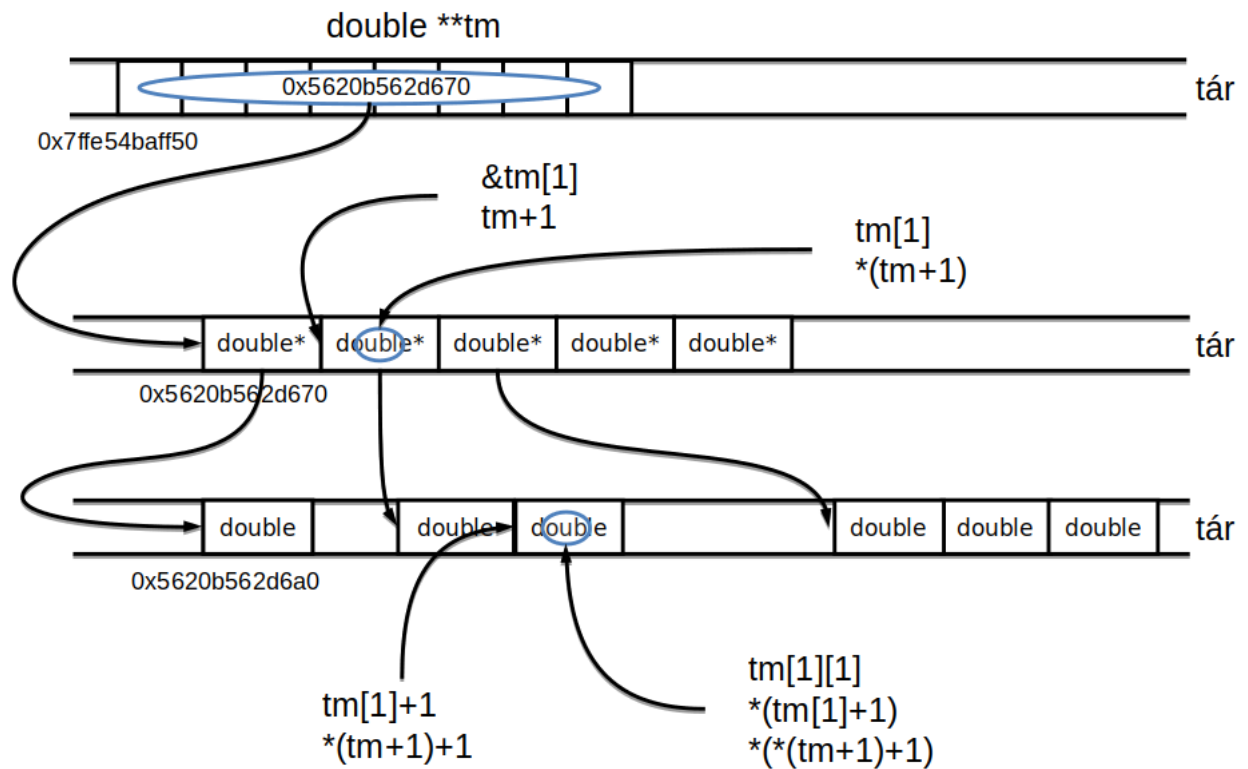
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

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

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

free (tm);

return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

Az

```
int nr = 5;
```

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

```
double **tm;
```

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

Ezek a sorok:

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

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

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

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

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

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

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

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

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

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Tutoráltam: Schachinger Zsoltot Megoldás videó:

Megoldás forrása :

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

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

```
public class ExorTitkosító {

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

    } catch(java.io.IOException e) {

        e.printStackTrace();

    }

}

}
```

Tanulságok, tapasztalatok, magyarázat...

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

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

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

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

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

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

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

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

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

    }

}

int
xor_tores (const char kulcs[], int kulcs_meret, char titkos[],
           int titkos_meret)
{
    xor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);
}

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

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

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

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



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

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

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

    return 0;
}
```

és a többi magos változat:

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

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

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

    return (double) titkos_meret / sz;
}

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

```
{
    // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket

    double szohossz = atlagos_szo_hossz (titkos, titkos_meret);

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

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

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

void
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{
    char *buffer;

    if ((buffer = (char *)malloc(sizeof(char)*titkos_meret)) == NULL)
    {
        printf("Memoria (buffer) falióra\n");
        exit(-1);
    }

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

    if (tiszta_lehet (buffer, titkos_meret))
    {
        printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
            kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],kulcs ↵
            [6],kulcs[7], buffer);
    }
}
```

```

    }

    free(buffer);
}

int
main (void)
{

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

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

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

    // osszes kulcs eloallitasa
#pragma omp parallel for private(kulcs)
    for (int ii = '0'; ii <= '9'; ++ii)
        for (int ji = '0'; ji <= '9'; ++ji)
            for (int ki = '0'; ki <= '9'; ++ki)
                for (int li = '0'; li <= '9'; ++li)
                    for (int mi = '0'; mi <= '9'; ++mi)
                        for (int ni = '0'; ni <= '9'; ++ni)
                            for (int oi = '0'; oi <= '9'; ++oi)
                                for (int pi = '0'; pi <= '9'; ++pi)
                                    {
                                        kulcs[0] = ii;
                                        kulcs[1] = ji;
                                        kulcs[2] = ki;
                                        kulcs[3] = li;
                                        kulcs[4] = mi;
                                        kulcs[5] = ni;
                                        kulcs[6] = oi;
                                        kulcs[7] = pi;

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

```

```
    }  
  
    return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

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

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

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

4.5. Neurális OR, AND és EXOR kapu

R

Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

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

```
# Copyright (C) 2019 Dr. Norbert 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])
```

Tanulságok, tapasztalatok, magyarázat...

Ez a program a neurális hálóra alapszik. **A neurális hálózat biológiai neuronok összekapcsolt csoportja. Modern használatban a szó alatt a mesterséges neurális hálót értjük, amelyek mesterséges neuronokból állnak.** forrás: https://hu.wikipedia.org/wiki/Neur%C3%A1lis_h%C3%A1l%C3%B3zat

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

Az OR kapu és annak "plotolása" itt található

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

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

plot(nn.or)
```

Az ORAND pedig itt:

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

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

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

plot(nn.orand)

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

Végül pedig az EXOR kapu:

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

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

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

plot(nn.exor)

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

4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

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

```
//main.cpp fájl tartalma
#include <iostream>
#include "mlp.cpp"
#include <png++/png.hpp>

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

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

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

    for(int i {0};i<png_image.get_width();++i)
        for(int j{0};j<png_image.get_height();++j)
            image[i*png_image.get_Width()+j]= png_image[i][j].red;
    double value = (*p) (image);
    std::cout <<value<<std::endl;

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

Tanulságok, tapasztalatok, magyarázat...

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

5. fejezet

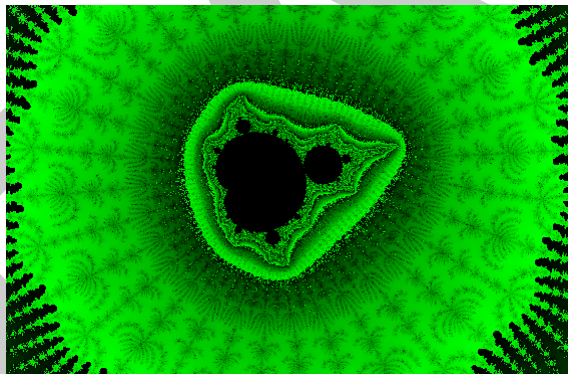
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása: `/mandelpngt.c++` nevű állománya.



5.1. ábra. A Mandelbrot halmaz a komplex síkon

A Mandelbrot halmazt 1980-ban találta meg Benoit Mandelbrot a komplex számsíkon. Komplex számok azok a számok, amelyek körében válaszolni lehet az olyan egyébként értelmezhetetlen kérdésekre, hogy melyik az a két szám, amelyet összeszorozva -9-et kapunk, mert ez a szám például a $3i$ komplex szám.

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800×800 -as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1} = z_n^2 + c$, ($0 \leq n$) képlet alapján úgy, hogy a c az éppen vizsgált rácspon. A z_0 az origó. Alkalmazva a képletet a

- $z_0 = 0$
- $z_1 = 0^2 + c = c$

- $z_2 = c^2 + c$
- $z_3 = (c^2 + c)^2 + c$
- $z_4 = ((c^2 + c)^2 + c)^2 + c$
- ... s így tovább.

Azaz kiindulunk az origóból (z_0) és elugrunk a rács első pontjába a $z_1 = c$ -be, aztán a c -től függően a további z -kbe. Ha ez az utazás kivezet a 2 sugarú körből, akkor azt mondjuk, hogy az a vizsgált rácpont nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végtelen sok z -t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácponthoz. Ha eközben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácpont a halmaz része. (Színes meg úgy lesz a kép, hogy változatosan színezzük, például minél későbbi z -nél lép ki a körből, annál sötétebbre).

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

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

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

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhax/attention-raising/Mandelbrot/3.1.2.cpp](https://github.com/bhaxor/attention-raising-Mandelbrot/3.1.2.cpp) nevű állománya.

```
// Verzio: 3.1.2.cpp
// Fordítás:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatás:
// ./3.1.2 mandel.png 1920 1080 2040 ↵
-0.01947381057309366392260585598705802112818 ↵
-0.0194738105725413418456426484226540196687 ↵
0.7985057569338268601555341774655971676111 ↵
0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ↵
0.4127655418209589255340574709407519549131 ↵
0.4127655418245818053080142817634623497725 ↵
0.2135387051768746491386963270997512154281 ↵
0.2135387051804975289126531379224616102874
// Nyomtatás:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --line-numbers=1 --left-footer=" ↵
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

            ++iteracio;
        }

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

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

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

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

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

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

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

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

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

```

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

    }

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

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

```

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

5.3. Biomorfok

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

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

A biomorfokra (a Julia halmazokat rajzoló bug-os programjával) rátaláló Clifford Pickover azt hitte természeti törvényre bukkant: https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf (lásd a 2307. oldal aljától).

A különbség a **Mandelbrot halmaz** és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a `c` változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a `c` befutja a vizsgált összes rácspontot.

```

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

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

        reC = a + k * dx;

```

```
imC = d - j * dy;
std::complex<double> c ( reC, imC );

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

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

    ++iteracio;
}
```

Ezzel szemben a Julia halmazos csipetben a cc nem változik, hanem minden vizsgált z rácspontra ugyanaz.

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon
    for ( int k = 0; k < szelesseg; ++k )
    {
        double reZ = a + k * dx;
        double imZ = d - j * dy;
        std::complex<double> z_n ( reZ, imZ );

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

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

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

```
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -l --line-numbers=1 --left-footer=" ↵
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↵
Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

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

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: [bhax/attention-raising/CUDA/mandelpngc_60x60_100.cu](https://bhax.attention-raising/CUDA/mandelpngc_60x60_100.cu) nevű állománya.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás videó: Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazsal.

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

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

5.6. Mandelbrot nagyító és utazó Java nyelven

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

Köszönet a forráskódért: [Lovász Botond](#)

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

public class Mandelbrot extends JFrame implements ActionListener {

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

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

        Container contentPane = getContentPane();

        contentPane.setLayout(null);

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

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

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

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

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

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

        validate();
    }

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

        }

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

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

```
        }
    }
}

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

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

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

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

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

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

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

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

```
public class PolárGenerátor {

    boolean nincsTárolt = true;
    double tárolt;

    public PolárGenerátor() {

        nincsTárolt = true;

    }

    public double következő() {

        if(nincsTárolt) {

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

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

                w = v1*v1 + v2*v2;
            } while (w > 1);
            tárolt = Math.sqrt(w);
            return (Math.atan2(v2, v1) / tárolt);
        }
        return tárolt;
    }
}
```

```
        } while (w > 1);

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

        tárolt = r*v2;
        nincsTárolt = !nincsTárolt;

        return r*v1;

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

public static void main(String[] args) {

    PolárGenerátor g = new PolárGenerátor();

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

}
```

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

```
#include "std_lib_facilities.h"

class PolarGenerator{

bool nincsTarolt=true;
double tarolt;

public :
double kovetkezo()
{

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

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

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

};

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

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

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

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

6.2. LZW

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

Megoldás videó:

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:

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



```
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail ←
// .com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1, http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2, csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3, http://progpater.blog.hu/2011/03/05/ ←
//     labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
//
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

typedef struct binfa
{
    int ertek;
    struct binfa *bal_nulla;
    struct binfa *jobb_egy;
```

```
} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

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

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

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

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

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

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

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

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

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

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

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

rszoras (gyoker);

double szoras = 0.0;

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

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

szabadit (gyoker);
```

```
}

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

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

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

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

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

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

```
{

    ++atlagdb;
    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);
    }
}
```

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

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

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

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

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

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

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

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

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

```
}
```

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

```
else
{
    if (fa->jobb_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;
    }
}
```

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

6.3. Fabejárás

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

Megoldás videó:

Preorderbejárás: azaz a gyöker elem majd a bal oldali részfa preorder bejárása, végül a jobb oldali részfa preorder bejárása.

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

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

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

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

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

        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

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

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

        --melyseg;
    }
}
```

forrás

6.4. Tag a gyökér

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

Megoldás videó:

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

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

};
```

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

6.5. Mutató a gyökér

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

Megoldás videó:

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

```
LZWBinFa ():fa (gyoker = new Csomopont('/'))
{
}
~LZWBinFa ()
{
    szabadit (gyoker->egyGyermek ());
    szabadit (gyoker->nullasGyermek ());
    delete gyoker;
}
```

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

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

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

Viszont ez még mindig nem elég, hiszen a memórafoglalást követően valahogy fel is kell őket szabadítani. És itt jön be az ~LZWBinFa destruktork.

6.6. Mozgató szemantika

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

Megoldás videó:

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

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

```
{
Node *uj = new Node ('1');
currentNode->newRightChild (uj);
currentNode = root;
}
else
{
currentNode = currentNode->getRightChild ();
}
}
}
void print (void)
{
depth = 0;
print (root, std::cout);
}

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

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

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

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

A lent található két függvény teszi lehetővé az objektumok mozgatását.

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

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

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

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

7. fejezet

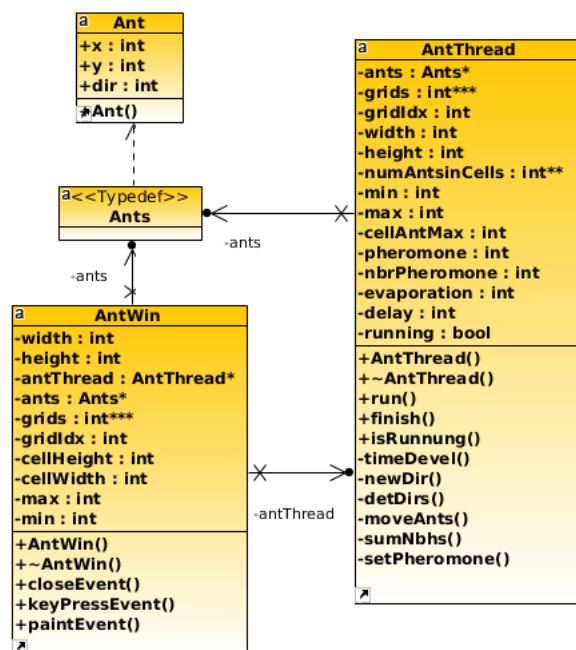
Helló, Conway!

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>

A forráskód Bátfai Norbert tulajdonában áll. Megoldás forrása:



7.1. ábra. A hangyaszimuláció UML diagram



7.2. ábra. A hangyák akcióban

```
// BHAX Myrmecologist
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// https://bhaxor.blog.hu/2018/09/26/hangyaszimulaciok
// https://bhaxor.blog.hu/2018/10/10/myrmecologist
//

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



```
#include "antwin.h"

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

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

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

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

```
parser.addOption ( cellamerete_opt );

parser.process ( a );

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

qsrand ( QDateTime::currentMSecsSinceEpoch() );

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

w.show();

return a.exec();
}
```

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

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

7.2. Java életjáték

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

Megoldás videó:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    return állapotúSzomszéd;
}

public void időFejlődés() {

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

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

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

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

        if(rácsElőtte[i][j] == ÉLŐ) {

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

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

public void run() {

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

        időFejlődés();
        repaint();
    }
}

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

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

}

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

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

```
rács[y+ 7][x+ 0] = ÉLŐ;  
rács[y+ 7][x+ 1] = ÉLŐ;  
  
rács[y+ 3][x+ 13] = ÉLŐ;  
  
rács[y+ 4][x+ 12] = ÉLŐ;  
rács[y+ 4][x+ 14] = ÉLŐ;  
  
rács[y+ 5][x+ 11] = ÉLŐ;  
rács[y+ 5][x+ 15] = ÉLŐ;  
rács[y+ 5][x+ 16] = ÉLŐ;  
rács[y+ 5][x+ 25] = ÉLŐ;  
  
rács[y+ 6][x+ 11] = ÉLŐ;  
rács[y+ 6][x+ 15] = ÉLŐ;  
rács[y+ 6][x+ 16] = ÉLŐ;  
rács[y+ 6][x+ 22] = ÉLŐ;  
rács[y+ 6][x+ 23] = ÉLŐ;  
rács[y+ 6][x+ 24] = ÉLŐ;  
rács[y+ 6][x+ 25] = ÉLŐ;  
  
rács[y+ 7][x+ 11] = ÉLŐ;  
rács[y+ 7][x+ 15] = ÉLŐ;  
rács[y+ 7][x+ 16] = ÉLŐ;  
rács[y+ 7][x+ 21] = ÉLŐ;  
rács[y+ 7][x+ 22] = ÉLŐ;  
rács[y+ 7][x+ 23] = ÉLŐ;  
rács[y+ 7][x+ 24] = ÉLŐ;  
  
rács[y+ 8][x+ 12] = ÉLŐ;  
rács[y+ 8][x+ 14] = ÉLŐ;  
rács[y+ 8][x+ 21] = ÉLŐ;  
rács[y+ 8][x+ 24] = ÉLŐ;  
rács[y+ 8][x+ 34] = ÉLŐ;  
rács[y+ 8][x+ 35] = ÉLŐ;  
  
rács[y+ 9][x+ 13] = ÉLŐ;  
rács[y+ 9][x+ 21] = ÉLŐ;  
rács[y+ 9][x+ 22] = ÉLŐ;  
rács[y+ 9][x+ 23] = ÉLŐ;  
rács[y+ 9][x+ 24] = ÉLŐ;  
rács[y+ 9][x+ 34] = ÉLŐ;  
rács[y+ 9][x+ 35] = ÉLŐ;  
  
rács[y+ 10][x+ 22] = ÉLŐ;  
rács[y+ 10][x+ 23] = ÉLŐ;  
rács[y+ 10][x+ 24] = ÉLŐ;  
rács[y+ 10][x+ 25] = ÉLŐ;  
  
rács[y+ 11][x+ 25] = ÉLŐ;
```

```
}

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

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

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

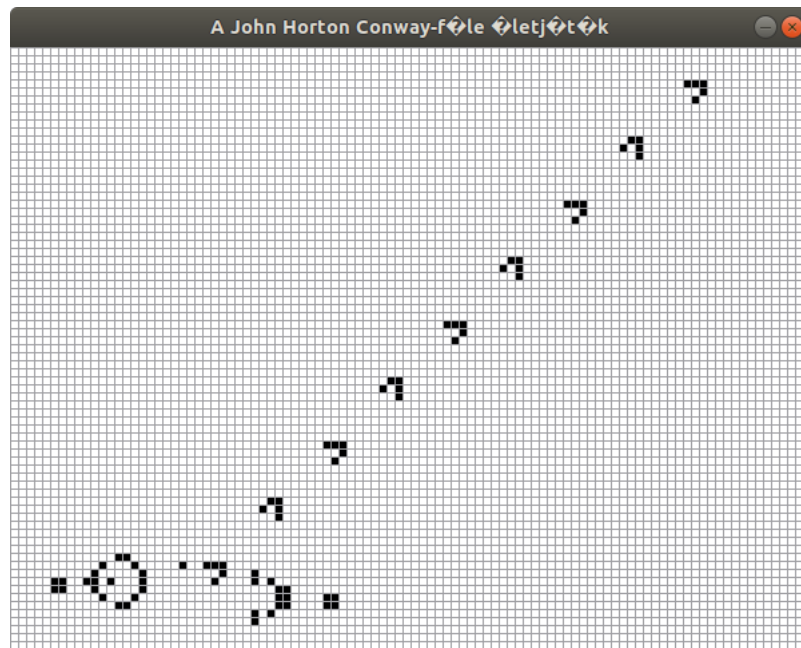
Tanulságok, tapasztalatok, magyarázat... A Java verzió ugyanazt tudja, mint a C++ verzió, persze a programnyelv miatt vannak különbségek a forráskódban. A Java kompatibilitása miatt viszont szinte bárhol lefuttatható a program, ahol jvm található. A programban megtalálható pár billentyű funkció is, melyekkel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

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

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



7.3. ábra. Az életjáték futás közben

```
//main.cpp tartalma
#include <QApplication>
#include "sejtablak.h"
#include <QDesktopWidget>

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

    return a.exec();
}
```

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

```
//sejtablak.h tartalma
#ifndef SEJTABLAK_H
#define SEJTABLAK_H

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

class SejtSzal;
```

```
class SejtAblak : public QMainWindow
{
    Q_OBJECT

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

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

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

    int szelesseg;
    int magassag;
    void paintEvent(QPaintEvent*);
    void siklo(bool **racs, int x, int y);
    void sikloKilovo(bool **racs, int x, int y);

private:
    SejtSzal* eletjatek;

};

#endif // SEJTABLAK_H
```

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

//sejtszal.h tartalma

```
#ifndef SEJTSZAL_H
#define SEJTSZAL_H

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

class SejtAblak;

class SejtSzal : public QThread
{
    Q_OBJECT

public:
    SejtSzal(bool ***racso, int szelesseg, int magassag,
             int varakozas, SejtAblak *sejtAblak);
    ~SejtSzal();
    void run();

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

#endif // SEJTSZAL_H
```

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

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

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: </esport-talent-search/>

Tanulságok, tapasztalatok, magyarázat...

DRAFT

8. fejezet

Helló, Gutenberg!

8.1. Programozási alapfogalmak

[?]

Egy számítógép programozására három nyelvi szintet különböztetünk meg:

- Gépi kód
- assembly szint
- Magas szint

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

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

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

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

Utasítások:

- értékadó utasítás: Ezen típusú utasítás a változók értékét módosítja a ↵
program futása során.
- Üres utasítás: ilyenkor a processor egy üres gépi utasítást hajt végre.
- Ugró utasítás: A program egyik soráról a másikra ugrik a vezérlés, ↵
általános alak a GOTO.
- Elágazásos utasítás(kétfelé ágazás utasítás): A program egy pontján két ↵
lehetőség közül választ a program egy feltétel alapján.
- Formája: if...then...else...
- Többirányú elágazás:A program egy pontján meghatározott számú opciókból ↵
kiválaszt egyet, amelyet végrehajt.

A választást egy kifejezés értéke szerint határozzuk meg.

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

A case-ágak értékei különbözzenek.

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

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

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

8.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

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

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

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

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

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

értékadással: char c = 'a'; int i = 0;

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

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

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

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

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

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

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

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

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

8.3. Programozás

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

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

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

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

objektumok és osztályok

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

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

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

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

Konstansok és inline függvények

A konstansokat a "közönséges" változókhoz hasonlóan használhatjuk, viszont ezek értékei nem változhatnak a program futása során. Bármiféle változtatás az értékével programhibához vezet.

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

elmelet/testprog.c++

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

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

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

Ezek kombinálása is lehetséges.

Inline függvények

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

I/O alapok

operátor túlterhelés

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

9. fejezet

Helló, Schwarzenegger!

9.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ↵
=====

"""Builds the MNIST network.

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

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

```
This file is used by the various "fully_connected_*.py" files and not meant to
be run.
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import math

import tensorflow as tf

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

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

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

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

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

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

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

    Args:
        logits: Logits tensor, float - [batch_size, NUM_CLASSES].
        labels: Labels tensor, int32 - [batch_size].

    Returns:
        loss: Loss tensor of type float.
    """
    labels = tf.to_int64(labels)
    cross_entropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
        logits, labels, name='xentropy')
    loss = tf.reduce_mean(cross_entropy, name='xentropy_mean')
    return loss

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

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

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

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

    Args:
        loss: Loss tensor, from loss().
        learning_rate: The learning rate to use for gradient descent.

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

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

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

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

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

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

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

9.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ↩
=====

"""A deep MNIST classifier using convolutional layers.
See extensive documentation at
https://www.tensorflow.org/get_started/mnist/pros
"""
# Disable linter warnings to maintain consistency with tutorial.
# pylint: disable=invalid-name
# pylint: disable=g-bad-import-order

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

import argparse
import sys
import tempfile

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

FLAGS = None

def deepnn(x):
    """deepnn builds the graph for a deep net for classifying digits.
    Args:
        x: an input tensor with the dimensions (N_examples, 784), where 784 is ↩
            the
            number of pixels in a standard MNIST image.
    Returns:
        A tuple (y, keep_prob). y is a tensor of shape (N_examples, 10), with ↩
```

```
    values
    equal to the logits of classifying the digit into one of 10 classes ( ←
    the
    digits 0-9). keep_prob is a scalar placeholder for the probability of
    dropout.
"""
# Reshape to use within a convolutional neural net.
# Last dimension is for "features" - there is only one here, since images ←
    are
# grayscale -- it would be 3 for an RGB image, 4 for RGBA, etc.
with tf.name_scope('reshape'):
    x_image = tf.reshape(x, [-1, 28, 28, 1])

# First convolutional layer - maps one grayscale image to 32 feature maps ←
    .
with tf.name_scope('conv1'):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

# Pooling layer - downsamples by 2X.
with tf.name_scope('pool1'):
    h_pool1 = max_pool_2x2(h_conv1)

# Second convolutional layer -- maps 32 feature maps to 64.
with tf.name_scope('conv2'):
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)

# Second pooling layer.
with tf.name_scope('pool2'):
    h_pool2 = max_pool_2x2(h_conv2)

# Fully connected layer 1 -- after 2 round of downsampling, our 28x28 ←
    image
# is down to 7x7x64 feature maps -- maps this to 1024 features.
with tf.name_scope('fc1'):
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Dropout - controls the complexity of the model, prevents co-adaptation ←
    of
# features.
with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

```
# Map the 1024 features to 10 classes, one for each digit
with tf.name_scope('fc2'):
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])

    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
    return y_conv, keep_prob

def conv2d(x, W):
    """conv2d returns a 2d convolution layer with full stride."""
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    """max_pool_2x2 downsamples a feature map by 2X."""
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                           strides=[1, 2, 2, 1], padding='SAME')

def weight_variable(shape):
    """weight_variable generates a weight variable of a given shape."""
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    """bias_variable generates a bias variable of a given shape."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

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

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])

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

    # Build the graph for the deep net
    y_conv, keep_prob = deepnn(x)

    with tf.name_scope('loss'):
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                                logits=y_conv)
    cross_entropy = tf.reduce_mean(cross_entropy)
```

```
with tf.name_scope('adam_optimizer'):
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
    accuracy = tf.reduce_mean(correct_prediction)

graph_location = tempfile.mkdtemp()
print('Saving graph to: %s' % graph_location)
train_writer = tf.summary.FileWriter(graph_location)
train_writer.add_graph(tf.get_default_graph())

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        if i % 100 == 0:
            train_accuracy = accuracy.eval(feed_dict={
                x: batch[0], y_: batch[1], keep_prob: 1.0})
            print('step %d, training accuracy %g' % (i, train_accuracy))
            train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

    print('test accuracy %g' % accuracy.eval(feed_dict={
        x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str,
                        default='/tmp/tensorflow/mnist/input_data',
                        help='Directory for storing input data')
    FLAGS, unparsed = parser.parse_known_args()
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

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

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

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

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

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

9.3. Minecraft-MALMÖ

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

Megoldás forrása:

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

DRAFT

10. fejezet

Helló, Chaitin!

10.1. Iteratív és rekurzív faktoriális Lisp-ben

Az SMNIST-el kiváltva

Megoldás videó:

Megoldás forrása:

10.2. Gimp Scheme Script-fu: króm effekt

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

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

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

Tanulságok, tapasztalatok, magyarázat...

10.3. Gimp Scheme Script-fu: név mandala

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

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

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

```
; bhax_mandala9.scm
;
; BHAX-Mandala creates a mandala from a text box.
; Copyright (C) 2019 Norbert Bátfai, batfai.norbert@inf.unideb.hu
;
; This program is free software: you can redistribute it and/or modify
```

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

(define (elem x lista)

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

)

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↵
    PIXELS font)))

  text-width
)
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  ;;
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↵
```

```
PIXELS font)))
;;; ved ki a lista 2. elemét
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ↵
  fontsize PIXELS font)))
;;;

(list text-width text-height)
)
)

; (text-width "alma" "Sans" 100)

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

    (gimp-image-insert-layer image layer 0 0)

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

    (gimp-context-set-foreground color)

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

    (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
    (gimp-image-insert-layer image text-layer 0 -1)
    (gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

)

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

(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"      "Bátf41 Haxor"
  SF-STRING      "Text2"     "BHAX"
  SF-FONT         "Font"      "Sans"
  SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
  SF-VALUE        "Width"     "1000"
  SF-VALUE        "Height"    "1000"
  SF-COLOR        "Color"     '(255 0 0)
  SF-GRADIENT     "Gradient"  "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

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

```
(define (elem x lista)

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

)
```

Mint láthatjuk, itt a define kulcsszóval lehet függvényeket inicializálni. Ez a függvény az "elem" névre hallgat, és 2 paramétert kér be, az "x"-et és a "lista"-át. Ha az x=1, akkor a (car lista) parancs hajtódik végre, ha nem, akkor pedig az (elem (x-1) (cdr lista)) parancs. Ez miatt a sor miatt a függvény rekurzív lesz, hiszen addig hajtja végre az "else" ágat, amíg az x!=1.

```
(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font))))

  text-width
)
)
```

Ez a függvény a szöveg szélességét kezeli. paraméterként bekéri a szöveget, a betűtípust és a betűméretet. A függvény törzsében a megadott paraméterekkel a gimp beállítja a szöveg kinézetét.

```
(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
  gradient)
(let*
  (
```

Ezen kezdetű függvény tekinthető a main függvénynek, hiszen itt alkalmazzuk az előbb megadott függvényeket, itt írjuk a program érdemi részét.

```
(gimp-image-insert-layer image layer 0 0)

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

Az első sor beilleszt egy layert, a második sor beállítja annak háttérszínét zöldre, a harmadik sor kitölti a hátteret, az utolsó sor pedig újra aktiválja a képet.

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Berners-Lee!

11.1. Olvasónaplók

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

C++ és Java összehasonlítás

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

logikai típus C++-ban bool, Java-ban boolean.

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

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

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

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

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

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

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

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

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

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

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

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

Java

C++

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

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

Python könyv élmény-napló

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

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

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

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

DRAFT

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

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

Megoldás forrása: [github link](#)

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

12.1. ábra. A JDK kód

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

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

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

```
public class polargenerator  
{
```

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

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

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

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

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

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

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

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

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

```
else  
{
```

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

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

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

12.2. "Gagyi"

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

Megoldás forrása:[github link](#)

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

Amint láthatjuk, az első két tagja a ciklus feltételeknek a két változó értékét hasonlítja össze. A harmadik viszont (**t!=x**) a változók referenciáját, ami az integer-cache miatt -128 és 127 között ugyanaz lesz. A gagyi.java fájl ezért végtelenciklust képez, hiszen a -129 már nincs benne a cache-ben, ezért a t!=x mindig igaz lesz. a gagyi2.java viszont a -128 értékkel dolgozik, ami benne van a cache-ben, tehát a ciklus nem jön létre.

Érdekesség még, hogy a Java6 óta bevezették a felső érték bindelesét, amely a

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

```
windsake@windsake-pc:~/Desktop/sajat/prog2/gagy1
bhax-textbook-fdl.pdf
bhax-textbook-fdl.xml
bhax-textbook-feladatok2-arroway.xml
bhax-textbook-feladatok2-berners.xml
bhax-textbook-feladatok2-liskov.xml
bhax-textbook-feladatok2.xml
bhax-textbook-feladatok-caesar.xml
bhax-textbook-feladatok-chaitin.xml
bhax-textbook-feladatok-chomsky.xml
bhax-textbook-feladatok-conway.xml
bhax-textbook-feladatok-gutenberg.xml
bhax-textbook-feladatok-mandelbrot.xml
bhax-textbook-feladatok-schwarzenegger.xml
bhax-textbook-feladatok-turing.xml
bhax-textbook-feladatok-welch.xml
bhax-textbook-feladatok.xml
bhax-textbook-intro.xml
bhax-textbook-motto.xml
bhax-textbook-pre.xml
bhax-textbook-revhistory.xml
bhax-textbook-subtitle.xml
bhax-textbook-titleabbrev.xml
[windsake@windsake-pc prog2]$ cd gagyi/
[windsake@windsake-pc gagyi]$ ls
gagy12.java gagyi.java
[windsake@windsake-pc gagyi]$ javac gagyi.java
[windsake@windsake-pc gagyi]$ java gagyi rev-parse master
-129
-129
C[windsake@windsake-pc gagyi]$ javac gagyi2.java
[windsake@windsake-pc gagyi]$ java gagyi2 remote --verbose
-128
-128
[windsake@windsake-pc gagyi]$
```

12.2. ábra. A gagyi futás

12.3. Yoda

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

Megoldás forrása: [github link](#)

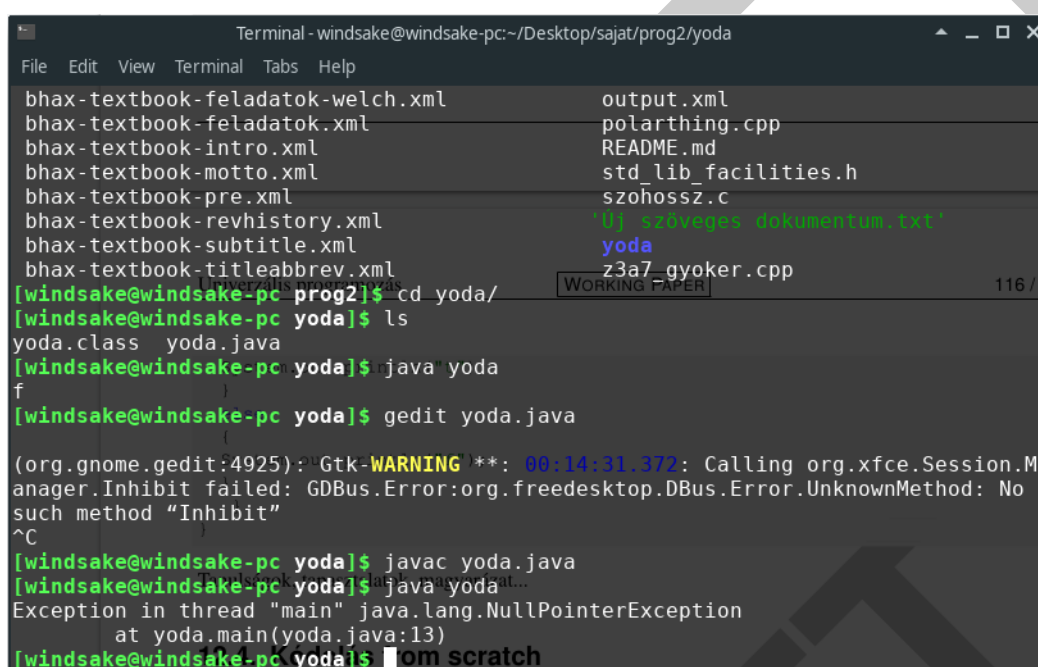
```
public class yoda
{
    String i;
    public yoda()
    {
        i=null;
    }
    public static void main(String[] args)
    {
        yoda f=new yoda();

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



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

A fenti programkód a yoda kondíciós megoldást mutatja, ami annyit tesz, hogy a konstanst rakjuk a ciklus-magban az első helyre, amelyel a NullPointerException kivételeket elkerülhetjük. A kikommentelt részt beillesztve pedig a program lefutásánál megkapjuk a nullpointer exception-t. a **yoda f=new yoda();** sor példányosítja a yoda osztályt.



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

12.3. ábra. A yoda futás

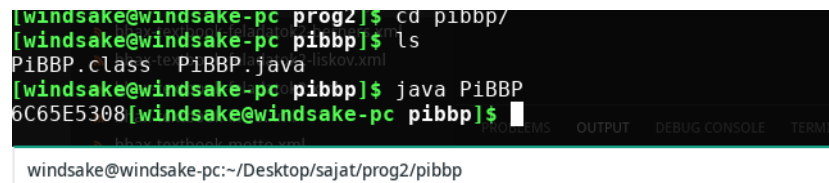
12.4. Homokozó

Megoldás forrása: [github link](#)

12.5. Kódolás from scratch

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

Megoldás forrása: [github link](#)



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

12.4. ábra. A piBBP futás

a **public PiBBP(int d)** rész a $d+1$ hexadecimális jegytől számítja ki a jegyeket, azon belül a 16^d pi képlet kiszámítása. A **public double d16Sj(int d, int j)** eljárás a $\{16^d S_j\}$ részlet kiszámítását végzi. **public long n16modk(int n, int k)** végzi a hatványozás modulóját, itt n a kitevő, k a modulus.

```
public String toString() {
    return d16PiHexaJegyek;
}
```

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

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

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

Megoldás forrása: [github link](#)

13.2. Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

13.3. Anti OO

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

Megoldás forrása: [github link](#)

13.4. Hello, Android!

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

13.5. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf (77-79 főlát)!

13.6. deprecated - Hello, Android!

Élesszük fel a <https://github.com/nbatfai/SamuEntropy/tree/master/cs> projektjeit és vessünk össze néhány egymásra következőt, hogy hogyan változtak a források!

13.7. Hello, SMNIST for Humans!

Fejleszd tovább az SMNIST for Humans projektet SMNIST for Anyone emberre szánt appá! Lásd az [smnist2_kutatasi_jegyzokonyv.pdf](#)-ben a részletesebb háttérrel!

14. fejezet

Helló, Mandelbrot!

14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nIERIEOs. <https://arato.inf.unideb.hu/batfai.norbert/UD> (28-32 fólia)

14.2. Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generáljunk belőle forrást!

14.3. Egy Esettan

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

14.4. BPMN

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

14.5. TEX uml

Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat).

14.6. BPEL Helló, Világ! - egy visszhang folyamat

Egy visszhang folyamat megvalósítása az alábbi teljes „videó tutorial” alapján: https://youtu.be/0OnlYWX2v_I

DRAFT

IV. rész

Irodalomjegyzék

DRAFT

14.7. Általános

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

14.8. C

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

14.9. C++

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

14.10. Lisp

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

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az 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.