

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Nagy Kristián

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	Nagy, Kristián	2019. november 11.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.6	2019-02-20	A tényleges érdemi munka elkezdése.	krisztian2545
0.0.7	2019-02-26	A Turing feladatcsokor gyakorlati feladatainak elkészítése.	krisztian2545
0.0.8	2019-03-05	A Chomsky feladatcsokor gyakorlati feladatainak elkészítése.	krisztian2545

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-03-12	A Turing előadás feladatainak és a Caesar feladatcsokor gyakorlati feladatainak elkészítése.	krisztian2545
0.1.0	2019-03-19	A Chomsky előadás feladatainak és a Mandelbrot feladatcsokor gyakorlati feladatainak elkészítése.	krisztian2545
0.1.1	2019-03-26	A Caesar előadás feladatainak és a Welch feladatcsokor gyakorlati feladatainak elkészítése.	krisztian2545
0.1.2	2019-04-02	A Mandelbrot előadás feladatainak és a Conway feladatcsokor gyakorlati feladatainak elkészítése.	krisztian2545
0.1.3	2019-04-23	A Schwarzenegger feladatcsokor előadás feladatainak elkészítése.	krisztian2545
0.1.4	2019-04-30	A Chaitin feladatcsokor előadás feladatainak elkészítése.	krisztian2545
1.0.0	2019-05-07	Az összes feladat elkészült.	krisztian2545

Ajánlás

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

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

DRAFT

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	7
2.3. Változók értékének felcserélése	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	13
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	18
3.3. Hivatalos nyelv	18
3.4. Saját lexikális elemző	20
3.5. l33t.l	21
3.6. A források olvasása	22
3.7. Logikus	26
3.8. Deklaráció	32

4. Helló, Caesar!	35
4.1. double ** háromszögmátrix	35
4.2. C EXOR titkosító	36
4.3. Java EXOR titkosító	37
4.4. C EXOR törő	38
4.5. Neurális OR, AND és EXOR kapu	40
4.6. Hiba-visszaterjesztéses perceptron	42
5. Helló, Mandelbrot!	44
5.1. A Mandelbrot halmaz	44
5.2. A Mandelbrot halmaz a std::complex osztállyal	47
5.3. Biomorfok	50
5.4. A Mandelbrot halmaz CUDA megvalósítása	53
5.5. Mandelbrot nagyító és utazó C++ nyelven	54
5.6. Mandelbrot nagyító és utazó Java nyelven	56
6. Helló, Welch!	59
6.1. Első osztályom	59
6.2. LZW	61
6.3. Fabejárás	64
6.4. Tag a gyökér	66
6.5. Mutató a gyökér	68
6.6. Mozgató szemantika	69
7. Helló, Conway!	71
7.1. Hangyszimulációk	71
7.2. Java életjáték	72
7.3. Qt C++ életjáték	75
7.4. BrainB Benchmark	78
8. Helló, Schwarzenegger!	80
8.1. Szoftmax Py MNIST	80
8.2. Mély MNIST	83
8.3. Minecraft-MALMÖ	86

9. Helló, Chaitin!	87
9.1. Iteratív és rekurzív faktoriális Lisp-ben	87
9.2. Gimp Scheme Script-fu: króm effekt	88
9.3. Gimp Scheme Script-fu: név mandala	89
10. Helló, Gutenberg!	90
10.1. Juhász István féle könyv	90
10.2. K & R, A C programozási nyelv	91
10.3. BME: Szoftverfejlesztés C++ nyelven / Benedek Zoltán, Levendovszky Tíhamér	92
III. Második felvonás	95
11. Helló, Berners-Lee!	97
11.1. C++ vs Java	97
11.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba: Gyors prototípus-fejlesztés Python és Java nyelven (35-51 oldal)	98
12. Helló, Arroway!	100
12.1. OO szemlélet	100
12.2. Homokatózó	102
12.3. „Gagyí”	105
12.4. Yoda	107
12.5. Kódolás from scratch	108
13. Helló, Liskov!	109
13.1. Liskov helyettesítés sértése	109
13.2. Szülő-gyerek	111
13.3. Anti OO	112
13.4. Hello, Android!	113
13.5. Ciklomatikus komplexitás	116
14. Helló, Mandelbrot!	118
14.1. Reverse engineering UML osztálydiagram	118
14.2. Forward engineering UML osztálydiagram	120
14.3. Egy esettan	130
14.4. BPMN	133
14.5. TeX UML	134

15. Helló, Chomsky!	135
15.1. Encoding	135
15.2. OOCWC lexer	136
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	136
15.4. Paszigráfia Rapszódia LuaLaTeX vizualizáció	139
15.5. Perceptron osztály	139
16. Helló, Stroustrup!	142
16.1. JDK osztályok	142
16.2. Másoló-mozgató szemantika	144
16.3. Hibásan implementált RSA törése	145
16.4. Változó argumentumszámú ctor	146
17. Helló, Gödel!	149
17.1. Gengszterek	149
17.2. C++11 Custom Allocator	150
17.3. STL map érték szerinti rendezése	151
17.4. Alternatív Tabella rendezése	152
17.5. Prolog családfa	152
18. Helló, Anonym!	153
18.1. FUTURE tevékenység editor	153
18.2. OOCWC Boost ASIO hálózatkezelése	153
18.3. SamuCam	153
18.4. BrainB	153
18.5. OSM térképre rajzolása	154
IV. Irodalomjegyzék	155
18.6. Általános	156
18.7. C	156
18.8. C++	156
18.9. Lisp	156

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/turing/inf_sleep.c

bhax/thematic_tutorials/bhax_textbook/turing/infinite_single.c

bhax/thematic_tutorials/bhax_textbook/turing/infinite_multi.c

A végtelen ciklusok nagyon hasznosak tudnak lenni például ha egy menüt akarunk írni ahol semmi másat nem csinálunk, csak várunk valamilyen bemenetre és az alapján végrehajtunk valamit, majd újra várunk. C-ben 2 féle módon írhatunk végtelen ciklust:

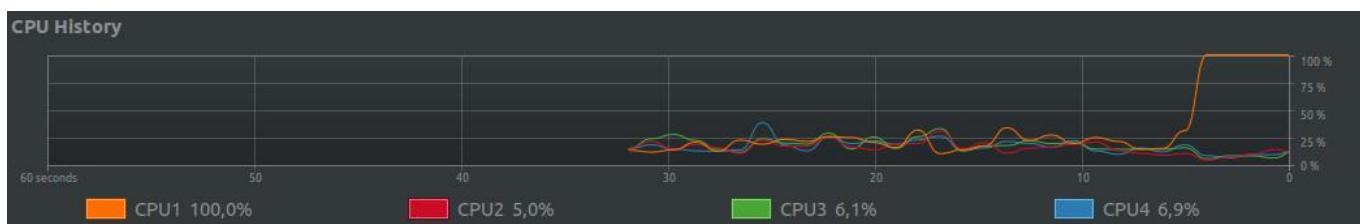
- for ciklussal

```
for(;;){  
    //infinite loop  
}
```

- while ciklussal

```
while(1){  
    //infinite loop  
}
```

Először nézzük meg a processzort 100%-on futtató programot. A könnyebb olvashatóság érdekében a while ciklust használtam és az "1"-es értékkel (amit a fordító logikai igaz értékként értelmez ebben az esetben) a ciklus soha nem ér véget. Mivel a ciklusba nem írtunk semmit, a program a while ciklus feltételének vizsgálásával tölti minden idejét, ami a processzor 100% leterheltségét eredményezi.



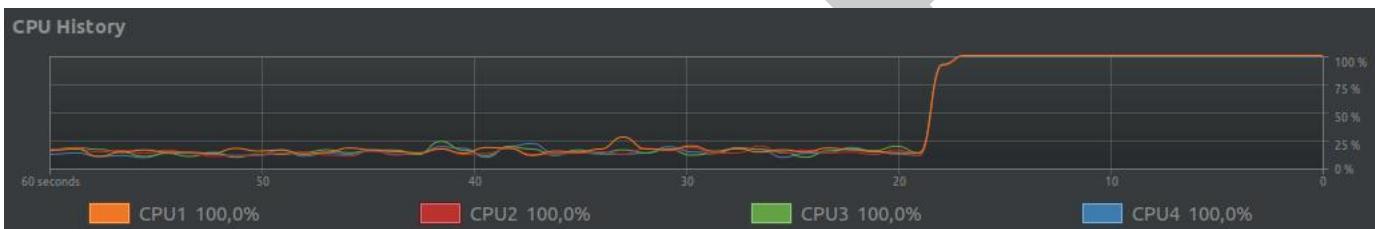
Ahhoz hogy az összes magot 100%-on pörgessük, csak 1 sorral kell kibővítenünk a programot, de ehhez szükségünk lesz az `omp.h` könyvtár implementálására és fordításkor a "-fopenmp" kapcsolóra.

```
#include <omp.h>

int main() {

    #pragma omp parallel
    while(1) {}

    return 0;
}
```



Ha 0%-os terhelést akarunk, akkor csökkenteni kell a feltételvizsgálat gyakoriságát. Vagyis a `sleep (seconds)` parancccsal várakozásra utasítjuk a processort.

```
#include <unistd.h>

int main() {

    while(1) sleep(1);

    return 0;
}
```

System Overview											
PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1	[idle]							10.0%			Tasks: 169, 690 thr; 1 running
2	[idle]							14.1%			Load average: 0.49 0.68 0.72
3	[idle]							10.6%			Uptime: 07:15:23
4	[idle]							9.9%			
Mem	[idle]							3.77G/7.70G			
Swp	[idle]							0K/3.81G			
<hr/>											
1070	gdm	20	0	111M	2724	12	S	0.0	0.0	0:00.00	(sd-pam)
1408	my	20	0	111M	2728	12	S	0.0	0.0	0:00.00	(sd-pam)
25849	my	20	0	4376	740	680	S	0.0	0.0	0:00.00	./inf sleep
11075	my	20	0	31388	5068	3568	S	0.0	0.1	0:00.02	/bin/bash
1069	gdm	20	0	77136	8316	6780	S	0.0	0.1	0:00.08	/lib/systemd/systemd --user
1407	my	20	0	77220	6464	5722	S	0.0	0.1	0:00.27	/lib/systemd/systemd-user

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

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

Megoldás videó:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{
    boolean Lefagy(Program P)
    {
        if (P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }
```

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

main(Input Q)
{
    Lefagy2(Q)
}

}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Nézzünk erre egy másik példát: Tegyük fel, hogy Lajos bárkiről megtudja mondani, hogy csődbe fog-e menni vagy sem (Lefagy fv.). A fia, Szabolcs ugyanazt csinálja, de ő már egy kicsit másképp csinálja (Lefagy2 fv.). Egyszer, amikor Szabolcs a saját pénzügyeit vizsgálta, válaszát apja válaszára építette. Ekkor 2 lehetőség merül fel. Egy, hogy ha Lajos azt mondja, hogy Szabolcs csődbe megy, akkor annak ellenére, hogy igazat ad apjának, nem megy csődbe, ha nemmel válszol, akkor viszont csődbe megy. Amint láthatjuk, ellentmondásba ütköztünk, vagyis ez a módszer így nem fog működni.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: bhax/theematic_tutorials/bhax_textbook/turing/valtcsera.c

Változók értékének felcserélésére leggyakrabban a rendezési algoritmusoknál van szükség. Ilyenkor a legegyszerűbb módszer jut az eszembe, miszerint veszek egy harmadik, úgymond segédváltozót, amiben ideiglenesen eltárolom az egyik változó értékét, hogy azt majd a másik változó kaphassa meg.

```
int c = a;
    a = b;
    b = c;
```

Ha segédváltozó nélkül szeretnénk megoldani a problémát, akkor kénytelenek vagyunk valamilyen műveleteket alkalmazni. Nem kell sokáig keresgálni, mivel az összeadás és kivonás tökéletesen megfelel erre a feladatra. Ehhez először összeadjuk őket az a változóba, majd abból kivonva a b-t megkapjuk először a b, majd az a értékét, ahogy azt a programkód szemlélteti:

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

2.4. Labdapattogás

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/turing/labdaIf.c](https://bhaxor.github.io/bhax/thematic_tutorials/bhax_textbook/turing/labdaIf.c) [bhax/thematic_tutorials/bhax_textbook/turing/labdaIfNelkul.c](https://bhaxor.github.io/bhax/thematic_tutorials/bhax_textbook/turing/labdaIfNelkul.c)

A feladat megoldásában tutorált: Tóth Balázs.

Ez a feladat sokkal egyszerűbb, mint az ember gondolná, legalábbis ha használhatunk if-eket. Csak annyit kell csinálni, hogy ciklusonként növeljük a labda pozícióját és ha elérjük az ablak kereteit, megfordítjuk a labda irányát. Ezután már csak meg kell jeleníteni a labdát.

```
#include <stdio.h>  
  
#define width 80  
#define height 24  
  
void draw(int x, int y){  
  
    for(int i = 0; i < y; i++) printf("\n");  
    for(int k = 0; k < x; k++) printf(" ");  
    printf("@\n");  
  
}  
  
int main(){  
  
    int x = 1;  
    int y = 1;  
    int a = 1;  
    int b = 1;  
  
    while(1){  
  
        system("clear");
```

```
if(x >= width || x <= 0) a *= -1;
if(y >= height || y <= 0) b *= -1;

draw(x, y);
x += a;
y += b;

usleep(15000);

}

return 0;
}
```

If-ek nélkül azonban már kicsit nehezebb a dolgunk, ugyanis egy kicsit gondolkodni kell. De ekkor se kell kétségbe esni, mert ugyebár a matematika a "barátunk" és minden nyújt számunkra egy bonyolultnak látszó képletet, hogy azt egyszer beötvözve örökre megszabaduljunk tőle. Most mi is ezt tesszük és még egy magyarázatot is csatolok hozzá.

```
draw(abs(width-(x++%(width*2))), abs(height-(y++%(height*2))));
```

Tegyük fel hogy az `x` értéke 1 és a `width` értéke pedig 5. Ha az 1-et elosztjuk maradékos osztással a `width` kétszeresével, akkor 1-et kapunk, amit kivonunk a szélességből. Ennek eredményeként a labda pattogása az ablak jobb alsó sarkából indul.

```
x = 1; width = 5;
|5 - ( 1 % ( 5 * 2 ) )| =
|5 - ( 1 % 10 )| =
|5 - 1| = 4
```

Ha az `x` eléri a 6-ot, az abszolútérték miatt a végeredmény növekedni kezd, egészen amíg el nem éri a `width` kétszerését, amikor újra csökkenni kezd és ez így megy körbe-körbe. A `++` az `x` után a számolások után megnöveli 1-el az `x`-et.

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. Használд ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/turing/bogomips.c

Mi is ez a BogoMips valójában? Megméri, hogy 1 másodperc alatt hányszor fut le a programunk egy ciklusa. Sokan abba a hibába esnek, hogy ez alapján hasonlítták össze a gépüket, hogy kié gyorsabb. Egyedül a linux bootolásakor érdemes odafigyelni rá, hogy a processzor és a cache rendben van-e.

A `ticks` változóba elmentjük a jelenlegi processzoridőt, majd a for ciklus egy bizonyos számú lefutása után a jelenlegi processzoridőből kivonjuk a `ticks` értékét, így megkapjuk hogy mennyi ideig tartott a ciklus x-szeri lefutása.

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

Ha ez az idő kisebb mint 1 másodperc, akkor legközelebb többször futtatjuk le a for ciklust, ellenben viszont némi átalakításokkal kiírjuk a gépünk BogoMips értékét.

```
Calibrating delay loop...ok - 690.00 BogoMIPS
```

A BogoMips while ciklus fejét használva mostmár könnyen megkapjuk szóhosszt. Konkrétan itt az történik, hogy az int típusú változó elején levő bitet ciklusonként eltoljuk és hozzáadunk egyet a szóhosszt tároló length változóhoz.

```
int x = 1;
int length = 0;

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

Az én gépemen a szóhossz mérete 32.

```
$ ./szohossz
A szóhossz mérete: 32
```

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/turing/helloGoogle.c

Most megvizsgáljuk a PageRank algoritmust, ami hatalmas fellendülést hozott a Google-nek.

Egy oldal PageRankját az határozza meg, hogy mennyi oldal hivatkozik rá. Mivel ez így magába nem lesz elég, mert az algoritmus azt is figyelembe veszi, hogy mekkora a PageRankjuk a rá hivatkozó oldalaknak. Például ha a Facebook hivatkozik a te oldaladra, akkor az sokkal többet jelent, mintha csak valami ismeretlen oldal hivatkozna rá, még akkor is ha több van belőlük.

A program elején létrehozunk egy kétdimenziós tömböt, ami meghatározza, hogy melyik mire hivatkozik. Ezután egy végtelen ciklusba bizonyos számításokkal addig finomítgatjuk az oldalak PR értékét, amíg már nincs értelme tovább számolatni.

```
#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 += (pagerank_temp[i] - pagerank[i]) * (pagerank_temp[i] - pagerank[i ↔
]);
return sqrt(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;  
}
```

2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

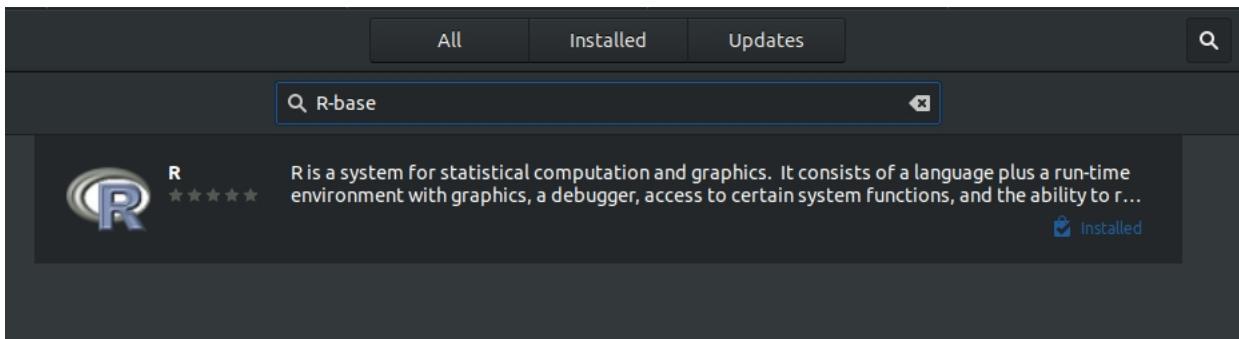
Megoldás videó: <https://youtu.be/xbYhp9G6VqQ>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/turing/brun.r

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com  
  
# This program is free software: you can redistribute it and/or modify  
# it under the terms of the GNU General Public License as published by  
# the Free Software Foundation, either version 3 of the License, or  
# (at your option) any later version.  
  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU General Public License for more details.  
  
# You should have received a copy of the GNU General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>  
  
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 szerint ha az ikerprímek reciprokát összeadjuk, akkor ez az összeg egy konkrét szám, ún. Brun-konstans. Hogy az ikerprímek száma véges vagy végtelen azt senki se tudja.

Az R nyelv használatához szükségünk lesz az R-base szoftverkörnyezet telepítésére, amit az Ubuntu Software programon keresztül tehetünk meg a legegyszerűbben.



Ezután a terminálba megnyitjuk az R környezetét (az **R** parancssal) és telepítjük a **matlab** csomagot, hogy kezelni tudjuk a prímeket. (Majd a **library(matlab)** parancssal betölthjük a csomagot)

```
my@my-GL62-6QD: ~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/t... - □ ×
File Edit View Search Terminal Help
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook$ R

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

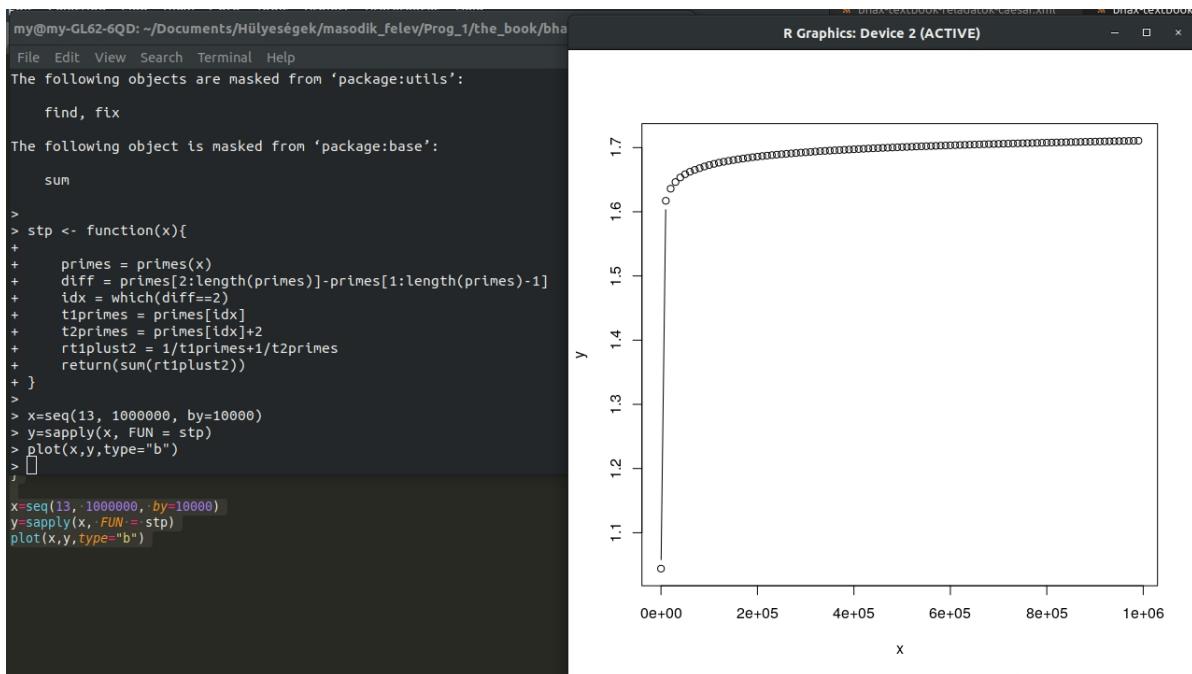
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

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

A `primes(meggi)` függvényteljesen lekérhetjük a prím számokat és a `primes[hanyadiktól : hanyadikig]` módon lekérhetjük annak tetszőleges intervallumát (Arra vigyázzunk, hogy amíg a `primes` először függvényként, másodszor viszont már azonosítóként szerepel). Az egymás mellett levő prímek egy-másból való kivonásával megkapjuk azok különbségét. Ahol a különbség 2, ott ikerprímekről beszélünk és a `which()` függvény visszaadja, hogy hanyadik helyen találjuk őket.

```
primes = primes(x)
diff = primes[2:length(primes)]-primes[1:length(primes)-1]
idx = which(diff==2)
```

Ezután 2 vektorba párhuzamosan eltároljuk ezeket az ikerprímeket és összeadjuk azok reciprokát. A `plot()` parancssal megjelenítjük az összeget és láthatjuk, hogy ez a függvény egy konkrét számhoz divergál.



2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

Megoldás forrása: [bhax_thematic_tutorials/bhax_textbook/turing/monty.r](https://bhaxor.github.io/bhax_thematic_tutorials/bhax_textbook/turing/monty.r)

A probléma szemléltetésként tegyük fel, hogy az előtted levő 3 láda közül az egyikbe kincs van a másik kettő üres. A játék elején választasz egyet, de mielőtt kinyithatnád, a játékvezető kinyitja az egyik üres látát (természetesen sose nem azt, amelyiket kivalasztottad). Ezután újra meggondolhatod, hogy a maradék 2 láda közül melyiket akarod kinyitni. Bárnmennyire is hihetetlennek tűnik, de nagyobb a valószínűsége, hogy a másikat választva megkapod a kincset, ellenben azzal, ha kitartasz az első döntésed mellett.

```

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

```

```
#   https://bhaxor.blog.hu/2019/01/03/ ←
#   erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan
#
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

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)
```

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky/toUnary.cpp

Míg a 10-es számrendszerben 10 számjegyünk van, az unárisban csak egy, az 1. Éppen ezért az átalakítás nagyon egyszerű lesz, hiszen csak annyi 1-est kell kiíratni, amennyi a megadott szám. Ehhez az átláthatóság kedvéért írunk egy függvényt, átadjuk neki a bekért számot és egy for ciklussal annyiszor íratunk ki egy 1-est, amennyi az bekért változó értéke.

```
#include <iostream>
#include <limits>

void DecimalToUnary(int x)
{
    for(int i=0; i<x; i++)
    {
        std::cout << "1";
    }
    std::cout << "\n";
}

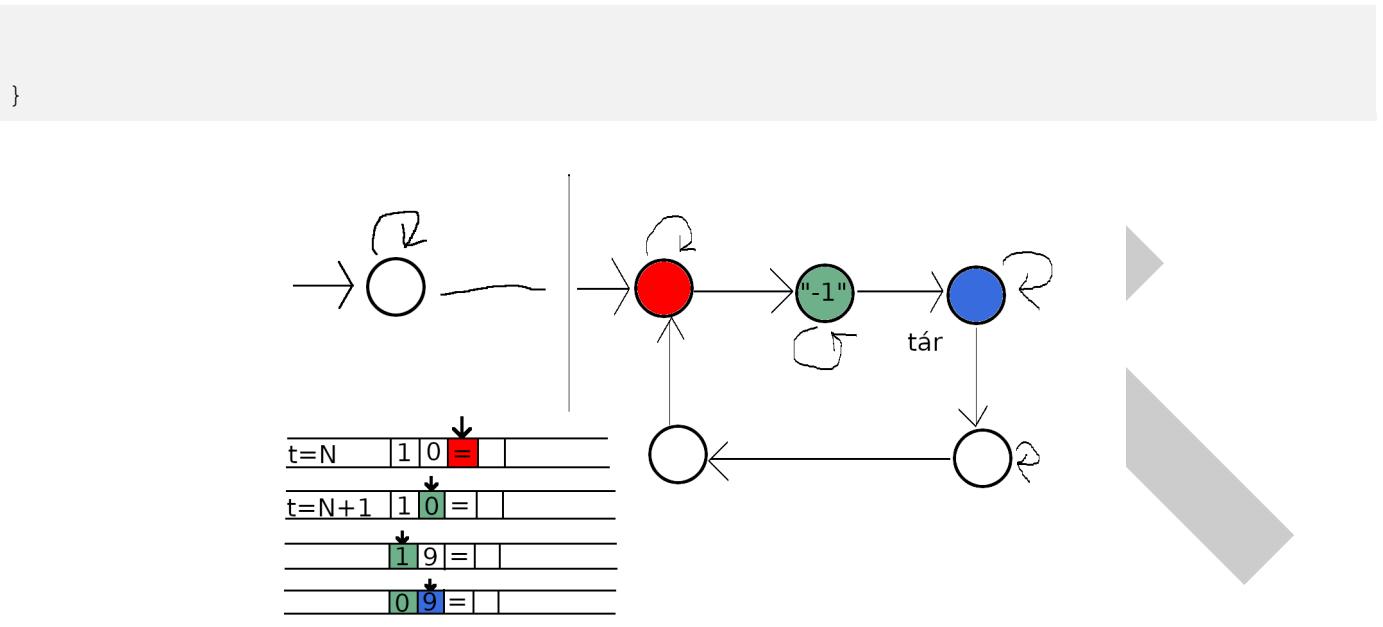
int main()
{
    int szam;

    std::cout << "Adj meg egy szamot:\n";

    std::cin >> szam;

    DecimalToUnary(szam);

    return 0;
}
```



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:

Környezetfüggő(hossznemcsökkentő)
 $P1XP2 \rightarrow P1QP2$, $P1, P2$ eleme $(VN \cup VT)^*$, X VN beli, Q VN U VT+ beli, kivéve S → üres, de akkor S nem lehet jobb oldali egyetlen szabályban sem.

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky/hivatk.c

<https://cs.wmich.edu/~gupta/teaching/cs4850/sumII06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm>

Először nézzük meg mi is az a BNF. A BNF (Backus-Naur Form) egy olyan nyelv, ami más nyelvek helyesírását írja le. Nézzünk meg ráegy egyszerű példát:

```
<egész szám> ::= <előjel><szám>
<előjel> ::= [-|+]
<szám> ::= <szamjegy>{<szamjegy>}
<szamjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

Ez alapján egy C utasítás BNF-ben így fog kinézni:

```
<statement> ::= <labeled_statement>
  | <compound_statement>
  | <expression_statement>
  | <selection_statement>
  | <iteration_statement>
  | <jump_statement>

<labeled_statement> ::= <identifier> : <statement>
  | case <constant_expression> : <statement>
  | default : <statement>

<compound_statement> ::= { <declaration_list> }
  | { <statement_list> }
  | { <declaration_list> <statement_list> }

<declaration_list> ::= <declaration>
  | <declaration> <declaration>

<statement_list> ::= <statement>
  | <statement_list> <statement>

<expression_statement> ::= <expression>

<selection_statement> ::= if ( <expression> ) <statement>
  | if ( <expression> ) <statement> else <statement>
  | switch ( <expression> ) <statement>

<iteration_statement> ::= while ( <expression> ) <statement>
  | do <statement> while ( <expression> ) ;
  | for ( <expression_statement> <expression_statement> ) <statement>
  | for ( <expression_statement> <expression_statement> <expression> ) <statement>

<jump_statement> ::= goto <identifier> ;
  | continue ;
  | break ;
  | return <expression> ;
```

A következő példával szemlélteti a C nyelv fejlődését:

```
int main(){
    // can't compile me with: -std=c89

    for(int i = 0; i < 10; i++)
        printf("Success!\n");

    return 0;
```

}

Ha a "-std=c89" kapcsolóval próbáljuk meg lefordíttatni ezt a programot, akkor hibába ütközünk.

```
hivatk.c: In function 'main':  
hivatk.c:5:2: error: C++ style comments are not allowed in ISO C90  
    // can't compile me with: -std=c89  
    ^  
hivatk.c:5:2: error: (this will be reported only once per input file)  
hivatk.c:7:2: error: 'for' loop initial declarations are only allowed in ←  
    C99 or C11 mode  
    for(int i = 0; i < 10; i++)  
    ^~~  
hivatk.c:7:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 ←  
    to compile your code
```

Tehát a c89-es szabvánnyal még nem lehetett dupla "/"-jeles kommentet írni és a `for` ciklus ciklusváltozóját a ciklus előtt kellett deklarálni.

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/chomsky/realnum.l](https://bhax.thematic_tutorials/bhax_textbook/chomsky/realnum.l)

Ez a lexer úgy működik, hogy az általunk megadott minták alapján átnézi az általunk megadott szöveget. Először megírjuk a mintákat ami alapján a lexer legenerálja a c programunkat és az fogja elvégezni a szövegelemzést.

A "%{" és "%}" jelek közé berakjuk a generálendő program elejére szánt sorokat, mint pl.: könyvtár importálás vagy deklaráció:

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

Ezután jönnek a definíciók. Jelen esetben egyedül a számjegy fogalmát kell definiálnunk, mivel ez már elegendő egy valós szám leírására.

```
digit [0-9]
```

Az első "%%" jel után megadjuk, hogy mit keresünk és hogy mit csinálunk, ha megtaláltuk a keresett szövegrészt.

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

Az utolsó rész - a második "%%" jel után - pedig a programunk, ahol elindítjuk a lexikális elemzést a `yylex()` függvény hívásával.

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

A `flex` csomag telepítése után már hozzá is láthatunk a fordításhoz. Először a lexerrel legeneráltatjuk a c fájlt, amit majd lefordítunk és futtatunk.

lex -o realnum.c realnum.l

gcc realnum.c -o realnum -lfl

3.5. I33t.I

Lexelj össze egy i33t cipher!

Megoldás video:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky/i33t.1

A 1337 (Leet) egy - az interneten használt - ábécé, ami a betűket számokkal vagy szimbólumokkal helyettesíti. Először hacker-ek és cracker-ek használták, de később soknak megtetszett és a felhasználóneveiket ilyen módon módosítva kezdték el használni.

A változatosság kedvéért minden betűhöz hozzárendelünk 4 laternatívát és egy random generált szám fogja megmondani, hogy melyiket válasszuk. Ezt úgy oldjuk meg, hogy a struct segítségével létrehozunk egy típust, ami egy karaktert és egy 4 elemű tömböt tartalmaz, majd ebből egy kétdimenziós tömböt készítünk.

```
struct cipher {
    char c;
    char *leet[4];
} i337d1c7 [] = {

    {'a', {"4", "4", "@", "/-\\"}},
    {'b', {"|o", "8", "|3", "|"}},
    {'c', {"c", "(", "<", "{"}},
    ...
    {'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);  
  
}
```

Amint azt láthatjuk, az "a" betűt felcserélte egy "4"-re és az "e"-t "3"-re (ezek a leggyakrabban használt átalakítások).

A feladat teljesítve.
4 f3l/-\d4t t3lj3sítv3.
A feladat teljesítve.
4 f3l4d4t t3l_j3s1t\3.

A fordításhoz és futtatáshoz ugyanúgy kell eljárni, mint az előző feladatban.

3.6. A források olvasása

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

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

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

Bugok

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

i.

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

Itt viszont akkor kezeli a jelkezelő a SIGINT jelet, ha a SIGINT jel nem lett figyelmen kívül hagyva.

```
/home/my/Documents/H\377\3771yes\377\377gek/masodik_felev/Prog_1/ ←
    the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/forrasolv.c: ←
        (in function main)
/home/my/Documents/H\377\3771yes\377\377gek/masodik_felev/Prog_1/ ←
    the_book/bhax/t
    thematic_tutorials/bhax_textbook/chomsky/forrasolv.c:10:8:
        Test expression for while not boolean, type int: 1
        Test expression type is not boolean or int. (Use -predboolint to ←
            inhibit
            warning)
/home/my/Documents/H\377\3771yes\377\377gek/masodik_felev/Prog_1/ ←
    the_book/bhax/t
    thematic_tutorials/bhax_textbook/chomsky/forrasolv.c:14:4:
        Return value (type [function (int) returns void]) ignored:
        signal(SIGINT, j...
        Result returned by function call is not used. If this is intended, ←
            can cast
            result to (void) to eliminate message. (Use -retvalother to inhibit ←
                warning)
/home/my/Documents/H\377\3771yes\377\377gek/masodik_felev/Prog_1/ ←
    the_book/bhax/t
    thematic_tutorials/bhax_textbook/chomsky/forrasolv.c:4:6:
        Function exported but not used outside forrasolv: jelkezelo
        A declaration is exported, but not used outside this module. ←
            Declaration can
            use static qualifier. (Use -exportlocal to inhibit warning)
        /home/my/Documents/H\377\3771yes\377\377gek/masodik_felev/Prog_1/ ←
            the_book/bha
            x/thematic_tutorials/bhax_textbook/chomsky/forrasolv.c:6:1:
                Definition of jelkezelo
```

Finished checking --- 3 code warnings

ii.

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

Ez egy `for` ciklus, ami 5-ször fog lefutni és minden egyes alkalommal 1-el növeli az `i` értékét preorder alakban.

iii.

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

Ez egy ugyanúgy 5-ször lefutó `for` ciklus, ami mostmár postorder alakban növeli a ciklusváltozót, de ettől függetlenül a ciklus ugyanannyiszor fog lefutni.

iv.

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

Ez nem úgy fog működni, ahogy azt szeretnénk. A **splint** parancssal megvizsgálva a programkódot rájövünk hogy miért. A program futása során nem egyértelmű, hogy mikor lesz növelve az `i` értéke, így például az `i` értéke a tömb nem várt pozíójára kerülhet.

```
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/for52.c: (in ←  
function main)  
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/t  
hematic_tutorials/bhax_textbook/chomsky/for52.c:6:31:  
Expression has undefined behavior (left operand uses i, modified ←  
by right  
operand): tomb[i] = i++  
Code has unspecified behavior. Order of evaluation of function ←  
parameters or  
subexpressions is not defined, so if a value is used and modified in  
different places not separated by a sequence point constraining ←  
evaluation  
order, then the result of the expression is unspecified. (Use - ←  
evalorder to  
inhibit warning)
```

Finished checking --- 1 code warning

v.

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

Ez a ciklus **minimum** `n`-szer le fog futni. Hogy **maximum** hányszor az attól függ, hogy mi lesz a `*d++` visszatérési értéke.

Abban az esetben, ha a `d` és az `s` `int` típusú pointerek, akkor a **splint** hibát jelez, mivel az és jelek jobb oldalán nem logikai értéket kapunk.

```
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/for53.c: (in ←  
function main)  
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/t
```

```
hematic_tutorials/bhax_textbook/chomsky/for53.c:9:18:  
Right operand of && is non-boolean (int): i < n && (*d++ = *s++)  
The operand of a boolean operator is not a boolean. Use +ptrnegate ←  
to allow !  
to be used on pointers. (Use -boolops to inhibit warning)
```

Finished checking --- 1 code warning

vi.

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

Ez a kód úgyanazt a hibát követi el, mint a tömbös, ezen felül amikor az egyik f függvénynek átadjuk az a-t és növeljük, akkor az módosítja a másik f függvénynek átadott értéket is.

```
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/thematic_tutorials/bhax_textbook/chomsky/print1.c: ( ←  
in function main)  
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/t  
hematic_tutorials/bhax_textbook/chomsky/print1.c:11:23:  
Argument 2 modifies a, used by argument 1 (order of evaluation of ←  
actual  
parameters is undefined): f(a, ++a)  
Code has unspecified behavior. Order of evaluation of function ←  
parameters or  
subexpressions is not defined, so if a value is used and modified in  
different places not separated by a sequence point constraining ←  
evaluation  
order, then the result of the expression is unspecified. (Use - ←  
evalorder to  
inhibit warning)  
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/t  
hematic_tutorials/bhax_textbook/chomsky/print1.c:11:31:  
Argument 1 modifies a, used by argument 2 (order of evaluation of ←  
actual  
parameters is undefined): f(++a, a)  
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/t  
hematic_tutorials/bhax_textbook/chomsky/print1.c:11:18:  
Argument 2 modifies a, used by argument 3 (order of evaluation of ←  
actual  
parameters is undefined): printf("%d %d", f(a, ++a), f(++a, a))  
/home/my/Documents/H\377\377lyes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/t  
hematic_tutorials/bhax_textbook/chomsky/print1.c:11:29:  
Argument 3 modifies a, used by argument 2 (order of evaluation of ←  
actual  
parameters is undefined): printf("%d %d", f(a, ++a), f(++a, a))
```

```
/home/my/Documents/H\377\3771yes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bhax/t  
hematic_tutorials/bhax_textbook/chomsky/print1.c:3:5:  
Function exported but not used outside print1: f  
A declaration is exported, but not used outside this module. ←  
Declaration can  
use static qualifier. (Use -exportlocal to inhibit warning)  
/home/my/Documents/H\377\3771yes\377\377gek/masodik_felev/Prog_1/ ←  
the_book/bha  
x/thematic_tutorials/bhax_textbook/chomsky/print1.c:5:1: Definition ←  
of f  
  
Finished checking --- 5 code warnings
```

vii.

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

Tegyük fel, hogy az `f` függvény visszatérési értéke az argumentum négyzete. Ekkor először kiírja az a négyzetét, majd magát az `f`-t.

viii.

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

Ha az `f` függvény 1-el nagyobb értéket ad vissza, mint az az érték amire az argumentum mutat, akkor először kiírjuk az `a+1`-et, majd az a értékét.

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/chomsky/forrasolv.c](#)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for51.c](#)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for52.c](#)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for53.c](#)
[bhax/thematic_tutorials/bhax_textbook/chomsky/for51.c](#)

Megoldás videó:

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/chomsky/matlog.tex](https://bhax.thematic_tutorials/bhax_textbook/chomsky/matlog.tex) [bhax/thematic_tutorials/bhax_textbook/chomsky/logikus.tex](https://bhax.thematic_tutorials/bhax_textbook/chomsky/logikus.tex)

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

A szöveg megértéséhez szükségesek a következő információk: A `\forall` a univerzális kvantornak felel meg, a `\exists` pedig az egzisztenciális kvantornak. A `\wedge` az és logikai jelet, a `\supset` pedig a implikáció logikai jelet szimbolizálja.

```
\item
Bármely x szám esetén létezik nála nagyobb y prím szám.\\
$(\forall x \exists y ((x < y) \wedge (y \text{ prim})) )\$

\item
Bármely x szám esetén léteznek nála nagyobb ikerprímek.\\
$(\forall x \exists y ((x < y) \wedge (y \text{ prim}) \wedge (\text{SSy} \text{ prim}))) \$

\item
Létezik olyan y, hogy minden prím szám kisebb nála.\\
$(\exists y \forall x (x \text{ prim}) \supset (x < y)) \$

\item
Létezik olyan y, aminél nincs nagyobb prím szám.\\
$(\exists y \forall x (y < x) \supset \neg (x \text{ prim})) \$
```

A matematikai logikai (matlog) nyelv a beszélt nyelvnél pontosabb kifejező eszköz. A beszélt nyelv szavakból és mondatokból épül, még a matlog nyelv termekből és formulákból. A termek változókból és függvényekből épülnek. Term például az az x változó, ami befutotta az embereket. A formulák termekből és prédkátumokból épülnek. Formula például a Szeret(x, y) két változós prédkátum. Illetve a logikai jelekkel a formulákból további formulák építhetők fel.

```
% http://detexify.kirelabs.org/classify.html

\documentclass{article}

% magyar betűk ebben a forrásban. hogy ne kelljen \'a, hanem elég legyen az ←
% á
\usepackage[utf8]{inputenc}
% legyen pl. magyar elválasztása a sorvégén a szavaknak:
\usepackage[magyar]{babel}

\usepackage{amsmath}
\usepackage{amssymb}
\usepackage{url}
\usepackage{listings}

\title{K\"onyvolvas\'as}
\date{\today}
\author{N\'andi B\'atfai \and Gr\'eta B\'atfai \and Matyi B\'atfai \and ←
Norbi B\'atfai }
```

```
\begin{document}  
  
\maketitle  
  
\section{MatLog}
```

A Péter Rózsa: Játék a végtelennel című könyvével kezdtünk. A 261. oldal 3. ← bekezdésétől kb. a 265. oldalig jeleltük le:

```
\begin{itemize}  
\item  
Bevezetés a matematikai logikába 1., \url{https://youtu.be/j1IBkFO3UNk}  
\end{itemize}
```

Majd Dragálin Albert: Bevezetés a matematikai logikába című könyvének ← jelöléseivel (használva közben a \url{http://detexify.kirelabs.org/ ← classify.html} lapot a legmegfelelőbb logikai szimbólumok megtalálásához ←) próbálkoztunk latex-ben:

```
\begin{itemize}  
\item  
Második TeXelés, \url{https://youtu.be/qly_9ECViBM}  
\end{itemize}
```

Aztán a Dragálin könyv 44. oldalával folytattuk, a természetes nyelvű ← mondatok matlog átíratával.

```
\subsection{Természetes nyelvű mondatok matlog átírata}
```

Dragálin Albert: Bevezetés a matematikai logikába, 44. oldal.

```
\subsubsection{A könyv példái}
```

Első stream.

```
\begin{itemize}  
\item  
Minden hal, kivéve a cápát, szereti a gyerekeket. \\  
$\forall x (\neg \text{Cápa}(x) \supset \text{SzeretiGyerekek}(x))$  
  
\item  
Minden ember, kivéve a pacifistákat, szereti a fegyvereket. \\  
$\forall x (\neg \text{pacifista}(x) \supset \text{Szeretifegyver}(x))$  
  
\item  
Péter akkor szellemes, ha részeg\\  
\$ \text{PéterRészeg} \supset \text{PéterSzellemes} \$  
  
\item  
Péter \textbf{csak} akkor szellemes, ha részeg.\\  
\$ \text{PéterSzellemes} \supset \text{PéterRészeg} \$
```

```
\item
Péter ha részeg, akkor szellemes.\\
\$ \text{PéterRészeg} \supset \text{PéterSzellemes}\$
```

```
\item
Péter \textbf{akkor és csak} akkor szellemes, ha részeg.\\
\$ ( \text{PéterRészeg} \supset \text{PéterSzellemes}) \wedge ( \text{ } \leftarrow
    \text{PéterSzellemes} \supset \text{PéterRészeg}) \$
```

```
\end{itemize}
```

Második stream.

```
\begin{itemize}
\item
Minden ember szeret valakit. \\
\$ \forall x \exists y \text{Szeret}(x, y) \$
```

```
\item
Valakit minden ember szeret. \\
\$ \exists y \forall x \text{Szeret}(x, y) \$ \text{vegyük észre, hogy mi a } \leftarrow
    \text{különbség az előző formulával összehasonlítva: a kvantokrok sorrendje } \leftarrow
    \text{számít!} \$
```

```
\item
Senki nem szeret mindenkit. \\
\$ \neg \exists x \forall y \text{Szeret}(x, y) \$
```

```
\item
Valaki szeret mindenkit. \\
\$ \exists x \forall y \text{Szeret}(x, y) \$
```

```
\item
Valaki senkit sem szeret. \\
\$ \exists x \forall y \neg \text{Szeret}(x, y) \$
```

```
\item
Mindenki szeret valakit vagy valaki szeret mindenkit. \\
\$ \forall x \exists y \text{Szeret}(x, y) \vee \exists x \forall y \text{Szeret}(x, y) \$
```

```
\end{itemize}
```

```
\subsection{A matlog nyelv}
```

```
\item
\$ (A \wedge B) \$ is az, olvasva "A és B", \LaTeX -ben leírva: \verb!(A \leftarrow
    wedge B)! $!
```

```
\item
$(A \vee B)$ is az, olvasva "A vagy B", \LaTeX -ben leírva: \verb!$(A \vee \leftrightarrow
B)$!
\item
$(A \supset B)$ is az, olvasva "A-ból következik B", \LaTeX -ben leírva: \leftrightarrow
verb!$(A \supset B)$!
\item
$\neg A$ is az, olvasva "Nem A", \LaTeX -ben leírva: \verb!$\neg A$!
\item
$\exists x A$ is az, olvasva "Van olyan x, hogy A", \LaTeX -ben leírva: \leftrightarrow
verb!$\exists x A$!
\item
$\forall x A$ is az, olvasva "Minden x-re A", \LaTeX -ben leírva: \verb!$\forall \leftrightarrow
forall x A$!
\end{itemize}
```

\subsubsection{Az Ar matlog nyelv}

A \$0\$, \$x\$, \$y\$, \$z \dots\$ változók természetes szám típusúak (a 0 egy konstans, azaz nem változó változó, a 0 természetes szám jele).

Az \$SS\$, \$+\$, \$\cdot\$ függvények.

A termek építésének kódja:

```
\begin{enumerate}
\item minden változó neve term, például $0$, $x$
\item ha $t$ term, akkor $St$ is az, például $Sx$, $SSS0$, $S(0+x)$
\item ha $t$, $v$ termek, akkor $(t+v)$ és $(t \cdot v)$ is termek, például $(x \leftrightarrow
+0)$, $(Sx+SSS0)$.
\end{enumerate}
```

Formulák építéséhez egyetlen prédikátum van, az \$\text{Egyenlő}(t, v)\$, amit olvashatóbban így írunk \$(t=v)\$.

Például az \$(x+0) = SSS0\$ egy formula. Ez a formula az \$x=3\$ kiértékelés mellett igaz egyébként hamis.

Mikor igaz az \$(x+y = Sx+x)\$ formula? Mikor igaz az \$(x = SSy)\$ formula?

\paragraph{Alapozzuk meg a következő formulákat!}

Lásd a \url{https://www.twitch.tv/nbatfai} csatornán közvetített élő adások archívumát a \url{https://www.youtube.com/c/nbatfai} YTB csatornán, konkrétan például: \url{https://youtu.be/ZexiPy3ZxsA} és \url{https://youtu.be/DUGPR1Xk_2w}.

```
\begin{itemize}
\item
```

```
$ (x \le y) \leftrightharpoons \exists z (z+x=y)$\item$ (x \neq y) \leftrightharpoons \neg (x=y)$\item$ (x < y) \leftrightharpoons \exists z (z+x=y) \wedge \neg (x=y)$\item$ (x < y) \leftrightharpoons (x \le y) \wedge (x \neq y)$\item$ (x \vee y) \leftrightharpoons \exists z (z \cdot x=y) \wedge (x \neq 0) \leftrightarrow $\$ (x \text{ páros}) \leftrightharpoons (\exists 0 \vee x) \$\item$ (\infty \text{ sok szám van}) \leftrightharpoons (\forall x \exists y (x < y \leftrightarrow )) \$\item$ (\text{véges sok szám van}) \leftrightharpoons (\exists y \forall x (x < y) \leftrightarrow ) \$\item$ (x \text{ prim}) \leftrightharpoons (\forall z (z \vee x) \supset (z = x \vee z=S0)) \wedge (\forall z (z \neq S0) \wedge (\forall y (y \text{ prim}) \wedge (S0 \wedge y))) \$\item$ (\infty \text{ sok prímszám van}) \leftrightharpoons (\forall x \exists y ((x < y) \wedge (y \text{ prim}))) \$\item$ (\infty \text{ sok iker-prímszám van}) \leftrightharpoons (\forall x \exists y ((x < y) \wedge (y \text{ prim})) \wedge (\forall z (z < y) \wedge (z \neq x) \wedge (z \neq y))) \$\item$ (\text{véges sok prímszám van}) \leftrightharpoons (\exists y \forall x (x < y) \supset (x \text{ prim})) \$\item$ (\text{véges sok prímszám van}) \leftrightharpoons (\exists y \forall x (y < x) \supset \neg (x \text{ prim})) \$\end{itemize}
```

Olvassuk el a most feldolgozott, Dragálin könyv 15–19 oldalait!

```
\paragraph{"Hol a tagadás lábat megveti"}
```

Felmerült, hogy a $\$(\text{v\'egek sok pr\'imsz\'am van})\$$ kifejezhető lenne a $\$(\leftarrow \infty \text{ Sok pr\'imsz\'am van})\$$ tagadásaként, azaz
 $\$(\text{v\'egek sok pr\'imsz\'am van}) \leftarrow \neg(\infty \text{ Sok pr\'imsz\'am van})\$$

Tagadjuk az egyik végességet kifejező formulánkat!

```
\begin{align*}
\neg \exists y \forall x & ((y < x) \supset \neg(x \text{ prim})) \\
\forall y \neg \forall x & & & \supset \neg(x \text{ prim})) \\
\forall y \exists x \neg & & & & \supset \neg(x \text{ prim})) \\
\forall y \exists x & & & & \wedge \neg \neg(x \text{ prim})) \\
\forall y \exists x & & & & \wedge (x \text{ prim})) \\
\end{align*}
```

az eredmény éppen a végtelenséges formulánk volt!

Ismételjük meg ezt a számítást a másik formulával!

```
\begin{align*}
\neg \exists y \forall x ((x \text{ prim}) \supset (x < y)) \\
\end{align*}
```

```
\end{document}
```

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referencia
- egészek tömbje
- egészek tömbjének referencia (nem az első elemé)
- egészre mutató mutatók tömbje
- egészre mutató mutatót visszaadó függvény

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

Mit vezetnek be a programba a következő nevek?

- `int a;`

Ez egy int típusú változó lesz.
- `int *b = &a;`

Ez egy int típusú mutató, aminek átadjuk az a változó referenciaértékét (tehát a memóriacímet).
- `int &r = a;`

Egy egész referenciája.
- `int c[5];`

Ez egy int típusú, 5 elemű egydimenziós tömb.
- `int (&tr)[5] = c;`

Egészek tömgjének referenciája.
- `int *d[5];`

Ez egy 5 elemű pointertömb, amely elemei int típusú értékekre mutatnak.
- `int *h();`

Egy olyan függvény, ami egészre mutató pointert ad vissza.
- `int *(*l)();`

Egy olyan függvényre mutató pointer, ami egészre mutató pointert ad vissza.
- `int (*v(int c))(int a, int b)`

Egészet visszaadó és két egészet kapó függvényre mutató pointert visszaadó, egészet kapó függvény.
- `int (*(*z)(int))(int, int);`

Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató pointert visszaadó, egészet kapó függvény.

```
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook/chomsky$ g++ deklar.cpp -o deklar
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook/chomsky$ cd ..
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook$ 
```

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/chomsky/deklar.cpp](#)

DRAFT

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 tárban!

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/caesar/haromszog.c](https://bhax.org/thematic_tutorials/bhax_textbook/caesar/haromszog.c)

Egy 5 soros háromszögmátrixot fogunk létrehozni. Ezt egy `double` típusú pointerekre mutató pointerrel fogjuk megvalósítani. A `malloc(size)` parancssal lefoglaljuk a sorok mutatói számára a memóriát. Hogy tudjuk mennyi bájtot foglaljunk le, a `sizeof()` függvényel lekérjük egy `double` méretét és megszorozzuk a sorok számával és a visszaadott pointert `double **` típusúra kényszerítjük. Megvizsgáljuk, hogy nem e kapott valamelyik elem `NULL` értéket, mert az azt jelentené, hogy nincs elég memória és kilép a programból.

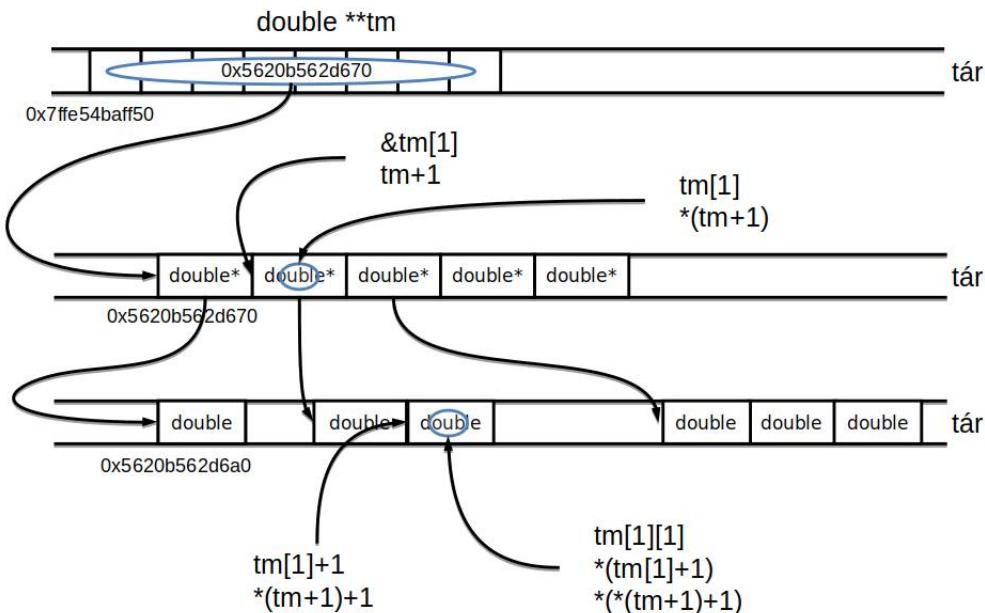
```
int nr = 5;
double **tm;

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

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

A for ciklussal lefoglaljuk az egyre növekvő elemszámú sorok elemeinek a memóriáját, amit szintén megvizsgálunk a Null értékkal.

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



Az elemek értékeinek módosítgatási után a `free()` eljárás használatával felszabadítjuk először az elemekre, majd a sorokra lefoglalt memóriát.

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

free (tm);
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/caesar/e.c

A titkosítást EXOR műveettel fogjuk végrehajtani egy általunk megadott kulcs és a titkosítandó szöveg között.

A `kulcs_meret` változóban eltároljuk a futtatáskor átadott második argumentum (tehát a kulcs) hosszát. Majd a `strncpy(hova, honnan, méret)` függvénytellyel a `kulcs` változóba másoljuk a kulcsot.

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

Folyamatosan beolvassuk a szöveget és a kulcs egyes elemeivel végrehajtjuk rajta a műveletet, miközben kiírjuk a titkosított szöveget a standard outputra.

```
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);

}
```

A futtatáskor megadjuk a kulcsot a titkosításhoz és a kimenetet átirányítjuk a titkos.szoveg állományba:

./e 56789012 <tiszta.txt > titkos.szoveg

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/caesar/ExorTitkosító.java

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcssSzöveg,
                          java.io.InputStream bejövőCsatorna,
                          java.io.OutputStream kimenőCsatorna)
                          throws java.io.IOException {

        byte[] kulcs = kulcssSzöveg.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();  
  
    }  
  
}
```

A programkód átírása során néhány változó deklarálása feleslegessé vált. Ilyen például a MAX_KULCS állandó. Míg a c verzióban előre deklaráltuk a kulcs tömböt és csak azután adtunk neki értéket, így volt haszná. Itt viszont a kulcs tömb megfelelő méretűre deklarálódik. A BUFFER_MERET változót is elhagytuk, bár az az előző kódban sem volt szükséges.

```
byte [] kulcs = kulcsSzöveg.getBytes();  
byte [] buffer = new byte[256];
```

A kulcs_meret változót is elhagytuk, mivel azt bármikor lekérhetjük, mint egy tömb méretét:

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

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/caesar/t.c

bhax/thematic_tutorials/bhax_textbook/caesar/t_par.c

Hiába tudjuk milyen módon lett titkosítva a szöveg, ahhoz hogy dekódoljuk, meg kell találnunk a kulcsot. Ezt úgy fogjuk csinálni, hogy szépen sorba végigmegyünk az összes kulcson és közbe keresünk néhány "kulcsszót", amelyek gyakran előfordulhatnak, ezért ha megegyeznek a dekódolt szóval, akkor kiíratjuk ezt a kulcsot és a szöveget, mint lehetséges megoldás és tovább keresünk. Erre azért van szükség, mert megeshet, hogy a kulcsszavak helyesen lettek dekódolva, de ettől még sok karakter eltér a várt eredménytől.

A kulcsok legenerálása és kipróbálása annyi egymásba ágyazott for ciklust igényel, amekkora a kulcs mérete. Ez esetben tudjuk, hogy 8 karakter hosszú.

```
// 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]\nTiszta ←
                                szoveg: [%s]\n",
                                ii, ji, ki, li, mi, ni, oi, pi, ←
                                titkos);

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

A következő 2 függvény fogja nekünk eldönteni, hogy rátaláltunk-e a keresett kulcsra. Ha az átlagos szóhossz 6 és 9 közé esik, valamit a dekódolt szövegben megtalálható a "hogy", a "nem", az "az" és a "ha" szavak, akkor kiírjuk a dekódolt szöveget.

```
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 szöveg 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");  
  
}
```

Mivel a kulcsok egyesével való kipróbálgatása elég időigényes, a for ciklusokat párhuzamosítjuk a végtelen ciklusoknál használt módon.

```
// 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);  
                                }  
    }
```

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

Tanulságok, tapasztalatok, magyarázat...

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com  
#  
# This program is free software: you can redistribute it and/or modify
```

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

library(neuralnet)

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

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

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

plot(nn.or)

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

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

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])
```

4.6. Hiba-visszaterjesztéses perceptron

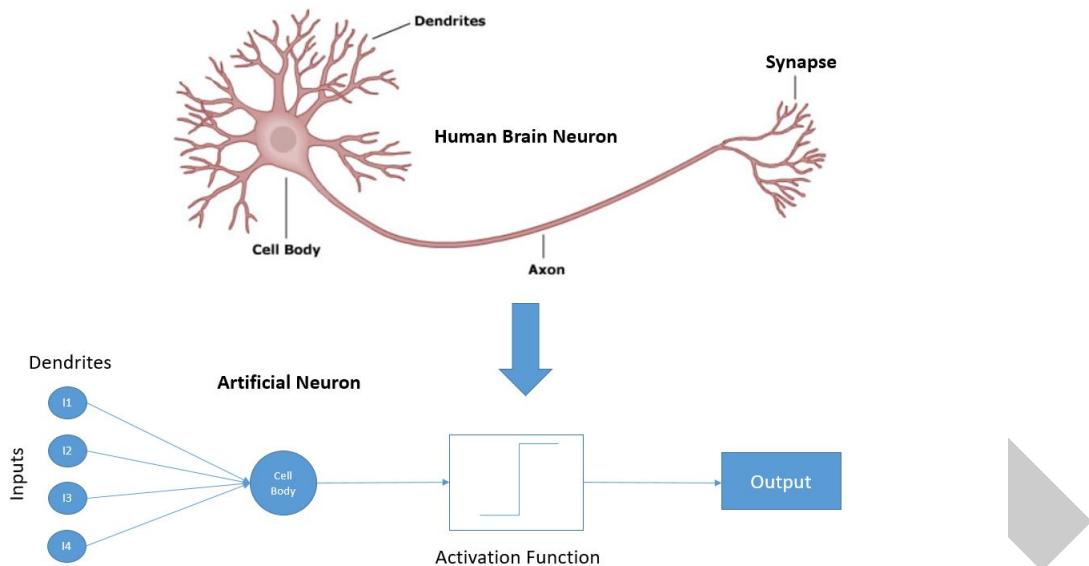
C++

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/caesar/percr.hpp](http://bhax.thematic_tutorials/bhax_textbook/caesar/percr.hpp)

http://www.cs.ubbcluj.ro/~csatol/mestint/pdfs/neur_halo_alap.pdf

A feladat megoldásában tutorált engem: Szilágyi Csaba.



Az agyi neuronok bemenetei a dendritek, amelyek más neuronokhoz csatlakoznak. A tényleges feldolgozást a neuron tulajdonképpeni sejtmagja végzi, és az eredményt az axonon keresztül juttatja el a többi neuronhoz. Az axon végezősei több másik neuron dendritjeihez csatlakozhatnak.

Egy neurális hálózat több neuronból állnak, amelyek rétegekbe vannak sorolva:

- Bemeneti réteg: azok a neuronok találhatók itt, amelyek a bemeneti jel továbbítását végzik a hálózat felé. A legtöbb esetben nem jelöljük őket külön.
- Rejtett réteg: a tulajdonképpeni feldolgozást végző neuronok tartoznak ide. Egy hálózaton belül több rejtett réteg is lehet.
- Kimeneti réteg: az itt található neuronok a külvilág felé továbbítják az információt. A feladatuk ugyanaz, mint a rejtett rétegbeli neuronoké.

A neurális hálózatok újrafelfedezése akkor kezdődött, amikor a 80-as évek elején felfedezték a tanítására alkalmas hibavisszaterjesztéses algoritmust. A többrétegű perceptron hálózatokban perceptronokat kötünk össze.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

telepíteni kell a qt-t!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/mandelbrot/mandelpngt.cpp

<https://hirmagazin.sulinet.hu/HU/pedagogia/mandelbrot-halmaz>

Ez a halmaz Benoit B. Mandelbrot, lengyel származású matematikusról, a megalkotójától kapta a nevét. (A képe e cikk ajánlója mellett látható.) Hogyan építhető fel ez a halmaz? A komplex számsíkon veszünk egy C pontot.

Erre képezzük a következő sorozatot:

$$Z_0 := C$$

$$Z_{i+1} := Z_i^2 + C$$

Ezzel az egyszerű rekurzióval definiált sorozatról be lehet bizonyítani, hogy bizonyos C számok választása esetében végtelenbe tart, vagy más C-k esetében pedig nullához tart. Más eset nem lehetséges. A Mandelbrot-halmaznak azok és csak azok a C komplex számok az elemei, amelyek esetében a fenti sorozat nullához tart. Ábrázolás esetében ezeket általában feketére szokták festeni, míg a többi pontot attól függően, hogy "milyen gyorsan" tart a végtelenbe.

Az első programunk a CPUt használva számolja ki a Mandelbrot halmaz elemeit. A fordításhoz szükségünk lesz néhány kapcsolóra: **g++ mandelpngt.cpp -o mandelpngt -lpng16 -O3**

```
// mandelpngt.cpp
// 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 mandelbrot/mandelpngt.cpp
```

```
// (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.  
//mandelbrot/mandelpngt.cppmandelbrot/mandelpngt.cpp  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <https://www.gnu.org/licenses/>.  
//  
// Version history  
//mandelbrot/mandelpngt.cpp  
// Mandelbrot png  
// Programozó Páternoszter/PARP  
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←  
_01_parhuzamos_prog_linux  
//  
// https://youtu.be/gvaqijHlRUs  
//  
#include <iostream>  
#include "png++/png.hpp"  
#include <sys/times.h>  
  
#define MERET 600  
#define ITER_HAT 32000  
  
void  
mandel (int kepadat[MERET] [MERET]) {  
  
    // Mérünk időt (PP 64)mandelbrot/mandelpngt.cppmandelbrot/mandelpngt.cpp  
    // Mérünk időt (PP 66)  
    clock_t delta = clock ();  
    struct tms tmsbuf1, tmsbuf2;  
    times (&tmsbuf1);  
  
    // számítás adatai  
    float a = -2.0, b = .7, c = -1.35, d = 1.35;  
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;  
  
    // a számítás  
    float dx = (b - a) / szelesseg;  
    float dy = (d - c) / magassag;  
    float reC, imC, rez, imZ, ujrez, ujimZ;  
    // Hány iterációt csináltunk?  
    int iteracio = 0;  
    // Végigzongorázzuk a szélesség x magasság rátcsot:  
    for (int j = 0; j < magassag; ++j)  
    {  
        //sor = j;
```

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

        ++iteracio;
    }

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

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

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

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

```
}

int kepadat[MERET][MERET];

mandel(kepadat);

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

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

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

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/mandelbrot/mandelpngt.cpp

A feladat ismét a Mandelbrot-halmaz kirajzoltatása egy fájlba, most viszont a komplex számok használatához complex típust fogunk használni. Egy ilyen típus deklarálásakor először megadjuk a valós, majd az imaginárius értékét a komplex számnak.

```
std::complex<double> c (valos, imaginarius);
```

A fordításhoz és futtatáshoz szükséges parancsok a forráskód elején szerepelnek. Ez a verzió már sokkal több lehetőséget tár elénk, mivel megadhatjuk a számolás paramétereit, mint pl. a kimeneti kép mérete vagy az n értéke.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ←
// -0.01947381057309366392260585598705802112818 ←
// -0.0194738105725413418456426484226540196687 ←
// 0.7985057569338268601555341774655971676111 ←
// 0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ←
// 0.4127655418209589255340574709407519549131 ←
// 0.4127655418245818053080142817634623497725 ←
// 0.2135387051768746491386963270997512154281 ←
// 0.2135387051804975289126531379224616102874
// Nyomtatas:
// 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;  
}
```

5.3. Biomorfok

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/mandelbrot/3.1.3.cpp
[https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_31_fraktalok/a_juliahalma...scorml](https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_31_fraktalok/a_juliahalma...)

Nem véletlen a többes szám, mivel Julia halmazból végtelen sok van. Hasonlóan indulunk, mint a Mandelbrot halmaznál, veszünk egy $f(z)=z^2+c$ kvadratikus komplex együtthatós polinomot. Most viszont a c konstansot rögzítjük és különböző x értékekre vizsgáljuk a pályákat. Hasonlóan a Mandelbrot halmazhoz, itt is a nem divergens pályájú pontok fognak a keresett halmazhoz tartozni. Mivel a c értéket bárhogy választhatjuk, végtelen sok Julia halmazt kapunk.

Ezekre a biomorfokra Pickover bukkant rá, amikor egy Júlia-halmazt akart kirajzolatni, viszont az iteráció feltételvizsgálatát hibásan írta meg: minden egyes pont kirajzolása előtt megvizsgálta, hogy a komplex szám valós vagy képzetes része kisebb-e, mint az iterációs határ.

```
// 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 -1 --line-numbers=1 --left-footer=" ←  
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←  
color  
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf  
//  
// BHAX Biomorphs
```

```
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/Vol19\_Iss5\_2305--2315\_Biomorphs\_via\_modified\_iterations.pdf
//

#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 szelesség magasság n a b c ←
                  d reC imC R" << std::endl;
    return -1;
}

png::image<png::rgb_pixel> kep ( szelesség, magasság );

double dx = ( xmax - xmin ) / szelesség;
double dy = ( ymax - ymin ) / magasság;

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

std::cout << "Számítás\n";

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

    for ( int x = 0; x < szelesség; ++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;
}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

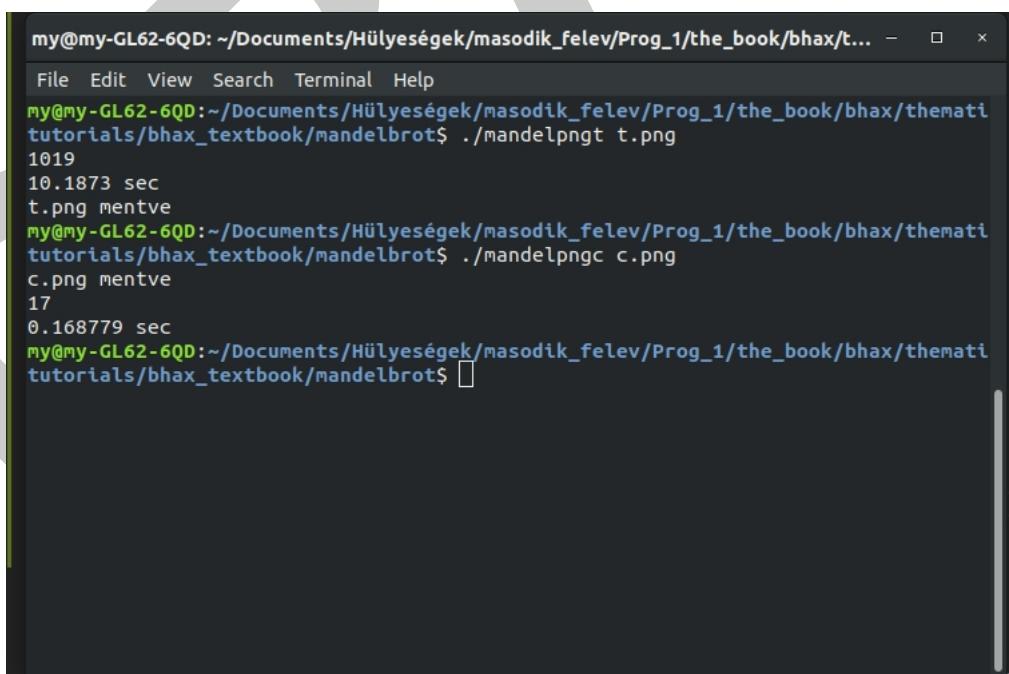
[bhax/thematic_tutorials/bhax_textbook/mandelbrot/mandelpngc_60x60_100.cu](#)

Hogy sokkal gyorsabbá tegyük a programot, az Nvidia által kifejlesztett CUDA nyújtotta lehetőséget fogjuk kihasználni. Ehhez szükséges a csomag telepítése:

sudo apt install nvidia-cuda-toolkit

nvcc mandelpngc_60x60_100.cu -lpng16 -O3 -o mandelpngc

Ezután ha lefordítjuk és futtatjuk a kódot, láthatjuk, hogy a sebesség kb. a 60-szorosára nő.



```
my@my-GL62-6QD: ~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook/mandelbrot$ ./mandelpngt t.png
1019
10.1873 sec
t.png mentve
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook/mandelbrot$ ./mandelpngc c.png
c.png mentve
17
0.168779 sec
my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook/mandelbrot$
```

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

A feladat megoldásában tutorált engem: Tóth Balázs.

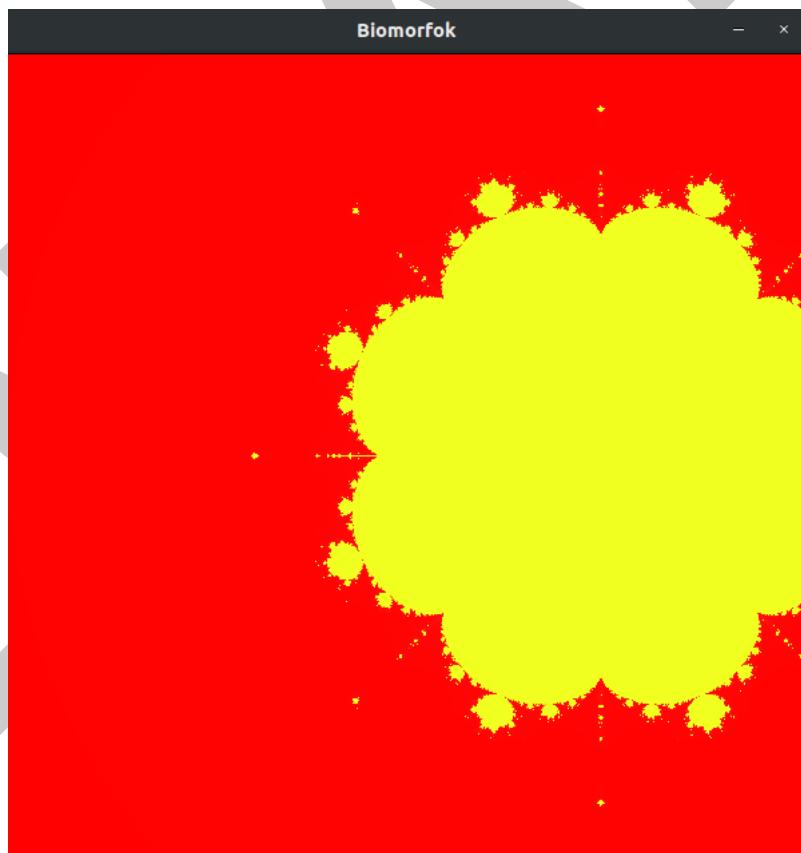
Ehhez a feladathoz szükséges a QT telepítése:

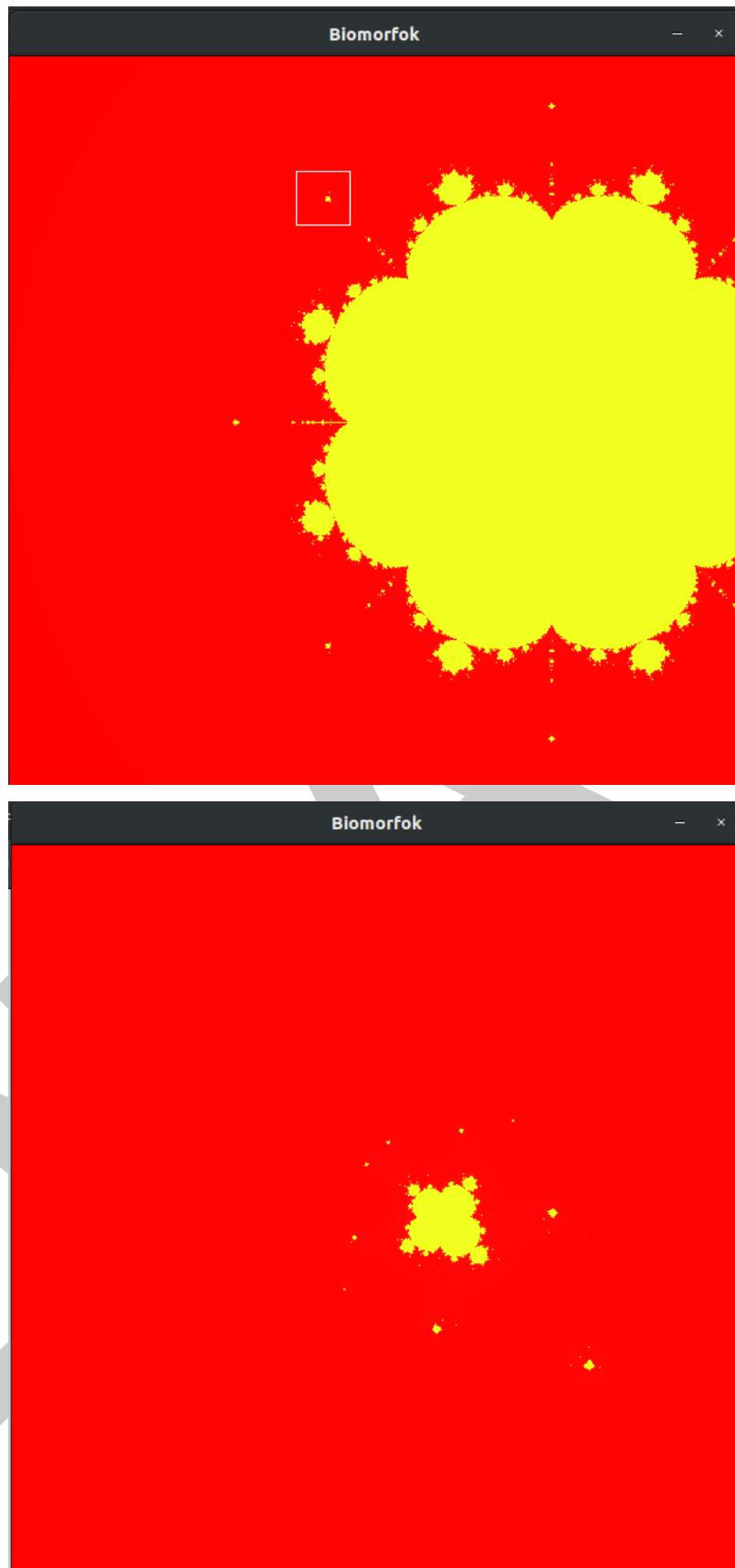
sudo apt-get install qt5-default

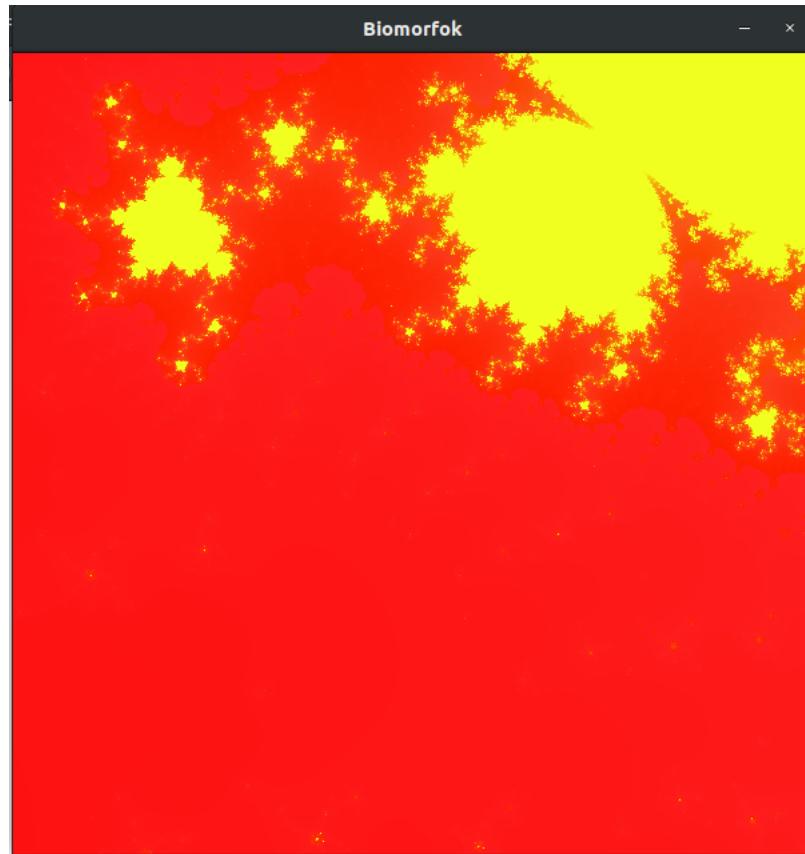
Ezután érdemes az összes forráskódot elkülöníteni egy mappába, ahol a **qmake -project** parancssal létrehozzuk a valami.pro fájlt, majd a **qmake valami.pro** parancssal a MakeFile-t. A futtatáshoz ki kell egészetünk a valami.pro fájlt a következő sorokkal:

```
QT += widgets  
CONFIG += c++11
```

Egy make parancs után már futtatható is a program.







5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása:

[bhax/thematic_tutorials/bhax_textbook/mandelbrot/java/MandelbrotHalmazNagyítás.java](#)
[bhax/thematic_tutorials/bhax_textbook/mandelbrot/java/MandelbrotHalmaz.java](#)
[bhax/thematic_tutorials/bhax_textbook/mandelbrot/java/MandelbrotIterációk.java](#)

Itt a grafikus megjelenítéshez az AWT csomagot fogjuk használni. Az általa nyújtott BufferedImage osztály nagyban megkönnyíti a dolgunkat, főleg amikor pillanatfelvételt készítünk.

```
public void pillanatfelvétel() {  
    // Az elmentendő kép elkészítése:  
    java.awt.image.BufferedImage mentKép =  
        new java.awt.image.BufferedImage(szélesség, magasság,  
        java.awt.image.BufferedImage.TYPE_INT_RGB);  
    java.awt.Graphics g = mentKép.getGraphics();  
    g.drawImage(kép, 0, 0, this);  
    g.setColor(java.awt.Color.BLACK);  
    g.drawString("a=" + a, 10, 15);  
    g.drawString("b=" + b, 10, 30);
```

```
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iterációsHatár, 10, 75);
if(számításFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
// A pillanatfelvétel képfájl nevének képzése:
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyitas_");
sb.append(++pillanatfelvételszámLálo);
sb.append("_");
// A fájl nevébe belelevesszük, hogy melyik tartományban
// találtuk a halmazt:
sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");
// png formátumú képet mentünk
try {
    javax.imageio.ImageIO.write(mentKép, "png",
        new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
    e.printStackTrace();
}
}
```

A kijelöléshez és nagyításhoz elmentjük az egér pozíóját az egérgomb lenyomásakor és felengedésekor, valamint amíg lenyomva tartjuk, folyamatosan újrarajzolunk mindenhol hogy látszódjon a négyzet amivel kijelölünk.

```
addMouseListener(new java.awt.event.MouseAdapter() {
    // Egér kattintással jelöljük ki a nagyítandó területet
    // bal felső sarkát vagy ugyancsak egér kattintással
    // vizsgáljuk egy adott pont iterációt:
    public void mousePressed(java.awt.event.MouseEvent m) {
        // Az egérmutató pozíciója
        x = m.getX();
        y = m.getY();
        // Az 1. egér gombbal a nagyítandó terület kijelölését
        // végezzük:
        if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
```

```
// A nagyítandó kijelölt területet bal felső sarka: (x, ←
// y)
// és szélessége (majd a vonszolás növeli)
mx = 0;
my = 0;
repaint();
} else {
    // Nem az 1. egér gombbal az egérmutató mutatta c
    // komplex számból indított iterációkat vizsgálhatjuk
    MandelbrotIterációk iterációk =
        new MandelbrotIterációk(
            MandelbrotHalmazNagyító.this, 50);
    new Thread(iterációk).start();
}
}
// Vonszolva kijelölünk egy területet...
// Ha felengedjük, akkor a kijelölt terület
// újraszámítása indul:
public void mouseReleased(java.awt.event.MouseEvent m) {
    if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
        double dx = (MandelbrotHalmazNagyító.this.b
                    - MandelbrotHalmazNagyító.this.a)
                    /MandelbrotHalmazNagyító.this.szélesség;
        double dy = (MandelbrotHalmazNagyító.this.d
                    - MandelbrotHalmazNagyító.this.c)
                    /MandelbrotHalmazNagyító.this.magasság;
        // Az új Mandelbrot nagyító objektum elkészítése:
        new MandelbrotHalmazNagyító(
            MandelbrotHalmazNagyító.this.a+x*dx,
            MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
            MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
            MandelbrotHalmazNagyító.this.d-y*dy,
            600,
            MandelbrotHalmazNagyító.this.iterációsHatár);
    }
}
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // A nagyítandó kijelölt terület szélessége és magassága:
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
```

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/welch/PolárGenerátor.java

Tanulságok, tapasztalatok, magyarázat... térd 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!

```
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 = Math.sqrt(u1*u1 + u2*u2);
                if(w <= 0.001) continue;
                w = 1.0/w;
                v1 *= w;
                v2 *= w;
            } while(v1*v1 + v2*v2 > 1.0);
            tárolt = Math.sqrt(v1*v1 + v2*v2);
            nincsTárolt = false;
        }
        return tárolt;
    }
}
```

```
v2 = 2*u2 - 1;

w = v1*v1 + v2*v2;

} 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());
}

}
```

Ha eltekintünk attól, hogy a logikai változót fordítva használták, a kód tényleg ugyanúgy van megírva, ahogyan azt mi tettük.

```
synchronized public double nextGaussian() {
    // See Knuth, ACP, Section 3.4.1 Algorithm C.
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1; // between -1 and 1
            v2 = 2 * nextDouble() - 1; // between -1 and 1
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}
```

```
}
```

Most hogy önbizalmat kaptunk, csináljuk me a c++ verziót is:

```
#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;
}
```

6.2. LZW

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/welch/binfa_c.c

Egy fa adatszerkezet leprogramozására c-ben a struct típus fog segíteni. Egy ilyen fa ugyanolyan szerkezetű elemekből áll: érték(gyökér), mutató a jobboldali levélelemre, mutató a beloldali levélelemre.

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

Ezután az `uj_elem()` függvénytel lefoglaljuk a memóriát a fa gyökér elemének és ha nincs neki elég memória, akkor egy hibaüzenet után kilépünk a programból.

```
BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

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

A while ciklus fejébe folyamatosan beolvassuk a futtatáskor átadott txt állomány karaktereit, amelyek a fa egyes elemei. A 0-ást balra, az 1-est jobbra fogjuk beágyazni a fába úgy, hogy ha ismeretlen bitsorozathoz jutunk, akkor azt "letörjük" és beletesszük a fába:

00011101110 -> 0 00 1 11 01 110



```
char b;

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

while (read (0, (void *) &b, 1))
{
    write (1, &b, 1);
    if (b == '0')
    {
        if (fa->bal_nulla == NULL)
```

```
{  
    fa->bal_nulla = uj_elem ();  
    fa->bal_nulla->ertek = 0;  
    fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;  
    fa = gyoker;  
}  
else  
{  
    fa = fa->bal_nulla;  
}  
}  
else  
{  
    if (fa->jobb_egy == NULL)  
    {  
        fa->jobb_egy = uj_elem ();  
        fa->jobb_egy->ertek = 1;  
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;  
        fa = gyoker;  
    }  
    else  
{  
        fa = fa->jobb_egy;  
    }  
}  
}  
}
```

A kiíratásnál a fát inorder módon bejárva kiíratjuk a fa elemeit és azok mélységét. Majd a végén felszabadítjuk a memóriát, amelyet elemenként kell elvégeznünk.

```
printf ("\n");  
kiir (gyoker);  
extern int max_melyseg;  
printf ("melyseg=%d", max_melyseg);  
szabadit (gyoker);  
}  
  
static int melyseg = 0;  
int max_melyseg = 0;  
  
void  
kiir (BINFA_PTR elem)  
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        if (melyseg > max_melyseg)  
            max_melyseg = melyseg;  
  
        kiir (elem->jobb_egy);  
        for (int i = 0; i < melyseg; ++i)
```

```
printf ("---");

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

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

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tamop425/0005_22_algoritmizalas_alapjai_scorm_11135_fa.html

bhax/thematic_tutorials/bhax_textbook/welch/binfa_preorder.c
bhax/thematic_tutorials/bhax_textbook/welch/binfa_postorder.c

Az előző feladatban a `kiir()` függvényben inorder módon jártuk be a fát, ami annyit jelent, hogy:

- járjuk be inorder módon a bal oldali részfát
- dolgozzuk fel a gyökérelemet
- járjuk be inorder módon a jobboldali részfát

Mi a karakteres környezet végett jobboldalról kezdtük a fabejárást, hogy a kirajzolt fa elrendezése helyes legyen.

Preorder bejárás: ha a fa üres, akkor vége az eljárásnak. Ha nem, akkor dolgozzuk fel a gyökérelemet, majd preorder eljárással járjuk be a bal oldali részfát, majd a jobboldali részfát.

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

```
{  
    ++melyseg;  
    if (melyseg > max_melyseg)  
        max_melyseg = melyseg;  
  
    // gyokerelem feldolgozása  
    for (int i = 0; i < melyseg; ++i)  
        printf ("---");  
  
    printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↔  
           , melyseg);  
  
    // baloldali részfa preorder bejárása  
    kiir (elem->bal_nulla);  
  
    // jobboldali részfa preorder bejárása  
    kiir (elem->jobb_egy);  
  
    --melyseg;  
}  
}
```

Posztorder bejárás: ha a fa üres, akkor vége az eljárásnak. Ha nem, akkor posztorder módon járjuk be a bal oldali, majd a jobb oldali részfát, végül dolgozzuk fel a gyökérelemet.

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

A pre- és posztorder módon bejárt fa nem valami látványos kirajzolva, viszont más célú adatfeldolgozásnál hasznosak lehetnek.

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/welch/z3a7.cpp

C++ nyelven a fát osztályok/objektumok segítségével fogjuk megvalósítani, aminek következetében a memóriaoglalás és felszabadítása az osztály része lesz konstruktur és destruktur szerepében.

A feladatnak megfelelően elkészítjük a LZWBInFa (Tree) osztályt és benne a Csomopont (Node) beágyazott osztályt, majd a fa mutatót a gyoker elemre állítjuk a konstruktornak.

```
class LZWBInFa
{
    LZWBInFa () :fa (&gyoker)
    {
    }
    ~LZWBInFa ()
    {
        szabadit (gyoker.egyesGyermek ());
        szabadit (gyoker.nullasGyermek ());
    }
    ...
    class Csomopont
    {
        char betu;
        Csomopont *balNulla;
        Csomopont *jobbEgy;
    };
    ...
    Csomopont *fa;
    ...
    Csomopont gyoker;
    int maxMelyseg;
    ...
};
```

Az osztályoknak köszönhetően minden ami a fával kapcsolatos, egy egységbe kezelhetjük, ezért a fa feltöltését isbeletettük. Ezt úgy csináltuk, hogy átírtuk a "<<" operátort.

```
void operator<< (char b)
{
    // Mit kell betenni éppen, '0'-t?
```

```
if (b == '0')
{
    /* Van '0'-s gyermeké az aktuális csomópontnak?
    megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
    if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
    {
        // elkészítjük, azaz páldányosítunk a '0' betű akt. ←
        // parammal
        Csomopont *uj = new Csomopont ('0');
        // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
        // jegyezze már be magának, hogy nullás gyereke mostantól ←
        // van
        // küldjük is Neki a gyerek címét:
        fa->ujNullasGyermek (uj);
        // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
        fa = &gyoker;
    }
    else // ha van, arra rálépünk
    {
        // azaz a "fa" pointer már majd a szóban forgó gyermekre ←
        // mutat:
        fa = fa->nullasGyermek ();
    }
}
// Mit kell betenni éppen, vagy '1'-et?
else
{
    if (!fa->egyesGyermek ())
    {
        Csomopont *uj = new Csomopont ('1');
        fa->ujEgyesGyermek (uj);
        fa = &gyoker;
    }
    else
    {
        fa = fa->egyesGyermek ();
    }
}
```

```
int
main (int argc, char *argv[])
{
    ...
    while (beFile.read ((char *) &b, sizeof (unsigned char)))
    {
```

```
...
for (int i = 0; i < 8; ++i)
{
    // maszkolunk eddig..., most már simán írjuk az if fejébe a ←
    // legmagasabb helyiértékű bit vizsgálatát
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ←
    // megmondja melyik:
    if (b & 0x80)
        // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW ←
        // fa objektumunkba
        binFa << '1';
    else
        // különben meg a '0' betűt:
        binFa << '0';
    b <<= 1;
}

}
...
}
```

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/welch/binfa_gyoker.cpp

A gyoker értékének átadása a program minden részén mostmár referenciahivatkozás nékül történik.

```
128c135
<                                //fa = &gyoker; e helyett
---
>                                fa = gyoker; // lett ez
```

Ez azért van, mert mostmár a gyoker egy mutató és így egyenesen a memóriacímet adjuk át a fa változónak, ami szintén egy mutató. Ahhoz hogy ez működjön a gyoker deklarációját is módosítanunk kell a következő módon:

```
Csomopont *gyoker = new Csomopont();
```

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/welch/binfa_mozgato.cpp

A mozgató szemantika azért lesz hasznos, mert amíg a másoló konstruktor a másoláskor megduplázza a lefoglalt memóriát, addig ezt a mozgatással elkerülhetjük a mutatók használatával.

Létrehozzuk a binFa2 objektumot és az `std::move()` függvényel "átmozgatjuk" a régi fa tartalmát az újba. Az `std::move()` valójában nem mozgat semmit, hanem az eredeti fából jobbérték referenciát készít (lásd: <https://en.cppreference.com/w/cpp/utility/move>).

```
LZWBInFa binFa2 = std::move(binFa);
```

Szóval ahhoz hogy ez működjön, át kell írnunk az egyenlőség operátort és meg kell írni a mozgató konstruktort. Az `std::swap` segítségével megcseréljük a két fa mutatóit, így az új fa a régire, míg a régi a semmibe fog mutatni.

```
LZWBInFa (LZWBInFa&& regi)
{
    gyoker = nullptr;
    *this = std::move(regi);
}

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

A kiíratásnál láthatjuk, hogy a mozgatás után már csak a binFa2-ben vannak meg az adatok.

```
/bhax_textbook/welch$ ./binfa_mozgato fa.txt -o ki.txt
```

Az eredeti fa mozgatás előtt:

```
-----1(1)
-----0(2)
---/(0)
-----1(2)
-----0(1)
-----0(2)
```

Az eredeti fa mozgatás után:

A binFa2 fa mozgatás után:

```
-----1(1)
```

-----0 (2)
---/(0)
-----1 (2)
-----0 (1)
-----0 (2)

DRAFT

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/conway/hangya/main.cpp

bhax/thematic_tutorials/bhax_textbook/conway/hangya/antwin.cpp

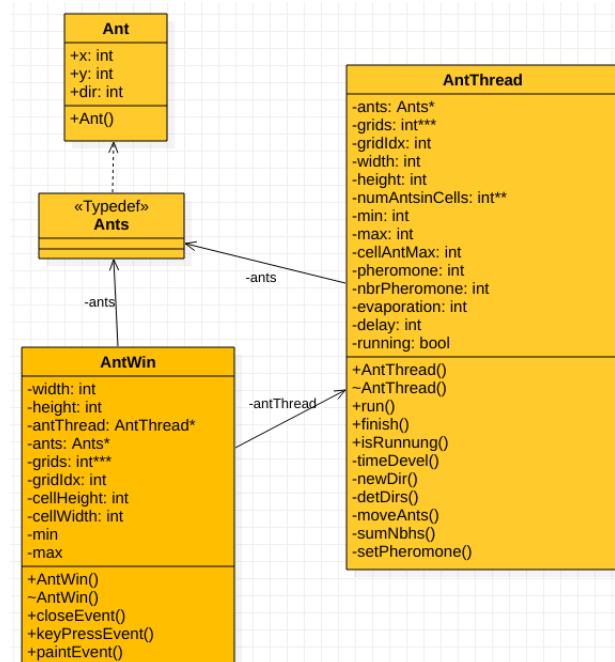
bhax/thematic_tutorials/bhax_textbook/conway/hangya/antthread.cpp

A mostani Qt projektünkkel a hangyák útvonalait fogjuk szimlálni. A hangyák igyekeznek egymás szaga alapján közlekedni, ha viszont nem éreznek maguk körül feromont, akkor az irány véletlenszerű. A hangyák számát, a feromon párolgását, a szimuláció sebességét és egyéb paramétereket parancssori argumentumként adjuk meg futtatáskor.

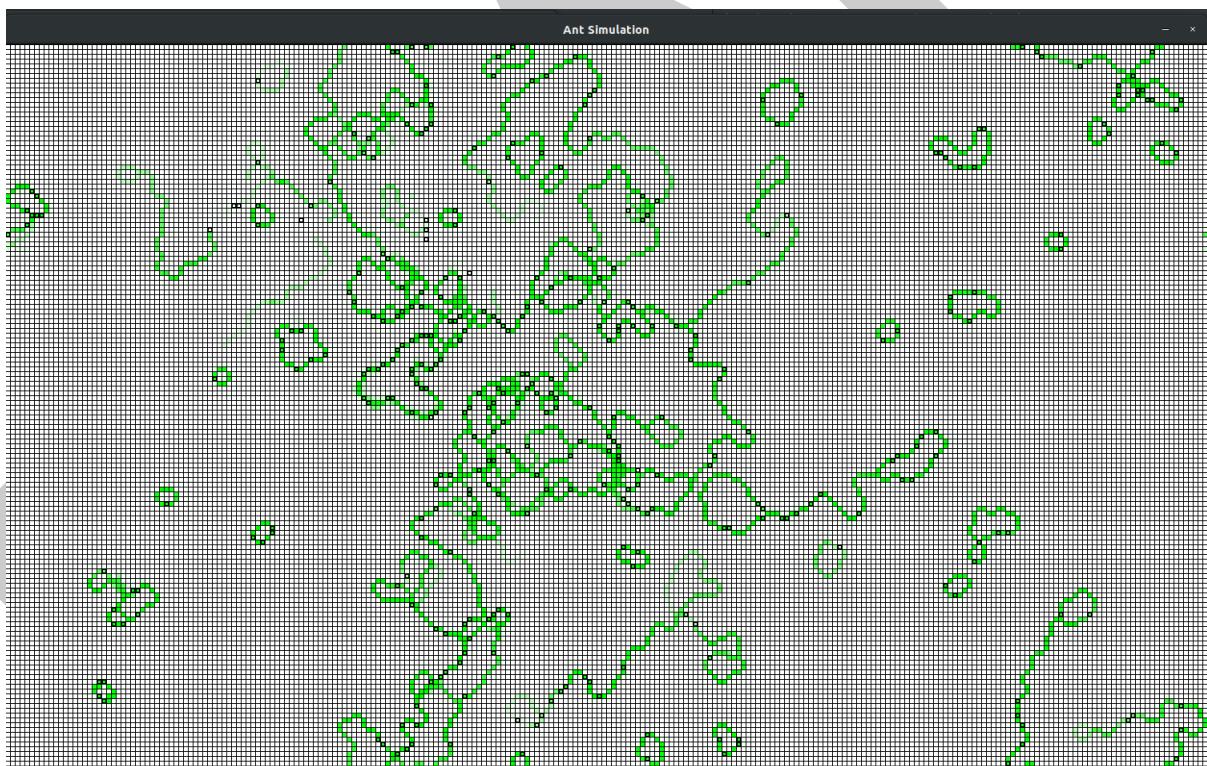
Options:

-h, --help	Displays this help.
-v, --version	Displays version information.
-w, --szelesseg <szelesseg>	Oszlopok (cellakban) szama.
-m, --magassag <magassag>	Sorok (cellakban) szama.
-n, --hangyaszam <hangyaszam>	Hangyak szama.
-t, --sebesseg <sebesseg>	2 lepes kozotti ido (millisec-ben).
-p, --parolgas <parolgas>	A parolgas erteke.
-f, --feromon <feromon>	A hagyott nyom erteke.
-s, --szomszed <szomszed>	A hagyott nyom erteke a szomszedokban.
-d, --alapertek <alapertek>	Indulo ertekek a cellakban.
-a, --maxcella <maxcella>	Cella max erteke.
-i, --mincella <mincella>	Cella min erteke.
-c, --cellameret <cellameret>	Hany hangya fer egy cellaba.

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



Minden egyes hangynak van pozíciója és iránya. A számításokhoz 2 rácsot használunk - egyik tartalmazza a jelenlegi állapotot, a másik pedig az újat - , így olyan, mintha minden egyszerre mozognának.



7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/conway/eletjatek/java/Sejtautomata.java

A feladat megoldásában tutorált: Tóth Balázs.

A kód elején definiáljuk a Sejtautomata osztályt a `java.awt.Frame` gyermeként hogy használni tudjuk az ablakkezelő parancsokat. Mivel ez a verzió főleg a megjelenítésben és az input kezelésében tér el a C++-ostól, nézzük hát meg ezeket.

A konstruktornak átadjuk az ablak méretét. Ezután jönnek a Listenerek, amelyek biztosítják az ablak interaktivitását. Az `addKeyListener()` függvényben a `getKeyCode()` adja vissza a leütött billentyű értékét:

- N billentyűzet esetén nagyít,
- K billentyűzet esetén kicsinyít,
- g billentyűzet esetén gyorsít,
- L billentyűzet esetén lassít.

```
public class Sejtautomata extends java.awt.Frame implements Runnable {  
    public static final boolean ÉLŐ = true;  
    public static final boolean HALOTT = false;  
    protected boolean[][][] rácsok = 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;  
        rácsok[0] = new boolean[magasság][szélesség];  
        rácsok[1] = new boolean[magasság][szélesség];  
        rácsIndex = 0;  
        rács = rácsok[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();
    }
});
```

Az egérgomb lenyomásával életre kelhetünk vagy éppen megölhetjük a sejteket. A robot objektumot itt még csak létrehozzuk és majd a paint () függvényben használjuk pillanatfelvétel készítésére.

```
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)));  
    }  
}
```

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

bhax/thematic_tutorials/bhax_textbook/conway/eletjatek/main.cpp
bhax/thematic_tutorials/bhax_textbook/conway/eletjatek/sejtablak.cpp
bhax/thematic_tutorials/bhax_textbook/conway/eletjatek/sejtszal.cpp

A feladat megoldásában tutorált: Schachinger Zsolt.

Ez az ún. játék valójában egy sejtszimulátor, amit John Horton Conway talált ki 1970-ben. A játék lényege, hogy az előre elhelyezett selytek mivé formálódnak vagy hogy életben maradnak e egyáltalán.

A sejtek alakulását a következő szabályok irányítják:

- minden sejtnek 2 állapota lehet: élő vagy halott.
- ha egy sejtnek kevesebb mint 2 élő szomszédja van, az meghal.
- ha egy sejtnek 2 vagy 3 élő szomszédja van, az életben marad.
- ha egy sejtnek több mint 3 élő szomszédja van, az meghal.
- ha egy halott sejtnek pontosan 3 élő szomszédja van, az életre kel.

```
int SejtSzal::szomszedokSzama(bool **racs,
                                int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
    // A nyolcszomszodok végigzongorza:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgált sejtet magt kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejttról színek szomszai
                // a szembe oldalakon ("peridikus hatrfelttel")
                int o = oszlop + j;
                if(o < 0)
                    o = szelesseg-1;
                else if(o >= szelesseg)
                    o = 0;

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

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

/**
```

```
* A sejttr idbeli fejldse a John H. Conway fle
* letjtk sejtautomata szablyai alapjn trtnik.
* A szablyok rszletes ismertetst lsd pldul a
* [MATEK JTK] hivatkozsban (Cskny Bla: Diszkrt
* matematikai jtkok. Polygon, Szeged 1998. 171. oldal.)
*/
void SejtSzal::idoFejlodes() {

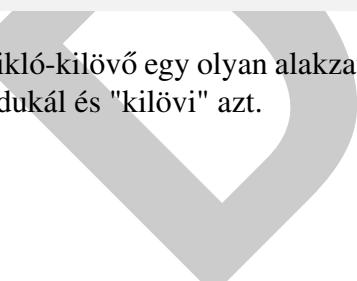
    bool **racsElotte = racsok[racsIndex];
    bool **racsUtana = racsok[(racsIndex+1)%2];

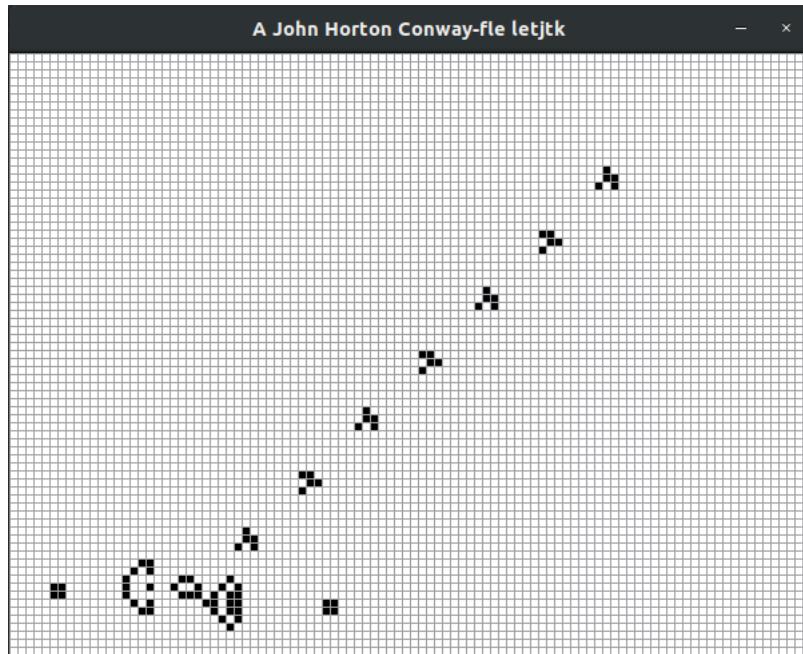
    for(int i=0; i<magassag; ++i) { // sorok
        for(int j=0; j<szelesség; ++j) { // oszlopok

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

            if(racsElotte[i][j] == SejtAblak::ELO) {
                /* 1 l marad, ha kett vagy hrom l
                szomszedja van, klnben halott lesz. */
                if(elok==2 || elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            } else {
                /* Halott halott marad, ha hrom l
                szomszedja van, klnben l lesz. */
                if(elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            }
        }
    }
    racsIndex = (racsIndex+1)%2;
}
```

A sikló-kilövő egy olyan alakzat, amelyben 2 ide-oda pattogó alakzat minden 15. ciklusban egy harmadikat produkál és "kilövi" azt.





7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

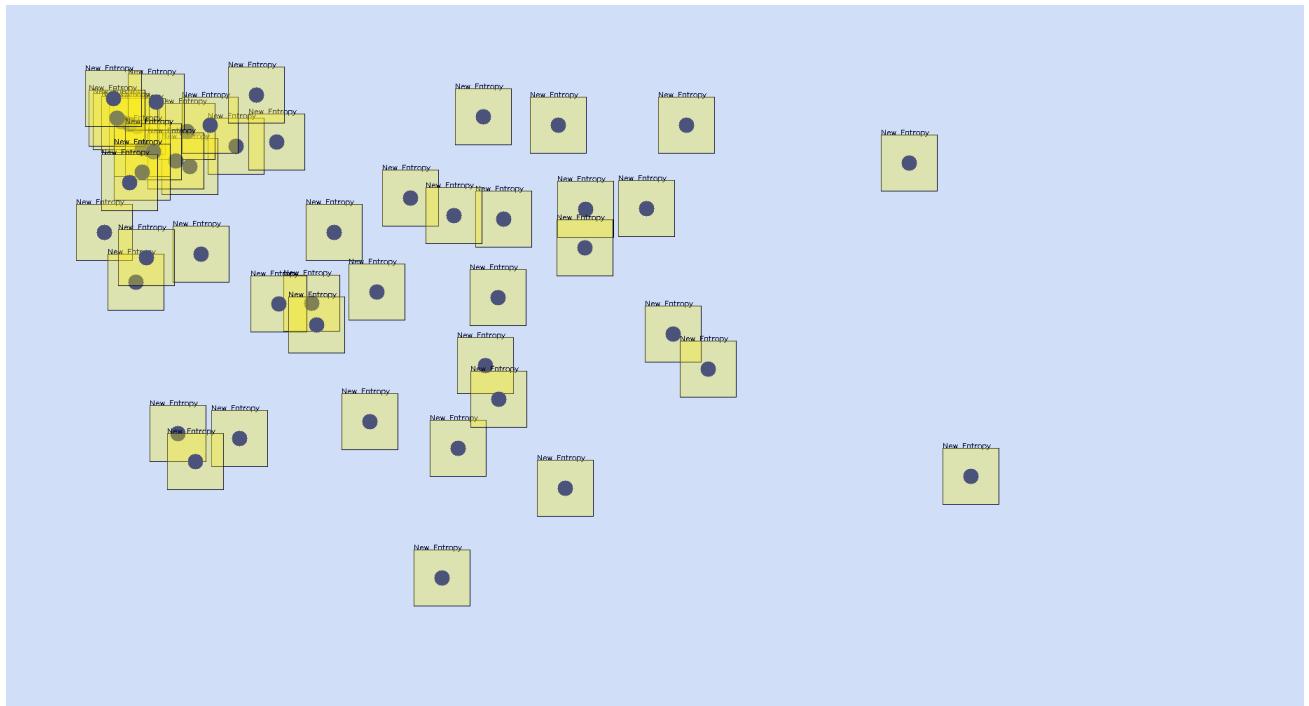
[bhax/thematic_tutorials/bhax_textbook/conway/brainb/main.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/brainb/BrainBWin.cpp](#)

[bhax/thematic_tutorials/bhax_textbook/conway/brainb/BrainBThread.cpp](#)

Ehhez a programhoz szükséges az opencv telepítése: **sudo apt-get install libopencv-dev**

Az esport kezd egyre inkább elterjedni. Ez a program az esportolóknak nyújt lehetőséget arra, hogy lemérjék teljesítményüket, pontosabban hogy menyire képesek a zavaró tényezők mellet a kijelölt célra koncentrálni. Az méréshez 10 percig kell követni a kijelölt négyzetet lenyomott egérgombbal.



Ha végeztünk, akkor az eredményt kiírja nekünk egy fájlba.

NEMESPOR BrainB Test 6.0.3

```

time      : 6000
bps       : 59860
noc       : 49
nop       : 0
lost      :
2230 820 3380 2260 61370 34150 23480 4850 0 21450 38690 30700 17570 15690 ←
    9910 29510 20380 25880 42260 43430 54240 73360 49950 59940 55620
mean     : 28844
var      : 21808.5
found    : 0 12400 15360 18040 40460 32870 20320 38910 39440 37640 31740 ←
    48490 49680 44280 57730 33190 16150 11870 19730 7320 8540 9720 14310 ←
    31040 48770 23730 20870 0 0 5950 19370 12320 11440 14170 24210 12440 ←
    23500 18670 25330 15740 8890 29860 23810 45750 39770 37730 41060 24600 ←
    22720 30870 27360 28220 42620 27790 36650 48690 39250 61970 51660 58200 ←
    54050 66020 61920 54830 64910 61410 45240 47300 49080
mean     : 31130
var      : 17635.2
lost2found: 0 49680 48770 0 0 12320 11440 15740 22720 48690 39250 54050 ←
    54830 45240
mean     : 28766
var      : 21827.6
found2lost: 61370 34150 23480 0 21450 30700 17570 29510 42260 43430 54240 ←
    73360 59940
mean     : 37804
var      : 20565
mean(lost2found) < mean(found2lost)
time     : 10:0
U R about 4.06311 Kilobytes

```

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/schwarzenegger/mnist_softmax.py

A program futtatásához szükséges a Python3 és a TensorFlow csomagok telepítése. A TensorFlow-nál választhatunk, hogy A CPU-s vagy a GPU-s (CUDA) verziót telepítjük. A nagyobb teljesítmény érdekében a CUDA-s megoldást javaslok, feltéve ha a kártyád rendelkezik ilyen támogatással. Az instalálásban segít a következő link: <https://www.tensorflow.org/install>

A kód elején importáljuk a szükséges modulokat, így a tensorflowot is "tf"-ként, hogy kevesebbet keljen gépelni.

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

import argparse

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

import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

import matplotlib.pyplot
```

Ezután a mnist változóba beolvassuk az adatokat. Az x változó fogja reprezentálni az első réteget, amely adatokkal látja el neurális hálónkat. A zeros() függvény által visszaadott csupa nullákból álló tensort felhasználva inicializáljuk a W változót, ami a súlyokat fogja tartalmazni. A matmul() visszaadja az x és a W szorzatát es b-t hozzáadva elmentjük az y változóba.

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

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

A `mnist.train.next_batch(100)` függvény biztosít nekünk 100 véletlenszerűen kiválasztott adatot a `mnist` adabázisból és ezeket használjuk fel a tanításhoz.

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

Ezután leteszteljük a neurális hálónkat és egy 0-1 közötti értékkel kiíratjuk annak pontosságát. Következő lépésekben külön ablakban megjelenik egy írott szám és kiíratjuk, hogy a hálózat ezt mire ismeri fel.

```
# Test trained model
print("-- A hálózat tesztelése")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("-- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.test.images,
                                                       y_: mnist.test.labels}))
print("-----")

print("-- A MNIST 42. tesztképenek felismerése, mutatom a számot, a továbbiakhoz csukd be az ablakat")

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

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

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

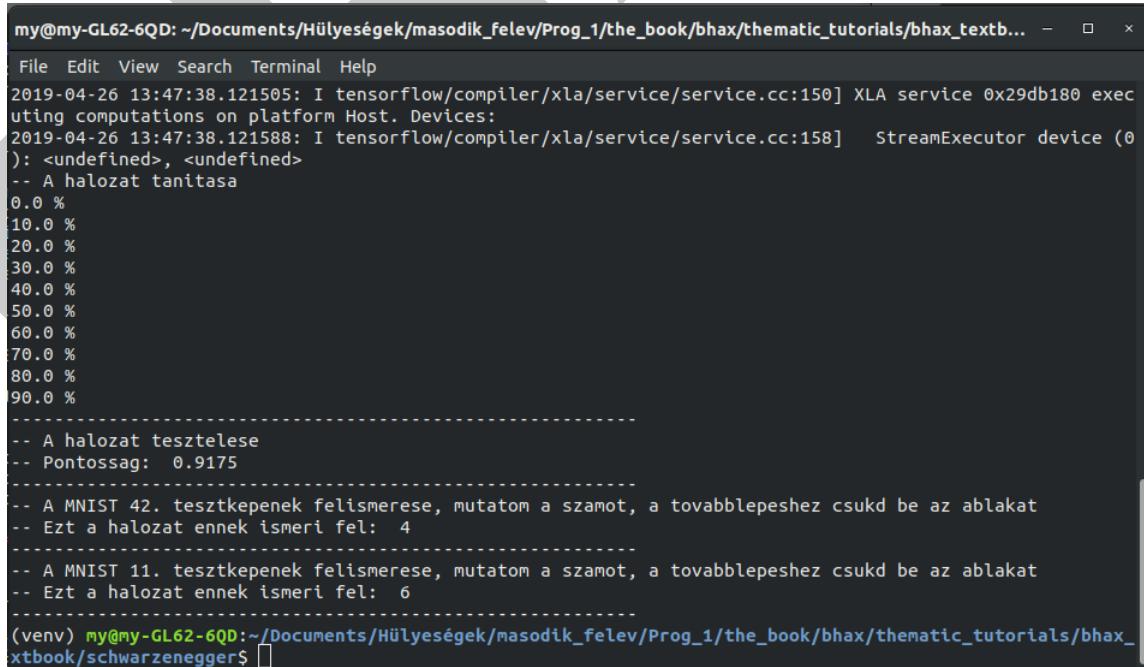
print("-- Ezt a hálózat ennek ismeri fel: ", classification[0])
print("-----")
```

```
#print("-- A sajat kezi 8-asom felismerese, mutatom a szamot, a ←
#      tovabblepeshet csukd be az ablakat")
print("-- A MNIST 11. tesztkepenek felismerese, mutatom a szamot, a ←
#      tovabblepeshet csukd be az ablakat")
# img = readimg()
# image = img.eval()
# image = image.reshape(28*28)
img = mnist.test.images[11]
image = img
matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm. ←
    binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

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

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

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



A terminal window showing the execution of a Python script. The script performs image classification on the MNIST dataset using TensorFlow. The terminal output shows the training progress (0.0% to 90.0%), the test accuracy (0.9175), and the classification results for two specific images (MNIST 42 and MNIST 11).

```
my@my-GL62-6QD: ~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook/schwarzenegger$ python3 bhax_textbook.py
2019-04-26 13:47:38.121505: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x29db180 executing computations on platform Host. Devices:
2019-04-26 13:47:38.121588: I tensorflow/compiler/xla/service/service.cc:158] StreamExecutor device (0
): <undefined>, <undefined>
-- A halozat tanitasa
0.0 %
10.0 %
20.0 %
30.0 %
40.0 %
50.0 %
60.0 %
70.0 %
80.0 %
90.0 %
-----
-- A halozat tesztelese
-- Pontossag: 0.9175
-----
-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a tovabblepeshet csukd be az ablakat
-- Ezt a halozat ennek ismeri fel: 4
-----
-- A MNIST 11. tesztkepenek felismerese, mutatom a szamot, a tovabblepeshet csukd be az ablakat
-- Ezt a halozat ennek ismeri fel: 6
(venv) my@my-GL62-6QD:~/Documents/Hülyeségek/masodik_felev/Prog_1/the_book/bhax/thematic_tutorials/bhax_textbook/schwarzenegger$
```

8.2. Mély MNIST

Python

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/schwarzenegger/mnist_deep.py

Tanulságok, tapasztalatok, magyarázat...

```
# 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)
```

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

1. PASSZOLÁS: <https://www.facebook.com/photo.php?fbid=2431150020252463&set=p.2431150020252463&t>

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chaitin/factorial.lisp

A feladat megoldásához a [portacle](#) fejlesztői környezetet használtam, így nem szükséges a Lisp könyvtárak instalálása.

Általában ha egy számunkra új programozási nyelvvel találkozunk nagyjából ki tudjuk következtetni, hogy mit is akar az a program csinálni. Ez a Lisp-nél ritkáság, legalábbis számomra. Az egészben a legfurcsább a prefix alak, amit hát nincs más választás mint megszokni. Az iterációs megoldás do segítségével a következőképpen néz ki:

```
(do ((n 1 (+ n)) ; n kezdeti értéke 1, ciklusonkent 1-el noveeljük
      (l 1 (* n l))) ; l kezdeti értéke 1, ciklusonkent megsorozzuk az n-el ↔
      addig, ...
      (= 6 n) ; ... amíg az n nem egyenlo 6-al
      l) ; visszatérünk az l értékevel
)
```

A do külösszó utáni zárójelben dekláraljuk a ciklusváltozókat. Ezeknek először megadjuk a nevét majd a kezdő értékét és a kifejezést ami szerint az változni fog a ciklusok során. Ezt követi az iterációs feltétel és a visszatérési érték. A ciklus fő blokkját most üresen hagytuk.

A rekurziós verzió sokkal átláthatóbb és ez már urahasznosítható. Ennél a syntax-nál már egyszerűbb megállapítani, hogy miről is van szó. A defun kulcsszó után függvénynév és változó a zárójelben. Az if-el végeztetünk az önhivatkozásnak ha az N értéke 1, ellenben megsorozzuk az N-t az egyetlen kisebb szám faktoriálisával. Az utolsó sor már a kész függvény használatát mutatja be.

```
(defun factorial (N)
  (if (= N 1)
    1
    (* N (factorial (- N 1)))))

(FACTORIAL 5)
```

9.2. Gimp Scheme Script-fu: króm effekt

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

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chaitin/chrome/bhax_chrome3.scm bhax/thematic_tutorials/bhax_textbook/chaitin/chrome/bhax_chrome_border2.scm

A működtetéshez be kell másolnunk a .scm fájlokat a gimp scripts mappájába, amit meglelünk az edit/preferences/scriptsmenüpontban.

Első lépésben előkészítjük a szöveget. Ez túgy csináljuk, hogy létrehozunk egy réteget, feketére állítjuk az előtér színét és kitöltéssel befeketítjük a képet. Miután fehérre cserélük a színt, létrehozzuk a szöveget, egy új rétgen középre helyezzük a szöveget és a két reétegből eggyet csinálunk.

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

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

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

Itt alkallmazzuk a Gauss elmosást, ahol a RUN-INTERACTIVE paraméter miatt megjelenik számunkra egy ablak, ahol még állítgathatjuk a paramétereket. A gimp-drawable-levels-nél a színek alsó és felső határának állításával egy kicsit élesítjük a képet és megégyezik homályosítunk rajta egy keveset.

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

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

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

Ebben a lépésben meghívjuk a szín szerinti kiválasztást a fekete színre, aminek eredményeként kijelölődik a háttér és ezt invertáljuk egy függvénytel. Ezután létrehozunk egy átlátszó réteget és hozzáadjuk a képhez. A 7-es lépésben szürkével kitöltjük az átlátszó réteget. Kockaleképezéssel 3d hatást adunk a szövegnekn. Végül a színgörbe módosításával fémes hatást adunk a szövegnekn. Ehhez létrehoztunk egy color-curve nevű függvényt, amely tartalmazza a 8 pont koordinátáját.

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

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

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

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

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

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

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

2. PASSZOLÁS: <https://www.facebook.com/photo.php?fbid=2431150020252463&set=p.2431150020252463&t>

10. fejezet

Helló, Gutenberg!

10.1. Juhász István féle könyv

Egy számítógép programozására három nyelvi szintet különböztetünk meg:

- Gépi kód
- assembly szint
- Magas szint

Mink a magas szintű programozási nyelvekkel foglalkozunk. A magas szintű nyelven megírt forrásszöveg helyesírását szintaktikának, jelentését szemantikának nevezzük. Egy magas szintű programozási nyelvet ez a kettő határoz meg. A processzor csak a gépi kódot érti, ezért a gépi kód előállításához vagy fordítót vagy interpretert kell használnunk. A fordítóprogram a lexikális elemzés, szintaktikai elemzés, szemantikai elemzés és kódgenerálás lépések végrehajtásával állítja elő a forrásszövegből a tárgyprogramot. Az interpereteres módszer is elvégzi az első 3 lépést, viszont tárgyprogramot nem készít, ehelyett soronként értelmezi és végrehajtja az utasításokat. A programnyelv szabványát hivatkozási nyelvnek hívjuk, ahol a szintaktikai és szemantikai szabályok vannak részletezve. A programok írásához leggyakrabban integrált fejlesztői környezetet (IDE - Integrated Development Environment) használunk, amelyek nagyban megkönnyítik a nagyobb projektek kezelését. Ezek tartalmazzák a szövegszerkesztőt, fordítót / interpretert és futtató rendszert. Az imperatív nyelvek algoritmusokból állnak. Ezek lehetnek eljárásorientált vagy objektumorientált nyelvek. És vannak a deklaratív nyelvek. Ezek nem algoritmikusak és a programozó csak a problémát adja meg, a nyelvi implementációba van beépítve a megoldás módja.

Az adattípusok absztrakt programozási eszközök, amelyek névvel (azonosítóval) rendelkeznek. Az adattípusokhoz hozzáartoznak a rajta végrehajtható műveletek. Valamely programozási nyelvekben mi válsztunk típust az azonosítóhoz, míg a többiben a fordítóprogram/interpreter. Egyes nyelvek megengedik számunkra, hogy saját típust definiálunk, ezek leggyakrabban egyszerűbb típusokból állnak. Ezek az alapvető típusok a következők: valós (float, double, long double) ill. egész szám (int, short[int], long[int]), karakteres (char), logikai típus és a mutató. Ha nevesített konstanst hozunk létre, akkor megadjuk a típusát és értékét az elején és utána már nem módosítható, csak a nevével hivatkozunk rá. Ezekben kívül ottvannak az összetett adattípusok, minit például a tömb vagy a rekord (Pascal). * a zárójelben levő típusnevek a C nyelvből vannak.

Az utasításoknak 2 nagy csoportja van: deklarációs és végrehajtható. A deklarációs utasítások a fordítóprogramnak szólnak. Ezek az utasítások nem fordulnak, de befolyásolják a tárgykódot. A tárgykód a

végrehajtható utasításokból (pl.: értékadó utasítás, üres u. (continue), ugró u. (GOTO), elágaztató u., ciklusszervező u., hívó u., vezérlésátadó u., i/o és egyéb utasítások) generálódik.

10.2. K & R, A C programozási nyelv

A változók és az állandók a programok alapvető részei. A deklarálásuk során megadjuk a nevüket, típusukat és esetleg a kezdeti értéküket. A nevekre nézve vannak némi megkötések, pl.: nem tartalmazhat pontot, nem lehetnek kulcsszavak (mint if, else, int), stb.

A c-ben csak néhány alapvető adattípus van:

- int - egész szám
- float - egyszeres pontosságú lebegőpontos szám
- double - kétszeres pontosságú lebegőpontos szám

Ezt kiegészíthetjük a short, long és unsigned minősítő szimbólumokkal. A short és a long a méretét szabályozza. Az unsigned(előjel nélküli) számokra a modulo 2n aritmetika szabályai vonatkoznak, ahol n az int típust ábrázolóbit-ek száma; az unsigned számok minden pozitívák.

```
short int x;
long int y;
unsigned int z;
```

Data Type	Size in Bytes	Range	Format
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
short signed int	2	-32768 to -32767	%d
short unsigned int	2	0 to 65535	%u
signed int	2	-32768 to -32767	%d
unsigned int	2	0 to 65535	%u
long signed int	4	-2147483648 to +2147483647	%ld
long unsigned int	4	0 to 4294967295	%lu
Float	4	3.4e-38 to 3.4e+38	%f
Double	8	1.7e-308 to 1.7e+308	%lf
long double	10	3.4e-4932 to 1.1e+4932	%Lf

Vezérlési adatszerkezetek:

- Utasítások és blokkok: a pontosvessző lezárja az utasításokat, a kapcsos zárójelek pedig összefognak több utasítást mintha egyek lennének (pl. függvények).
- Elágaztató utasítás: az if és else utasításokkal eldönthetjük hogy a megadott kifejezés logikai értékétől függően mely utasítások hajtódjanak végre. Többirányú elágaztatás a switch utasítással érhető el, ahol az elágazás már nem logikai érték alapján történik.

- Ciklusszervező utasítások: lehetővé teszik valamennyi utasítás többszöri megismétlését. Előre meghatározott ismétlésszám esetén `for` ciklust, egyébként pedig `while` ciklust ajánlott használni. Az utóbbinál válszthatunk hogy az utasítások végrehajtása előtt vagy után akarjuk kiértékelni az iteráció folytatását meghatározó feltételt. Ez akkor hasznos, ha az utasításokat legalább egyszer le akarjuk futtatni.
- Vezérlő utasítások: a `break` utasítással azonnal kiléünk abból a ciklusból vagy elágaztatásból, amelyből ezt meghívtuk. Leggyakrabban a `switch` elágaztatásnál használjuk, hogy elkerüljük az egyes ágak egymás utáni végrehajtását. A `return` utasítás megszakítja a függvény futását és a neki átadott kifejezés értékét függvényértékként visszaadja.

10.3. BME: Szoftverfejlesztés C++ nyelven / Benedek Zoltán, Levendovszky Tíhamér

C vs C++:

Függvényparaméterek és viisszatérési értékeik: A következő C kódban a az üres paraméterlista azt jelenti, hogy a függvény akármennyi paraméterrel hívható.

```
void f ()  
{  
}
```

Ezzel ellentétben az üres paraméterlistát a C++-ban a `void` kulcsszó jelenti, így viszont már csak paraméter nélkül hívható a függvény. Ha tetszőleges számú paramétert akarunk, akkor a `void` helyett 3 pontot kell írni.

```
void f(void)  
{  
}
```

Main függvény: A C++ nyelvben a ha nem írunk `return`-t a függvénybe, akkor a fordító automatikusan sikeres lefutásjelzőt rak oda.

A `bool` típus: A C++-ba bevezették a logikai típust, amelynek értékei a `true` és a `false`, amelyek a nyelv kulcsszavai közé tartoznak.

```
bool negyesjegymegajanlo = true;
```

Függvények túlterhelése: Míg C-ben a függvényeket csak a nevük, C++-ban a nevük és az argumentumlistájuk azonosítják. Ez lehetővé teszi, hogy elkerüljük az azonos céllal rendelkező metódusok erőltett neveit.

```
void drawLine(int x, int y, int x2, int y2)  
{  
    ...  
}  
  
void drawLine(int x, int y, int x2, int y2, string color)  
{  
    ...  
}
```

```
void main()
{
    ...
    drawLine(0, 0, 100, 50);
    drawLine(10, 10, 500, 600, "white");
}
```

Alapértelmezett függvényargumentumok bevezetése a C++-ba:

```
void drawLine(int x, int y, int x2, int y2, string color = "black")
{
    ...
}

void main()
{
    ...
    drawLine(0, 0, 100, 50);
    drawLine(10, 10, 500, 600, "white");
}
```

Paraméterátadás referenciatípussal: A kóvetkező C programtól valószínülg azt várnánk, hogy az a értékét megnöveljük 2-vel. Ez azonban nem fog megtörténni, mert amikor átadjuk az a értékét, akkor az lemasolódik az i változóba ahol módosul is, de ez semmilyen hatással nincs az a értékére.

```
#include<stdio.h>

void f(int i)
{
    i = i + 2;
}

int main(void)
{
    int a = 0;
    f(a);
    printf("%d\n", a);
}
```

A várt eredmény eléréséhez létrehozunk egy mutatót, amely a referenciatípusként átadott memóriacímre fog mutatni.

```
#include<stdio.h>

void f(int* i)
{
    (*i) = (*i) + 2;
}

int main(void)
```

```
{  
    int a = 0;  
    f(&a);  
    printf("%d\n", a);  
}
```

DRAFT

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. C++ vs Java

Ez a szekció a következő 2 könyv összehasonlítását tartalmazza:

- Benedek Zoltán, Levendovszky Tíhamér Szoftverfejlesztés C++ nyelven
- Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.

"A Java nyelv teljesen objektumorientált." Ezzel a kijelentéssel kezdődik a Java 2 útikalauz könyvünk 1.1-es szekciója. Ha már van egy kis tapasztalatunk a c++ nyelvvel bizonyára nem ijedünk meg az objektumorientált szemléletmódtól, viszont a Javában ez nem csupán lehetőséggént tárul elénk, hanem akár a legegyszerűbb programunk (mint pl. a "Hello world") is egy osztály része kell hogy legyen.

A forráskód lefordításából egy ún. Java bájtkódot generál le nekünk, amelyet a JVM (Java Virtuális Gép) interpreter kezel. Ennek köszönhető a Java platformfüggetlensége, amely a c++-al szemben hatalmas segítség legfőképpen a kezdő programozóknak. Ezért viszont a Java hatalmas árat kellett hogy fizessen a sebesség terén, aminek eredményeként a játékfejlesztők szívesebben választják a sebességet a platformfüggetlenséggel kapcsolatos bajlódás helyett, aminek minden gamer nagyon örül. A JVM ezt azzal igyekszik kompenzálni, hogy közvetlenül futtatás előtt platformfüggő gépi kódra fordítja ([JIT](#)) a bájtkódot, ezzel búcsút mondva az interpretálásból adódó időveszteségeknek.

```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello world!");
    }
}
```

Ez a nagyon egyszerű és rövid kód elárul néhány fontos követelményt a c++-al szemben. Abban meggyeznek, hogy a `main` metódus fut le először, de mivel minden osztály és objektum, így a `main` metódus hívásához szükség lenne először a `HelloWorld` osztály egy példányának létrehozására. Hogy ezt elkerüljük, a `main` metódust minden esetben a `static` kulcsszóval egészítjük ki. C++-ban elegendő volt a `main` kulcsszóval megkezdeni a programot, ami alapértelmezetten `int` visszatérési típusat kapott, ezzel ellentétben a Javában minden metódusnak kötelező megadnunk a visszatérési típusát, ami esetünkben a `void` típus lesz. A következő szabály a `String[] args` argumentumok megadása, amely a program

paramétereinek használatát teszi lehetővé. Ami a kódból nem adódik, de mégis fontos tudnivaló, hogy a Java forráskódunk neve jó ha megegyezik az osztályunk nevével.

A metódusnév túlterhelésének a metódusnév többszörös használatát nevezük, amelynek segítségével egy metódusnév eltérő számú és/vagy típusú paraméterekkel is meghívható. Erre Javában és c++-ban egyaránt van lehetőség. Példák:

```
public class Vet{  
    void Cure(Dog d) {...}  
    void Cure(Cat c) {...}  
    void Cure(Cat c, String symptom) {...}  
    ...  
}  
...  
myDog = new Dog(); myCat = new Cat();  
myVet = new Vet();  
...  
myVet.Cure(myDog); myVet.Cure(myCat);  
myVet.Cure(myCat, "coughing");  
...
```

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

A Python egy magas szintű, általános célú, dinamikus, objektumorientált és platformfüggetlen szkriptnyelv, amely 1990-ben alakult. A program futtatásához interpretert használ megspórólva nekünk a forráskód fordításának idejét. Az interpreter a PC-s rendszereken kívül - Windows, Unix, MacOS - az iPhone mobil operációs rendszeren is elérhető.

A fordítóprogramon kívül a zárójeleket és pontosvesszőket is elhagyja a nyelv, ezek helyett a behúzások jelölik a parancsok helyét, így a forráskód átláthatóbb és rövidebb, mint ha azt c++-ban vagy Java-ban írtuk volna. Viszont érdemes észben tartani, hogy ebből kifolyólag a kód nem kezdődhet behúzással. Az interpreter ún. tokenekre bontva értelmezi a sorokat, amelyek között tetszőleges mennyiségű szóköz (whitespace) lehet. A token lehet azonosító, kulcsszó, operátor, delimeter vagy literál. Azonosítók használatakor figyeljünk oda arra, hogy a Python nagybetűérzékeny. Ezen kívül számokat és alulvonásokat (_) is használhatunk.

Pythonban a változók objektumokként vannak kezelve. A változó típusát a benne tárolt adat típusa határozza meg, amelyet az interpreter állapít meg futás közben. Így lehetőség van arra, hogy egy változóhoz idővel különböző típusú adatokat rendeljünk. A lehetséges adattípusok: számok, sztringek, ennesek (tuples, n-es), listák, szótárak (dictionaries).

A számok 3 nagy típusa az egészek, lebegőpontosak és a komplex számok. Az egészek tovább bonthatók decimálisakra, oktalísakra (0-val kezdődik) és hexadecimálisokra (0x-el kezdődik). A lebegőpontos számok az x.yEz formulát követik. A szekvenciátípusok közétartoznak a sztringek, ennesek és a listák, de lehetőség van saját szekvenciátípus létrehozására is. Sztringek megadásánál az idézőjel és az aposztróf egyaránt működik, az "u" betű segítségével pedig Unicode szöveggel bővíthetjük a sztringünket:

```
print u"Hello %s, kedves %s!" ("Pisti", "barátunk")
# A kimeneten: "Hello Pisti, kedves barátunk!"
```

Megjegyzés készítéséhez a # jelet használja, amely a sor végéig érvényes. A megjegyzésblokk viszont 3 egymás utáni idézőjelel kezdődik és végződik. Az ennek (tuples) objektumok halmaza, egymástól vesszővel elválasztva zárójelbe írjuk. A benne szereplő objektumok típusa eltérő is lehet. Példa:

```
('a', 'b', 'c')
tuples('abc') #ugyanaz mint az előző
```

DRAFT

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG> (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPORG repó: source/labor/polargen)

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/arroway/PolárGenerátor.java

bhax/thematic_tutorials/bhax_textbook/arroway/polargen.cpp

Ennek a feladatnak a megoldásához nincs szükségünk a matematikai háttérre, mivel csak egy osztályt kell írnunk, amely képes kezelni a már megtervezett számításokat. A feladatunk egyszerű: van egy számítás - amivel igázából nem sokat kell foglalkoznunk, mert csak lemásoljuk az okos emberek által kitalált képleteket -, amihez írnunk kell egy osztályt, amely képes azt kezelní.

```
boolean nincsTárolt = true;
double tárolt;

public double következő() {

    if(nincsTárolt) {

        // számítások
        ...

        tárolt = egyik_eredmény;
        nincsTárolt = !nincsTárolt;

        return másik_eredmény;
    } else {
}
```

```
        nincsTárolt = !nincsTárolt;
        return tárolt;
    }
}
```

A legfontosabb rész el is készült. Ez az egész egy osztályban lesz, ezért definiáljuk a nincsTárolt és tárolt mezőket, hogy azok a metódusunkon kívül is megtartsák értéküket. A függvényünk első hívásával, mivel még nincs egy tárolt érték se, generálunk 2 értéket és az egyiket eltároljuk, a másikat pedig visszaadjuk. A függvény második hívásakor viszont már nem generálunk semmit, csak visszaadjuk az eltárolt értéket és a nincsTárolt változót újra hamisra állítjuk.

A main függvényben az osztály példányosítása után a tesztelés és demonstráció kedvéért meghívjuk 10-szer a metódusunkat, ami ugyebár csak 5-ször (minden második ciklusban, kezdve az elsőtől) fog számításokat végezni és szintén 5-ször visszaadjja az eltárolt eredményeket.

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

    PolárGenerátor g = new PolárGenerátor();

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

}
```

A teljes java forráskódot megleled a "megoldás forrása"-inál. Most inkább nézzük meg ennek a c++-os verzióját. Amint láthatjuk, lényegében ugyanaz, csupán a kiíratáshoz és a randomizáláshoz szükséges részek térnek el, meg abban, hogy a main az osztályon kívül van, mivel ez c++, és nem java.

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

class PolarGenerator{

    bool nincsTarolt = true;
    double tarolt;

    public :
    double következő()
    {

        if(nincsTarolt)
        {

            double u1,u2,v1,v2,w;
            do{

                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;

}

elsepublic static void main(String[] args) {

PolárGenerátor g = new PolárGenerátor();

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

}

{
    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;
}
```

12.2. Homokózó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciakat kell kiirtani és minden másik működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Serveltbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/arroway/BinfaServlet.java

bhax/thematic_tutorials/bhax_textbook/arroway/Lzw.java

A java nyelvnek van egy olyan tulajdonsága, miszerint minden referencia és így nem kell azon gondolkodnunk, hogy most pointert vagy referenciát használunk. Ennek köszönhetően a c++-os verzióból nyugodtan eltávolíthatjuk a pointereket jelző "*" -ot és a program ugyanúgy fog működni.

```
...
private Csomopont balNulla = null;
private Csomopont jobbEgy = null;
...
private Csomopont fa = null;
...
protected Csomopont gyoker = new Csomopont('/');
...
```

A fejfájás elkerülése érdekében a következő lépés lehet a ";" -k eltávolítása az osztályok és metódusaik végéről. Ezután a kiíratásokhoz, valamint a fájlkezeléshez szükséges metódus / típus cserék elvégeztével kész is a javás verzió (lásd: Lzw.java a megoldás forrásánál)

A Servletbe való beépítéshez szükségünk lesz egy szerver létrehozására a saját gépünkön. Ezt a Tomcat tette lehetővé számomra. A telepítéshez segítséget találsz [itt](#).

Mivel a programunk jelenleg fájlból olvassa be az adatokat és a végeredményt is egy fájlba menti, a kód ezen részét elhagyjuk és csupán a LZWBinFa osztályt hagyjuk meg, amelyet egy Servletbe fogunk beilleszteni.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class BinfaServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        String fa = request.getParameter("fa");

        LZWBinFa binFa = new LZWBinFa();

        boolean kommentben = false;
        byte[] bFa = fa.getBytes();
        for (int i = 0; i < bFa.length; i++) {

            if (bFa[i] == 0x3e) {
                kommentben = true;
                continue;
            }

            if (bFa[i] == 0x0a) {
```

```
kommentben = false;
    continue;
}

if (kommentben) {
    continue;
}

if (bFa[i] == 0x4e) {
    continue;
}

for (int j = 0; j < 8; ++j) {

    if ((bFa[i] & 0x80) != 0) {
        binFa.egyBitFeldolg('1');
    } else {
        binFa.egyBitFeldolg('0');
    }
    bFa[i] <<= 1;

}

}

PrintWriter out = response.getWriter();
out.println("<HTML>");
out.println("<HEAD>");
out.println("<TITLE>Servlet Testing</TITLE>");
out.println("</HEAD>");
out.println("<BODY>");
out.println("<p>");
out.println("fa = " + fa);
out.println("</p>");
out.println("<pre>");
binFa.kiir(out);
out.println("depth = " + binFa.getMelyseg());
out.println("mean = " + binFa.getAtlag());
out.println("var = " + binFa.getSzoras());
out.println("</pre>");
out.println("</BODY>");
out.println("</HTML>");
}
}
```

12.3. „Gagyi”

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/arroway/Gagyi.java

A feltétel összegezve valahogy így hangzik: x egyenlő t és x nem egyenlő t . Első ránézésre azt hinnénk, hogy a ciklus semmilyen esetben sem fog lefutni. Viszont ez nem igaz, mivel a változóknak sem az értéke sem a típusa sincs megadva. Ebben az esetben ha x és t objektumok, akkor az értékük megegyezhet az első kettő feltételenél, de a harmadiknál eltér, mivel azok külön objektumok, külön vannak eltárolva, tehát eltér a referenciajuk. Ekkor viszont ha az értékük ugyanaz, a ciklus végtelen ciklus lesz.

```
/**  
 * Cache to support the object identity semantics of autoboxing for ←  
 * values between  
 * -128 and 127 (inclusive) as required by JLS.  
 *  
 * The cache is initialized on first usage. The size of the cache  
 * may be controlled by the {@code -XX:AutoBoxCacheMax=<size>} option.  
 * During VM initialization, java.lang.Integer.IntegerCache.high ←  
 * property  
 * may be set and saved in the private system properties in the  
 * jdk.internal.misc.VM class.  
 */  
  
private static class IntegerCache {  
    static final int low = -128;  
    static final int high;  
    static final Integer cache[];  
  
    static {  
        // high value may be configured by property  
        int h = 127;  
        String integerCacheHighPropValue =  
            VM.getSavedProperty("java.lang.Integer.IntegerCache.high");  
        if (integerCacheHighPropValue != null) {  
            try {  
                int i = parseInt(integerCacheHighPropValue);  
                i = Math.max(i, 127);  
                // Maximum array size is Integer.MAX_VALUE  
                h = Math.min(i, Integer.MAX_VALUE - (-low) - 1);  
            } catch( NumberFormatException nfe) {  
                // If the property cannot be parsed into an int, ignore ←  
                // it.  
            }  
        }  
    }  
}
```

```
high = h;

cache = new Integer[(high - low) + 1];
int j = low;
for(int k = 0; k < cache.length; k++)
    cache[k] = new Integer(j++);

// range [-128, 127] must be interned (JLS7 5.1.7)
assert IntegerCache.high >= 127;
}

private IntegerCache() {}

/** 
 * Returns an {@code Integer} instance representing the specified
 * {@code int} value. If a new {@code Integer} instance is not
 * required, this method should generally be used in preference to
 * the constructor {@link #Integer(int)}, as this method is likely
 * to yield significantly better space and time performance by
 * caching frequently requested values.
 *
 * This method will always cache values in the range -128 to 127,
 * inclusive, and may cache other values outside of this range.
 *
 * @param i an {@code int} value.
 * @return an {@code Integer} instance representing {@code i}.
 * @since 1.5
 */
@HotSpotIntrinsicCandidate
public static Integer valueOf(int i) {
    if (i >= IntegerCache.low && i <= IntegerCache.high)
        return IntegerCache.cache[i + (-IntegerCache.low)];
    return new Integer(i);
}
```

Ha megnézzük a JDK-ban az Integer.java forráskódját, akkor az eddigi logikánk nem helyes, ugyanis az Integer objektum létrehozásakor a -128 és 127 közötti értékek egy már előre létrehozott, cache-elt referenciát kapnak. Tehát ha létrehozunk 2 Integer típusú objektumot ugyanazzal az értékkel, akkor -128 és 127 közötti érték esetén a harmadik feltétel hamis lesz, mivel ugyanarra a cahce-elt értékre mutatnak, egyébként viszont a ciklus végtelen ciklus lesz.

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

    Integer x = 128; // 127
    Integer t = 128; // 127

    while (x <= t && x >= t && t != x) {
        System.out.print("végtelen ciklus");
    }
}
```

```
        System.out.println("vége...");  
    }
```

12.4. Yoda

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

https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/arroway/Yoda.java

A Yoda condition egy programozási stílus, aminek az elnevezését Yoda nem hagyományos szórendje ihletett. Ez a programozásban úgy nyilvánul meg, hogy a feltételek írásakor a konstans értéket írjuk a feltétel bal oldalára.

```
// with Yoda condition (const, variable)  
if(36 = x) {} // syntax error  
  
//without Yoda condition (variable, const)  
if(x = 36) {} // logical error
```

Ennek az az előnye, hogy ezt szokássá alakítva elkerülhetünk néhány logikai hibát, mint pl. ha 2 egyenlőségjel helyett csak 1-et írunk, akkor a fordító nem jelez hibát, viszont a program nem úgy fog működni, ahogy azt mi szeretnénk. Ugyanez igaz akkor is, ha String-eket hasonlítunk össze nem Yoda módszerrel és a változónk null értékű, de mi csak annyiról értesülünk, hogy a két szöveg nem egyezik meg.

```
public class Yoda{  
  
    public static void main(String[] args){  
  
        String yc = null;  
  
        // with Yoda condition  
        /*  
         * if( "true".equals(yc) ){  
         *     System.out.println("Yoda's power!");  
         * } */  
  
        // without Yoda condition  
        if( yc.equals("true") ) {}  
  
    }  
  
}
```

Ha ezt lefordítjuk és futtatjuk, akkor a programunk természetesen java.lang.NullPointerException-vel leáll.

```
§ java Yoda
Exception in thread "main" java.lang.NullPointerException
at Yoda.main(Yoda.java:14)
```

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 (mert ha csak lemasolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Megoldás videó:

Megoldás forrása:

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/lisk_hely/LiskSert.java

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/lisk_hely/LiskRaFi.java

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/lisk_hely/lisks.cpp

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/lisk_hely/liskrf.cpp

Liskov elve szerint egy típushierarchia szülő osztályokból és származtatott osztályokból áll úgy, hogy a származtatott osztályok rendelkeznek a szülő osztály összes tulajdonságával és viselkedésével plusz valami extrával. Hozzunk létre egy osztályt az autóknak, amely tartalmazza egy autó általános jellemzőit. Ezután származtassuk belőle az Audi osztályt. Ezután a `drive` metódusnak átadunk egy Car vagy Audi típusú objektumot. minden jól megy, az autókat tudjuk vezetni, de a gyerekünknek is megtetszik az autózás, ezért a Car-ból származtatva létrehozzunk egy játék autó (ToyBMW) osztályt, hogy legyen mivel játszania. Előbb vagy utóbb észrevesszük, hogy az általunk kreált osztályhierarchia nem reális, hiszen egy játék autót nem lehet vezetni. Megsértettük a Liskov elvet.

```
public class LiskSert {  
  
    void drive(Car c) {  
        c.go();  
        //...  
    }  
}
```

```
public static void main(String[] args) {  
    Car c = new Car();  
    drive(c);  
  
    Audi a = new Audi();  
    drive(a);  
  
    ToyBMW tbmw = new ToyBMW();  
    drive(tbmw); // this is a problem, because you can't drive ←  
                 drive a toy car  
}  
  
class Car{  
    void go() {}  
  
class Audi extends Car{  
    // no problem, because you can drive a car  
  
class ToyBMW extends Car{  
    // but you can't drive a toy car
```

Hogy helyrehozzuk a hibákat, beiktatunk egy köztes RealCar osztályt a Car és az Audi és egyéb vezethető autók közé.

```
public class LiskRaFigyel{  
  
    void drive(RealCar rc) {  
        rc.go();  
        //...  
    }  
  
    public static void main(String[] args) {  
  
        Audi a = new Audi();  
        drive(a);  
  
        //ToyBMW tbmw = new ToyBMW();  
        //drive(tbmw);  this is a problem, because you can't drive ←  
        //               drive a toy car  
  
    }  
  
}
```

```
class Car{ }

class RealCar extends Car{
    void go() {}
}

class Audi extends RealCar{
    // no problem, because you can drive a car
}

class ToyBMW extends Car{
    // but you can't drive a toy car
}
```

13.2. Szülő-gyerek

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/szulo_gyerek/java/Osztaly.java

bhax/thematic_tutorials/bhax_textbook/liskov/szulo_gyerek/c++/szulogyerek.cpp

Javában: létrehozunk egy osztályt, ami a "szülő" osztály lesz és létrehozzuk a `kiir` metódust. Származtatunk belőle egy "gyerek" osztályt. Itt túlterheljük az ősosztály `kiir` metódusát és létrehozunk egy új (`ujKiir`) metódust is. A tartalmuk lényegtelen, mivel minden egyedül a metódusok elérhetősége érdekel. Ezután jöhet is a tesztelés, ahol létrehozunk egy "gyerek" típusú objektumot, amire egy "szülő" típusú referencia mutat:

```
public class Osztaly{

    public static void main(String[] args) {
        Szulo sz = new Gyerek();
        sz.kiir();
        sz.ujKiir();
    }
}

class Szulo{
    public void kiir(){
        System.out.println("Szulo");
    }
}
```

```
class Gyerek extends Szulo{  
  
    public void kiir(){  
        System.out.println("Gyerek");  
    }  
  
    public void ujKiir(){  
        System.out.println("uj");  
    }  
  
}  
  
$ javac Osztaly.java  
Osztaly.java:6: error: cannot find symbol  
    sz.ujKiir();  
           ^  
      symbol:   method ujKiir()  
      location: variable sz of type Szulo  
1 error
```

A fordítás sikertelen, ugyanis a Szulo nem ismeri a Gyerek által létrehozott új metódust. Viszont ha a túlterhelt metódust próbáljuk elérni, akkor az eredmény:

```
$ java Osztaly  
Gyerek
```

Ez a dinamikus kötésnek köszönhető, amely futási időben rendeli hozzá a metódust az objektumhoz. Java-ban ez alapértelmezett. Statikus kötést (fordítás során történik) is elérhetünk ha a static kulcsszóval deklaráljuk. Ekkor viszont a szülő metódusa fog lefutni.

13.3. Anti OO

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

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/antioo/PiBBPBench.java

bhax/thematic_tutorials/bhax_textbook/liskov/antioo/PiBBPBench.cs

bhax/thematic_tutorials/bhax_textbook/liskov/antioo/pi_bbp_bench.c

A BBP algoritmusnak jóvoltából képesek vagyunk másodpercek alatt kiszámoltatni a Pi hexadecimális kifejtésének 0. pozíciótól számított szinte akárhány darab jegyét. Viszont minnél távolabbi értékeket vizsgálunk, annál inkább számít a teljesítmény. Először vizsgáljuk meg a java verziót. A d+1. hexa jegytől számoljuk a hexa jegyeket, amely deklarálása a for ciklus fejlécében szerepel.

```
...
int jegy = 0;

long delta = System.currentTimeMillis();

for(int d=1000000; d<1000001; ++d) {

    d16Pi = 0.0d;

    d16S1t = d16Sj(d, 1);
    ...

$ java PiBBPBench
6
1.803

$ java PiBBPBench
7
21.935

$ java PiBBPBench
12
243.427
```

* A számításokat az alábbi erőforrásokkal felszerelt gépen végeztem:
OS: Ubuntu 18.04 Lts
CPU: intel core i5 6300HQ
RAM: 8 GB 2133 MT/s DDR4

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.

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/smnist3/ScaleAdapt.java

bhax/thematic_tutorials/bhax_textbook/liskov/smnist3/SMNISTE3Activity.java

bhax/thematic_tutorials/bhax_textbook/liskov/smnist3/SMNISTSurfaceView.java

bhax/thematic_tutorials/bhax_textbook/liskov/smnist3/SurfaceEvents.java

A projekt felélesztésében tutorált: Racs Tamás

Mivel csupán a forrásfájlokkal rendelkeztünk, létre kellett hoznunk egy üres Android Studio projektet. Egy kis módosítással a `AndroidManifest` fájlban és a forrásfájlok bemásolásával sikerült feléleszteni a projektet. Ehhez szükség volt az Android Studio és a szükséges Android SDK telepítéséhez, amihez **itt** találsz segítséget.

A színvilág változtatásához a `SMNISTSurfaceView.java` fájlt kell átírni. Az eddigi megoldást, ahol 2 háttérszín váltakozott ciklusonként...

```
/*int[] bgColor =
{
    android.graphics.Color.rgb(11, 180, 250),
    android.graphics.Color.rgb(11, 250, 180)
};*/
//int bgIdx = 0;
...
//bgIdx = (bgIdx + 1) % 2;
...
//canvas.drawColor(bgColor[bgIdx]);
```

..., mostmár csak 1 szín van, így nem zavar a színek váltakozása. A színeket úgy választottam, hogy az kellemes legyen a szemnek és megőrizze a kontrasztot, hogy az ne akadályozza felhasználót a pontok számának meghatározásában.

```
public void onDraw(android.graphics.Canvas canvas) {
    ...
    canvas.drawColor(android.graphics.Color.rgb(25, 0, 51)); ←
        // háttérszín
    ...
}
private void cinit(android.content.Context context) {

    textPaint.setColor(android.graphics.Color.BLACK); // belső ←
        kör színe
    textPaint.setStyle(android.graphics.Paint.Style. ←
        FILL_AND_STROKE);
    textPaint.setAntiAlias(true);
    textPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
    textPaint.setTextSize(50);

    msgPaint.setColor(android.graphics.Color.rgb(192, 192, 192)); ←
        // világosszürke; feliratok színe
    msgPaint.setStyle(android.graphics.Paint.Style. ←
        FILL_AND_STROKE);
    msgPaint.setAntiAlias(true);
    msgPaint.setTextAlign(android.graphics.Paint.Align.LEFT);
    msgPaint.setTextSize(40);

    dotPaint.setColor(android.graphics.Color.WHITE); // belső ←
        pontok színe
    dotPaint.setStyle(android.graphics.Paint.Style. ←
        FILL_AND_STROKE);
```

```
dotPaint.setAntiAlias(true);
dotPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
dotPaint.setTextSize(50);

borderPaint.setStrokeWidth(2);
borderPaint.setColor(android.graphics.Color.BLACK); // a ←
    naracssárba körökben levő számok színe
fillPaint.setStyle(android.graphics.Paint.Style.FILL);
fillPaint.setColor(android.graphics.Color.rgb(240, 102, 0)); ←
    // narancssárba; a számok háttere
...
}
```

Végül az app futtatásához szükség van a telefonodra vagy egy emulátorra. Ha a saját telefonod akarod használni, akkor előbb be kell kapcsolnod az USB debugging-ot a fejlesztői beállításokban.

DRAFT



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 fóliát)!

Megoldás videó:

Megoldás forrása:

DRAFT

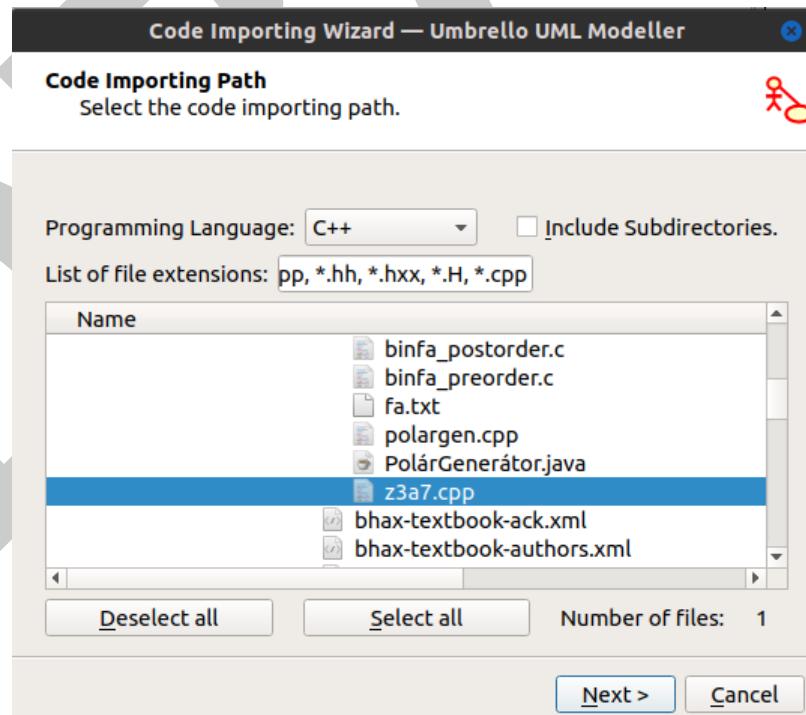
14. fejezet

Helló, Mandelbrot!

14.1. Reverse engineering UML osztálydiagram

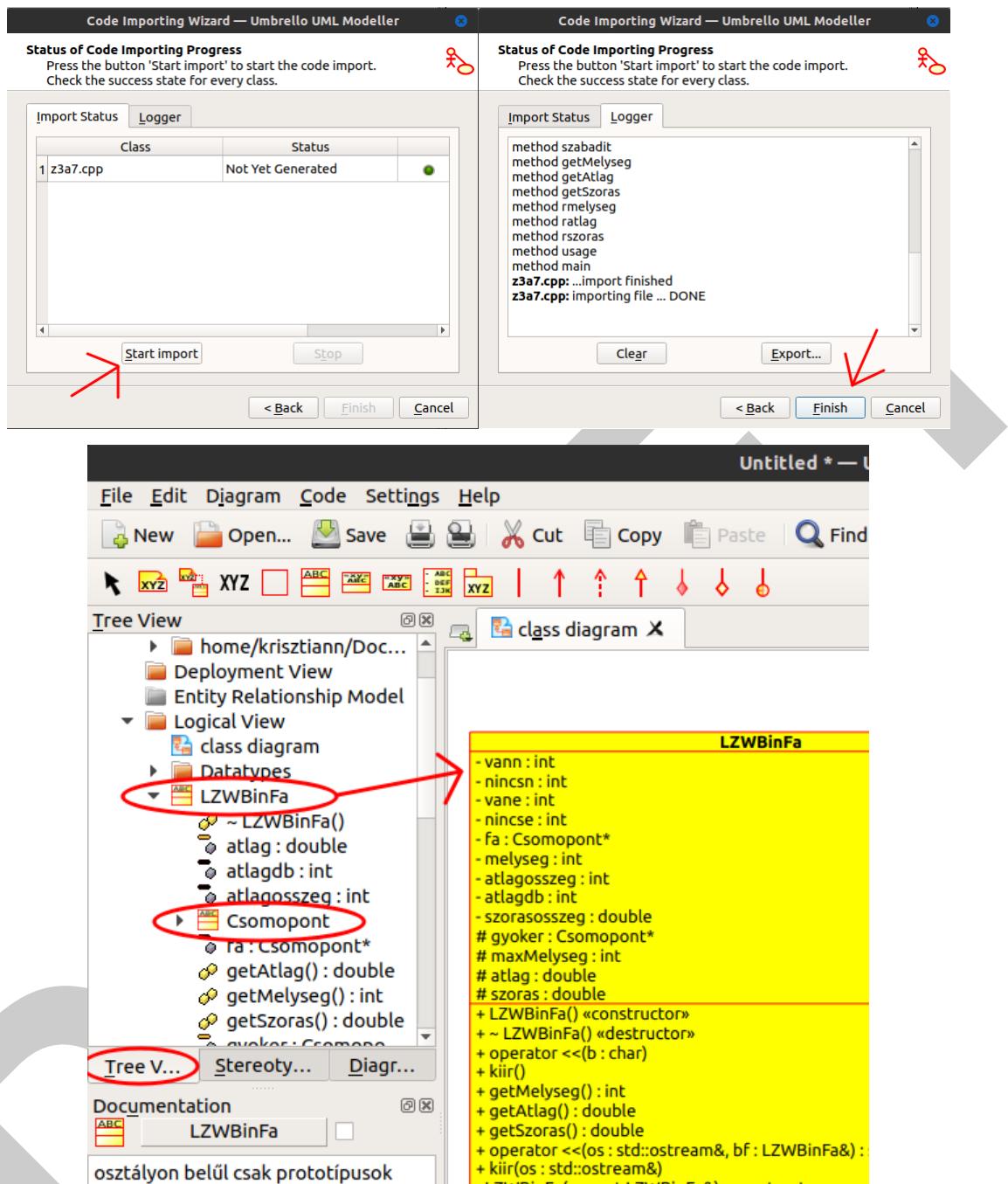
UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nlERIEOs. <https://arato.inf.unideb.hu/batfai.norbert/UD.html>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

Az Umbrello programot választottam, mivel ingyenes és tökéletesen megfelel a feladat igényeinek. Magunktól is elkészíthetnénk az osztálydiagramot, de minek húznánk ezzel az időt, ha a program meg tudja csinálni helyettünk. Nézzük hogyan: A `Code` menüpont alatt a `Code Importing Wizard` opcióról kattintva megjelenik számunkra egy ablak, ahol kiválasztjuk az importálendő forráskód nyelvét, odanavigálunk és kiválasztjuk a forrásfájlt.

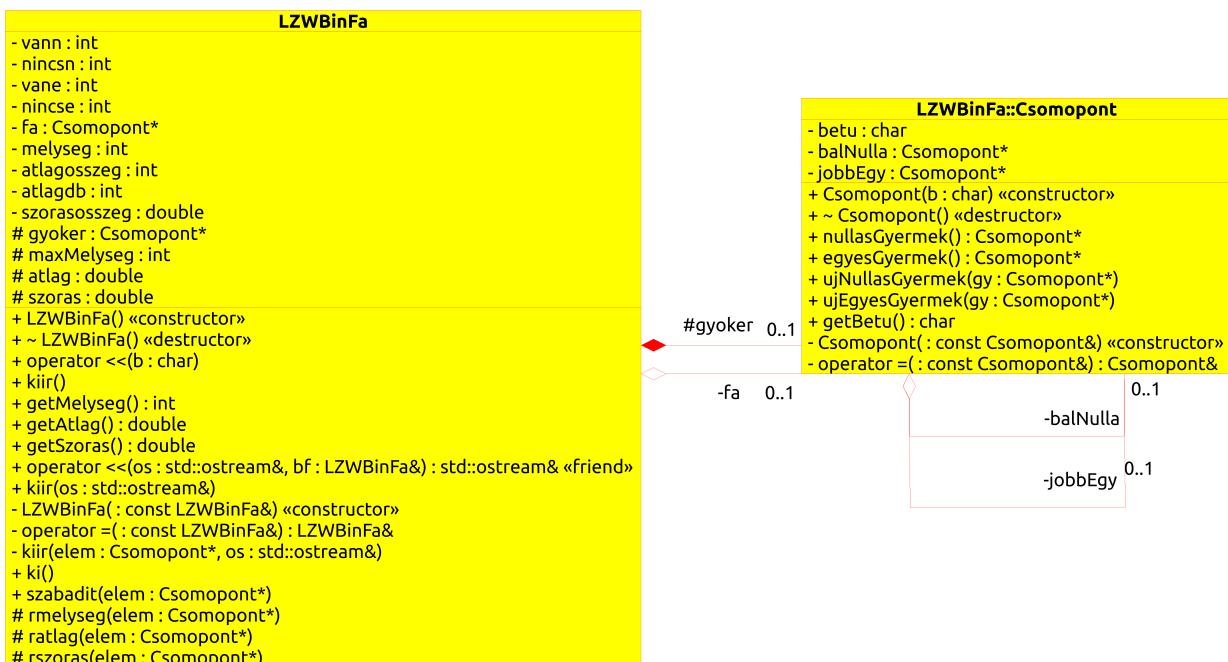


A következő oldalon rá kell mennünk a `Start import` gombra és kész is vagyunk az importálással. Mostmár a `Tree view`-ban megjelent az osztályunk és azt egy sima behúzással megjeleníthetjük, de ne

felejtsük el a Csomopont osztályt is behúzni.



A végeredmény:



A fejléc tartalmazza az osztály nevét, alatta vannak a mezők és úgy a metódusok. Amint azt láthatjuk, a `Csomopont` kompozícióban van az `LZWBinFa`-val, mivel ha kitörölnénk egy `LZWBinFa` objektumot, akkor a hozzá tartozó `Csomopont` objektum is törlődne. Aggregáció esetén ez nem lenne igaz.

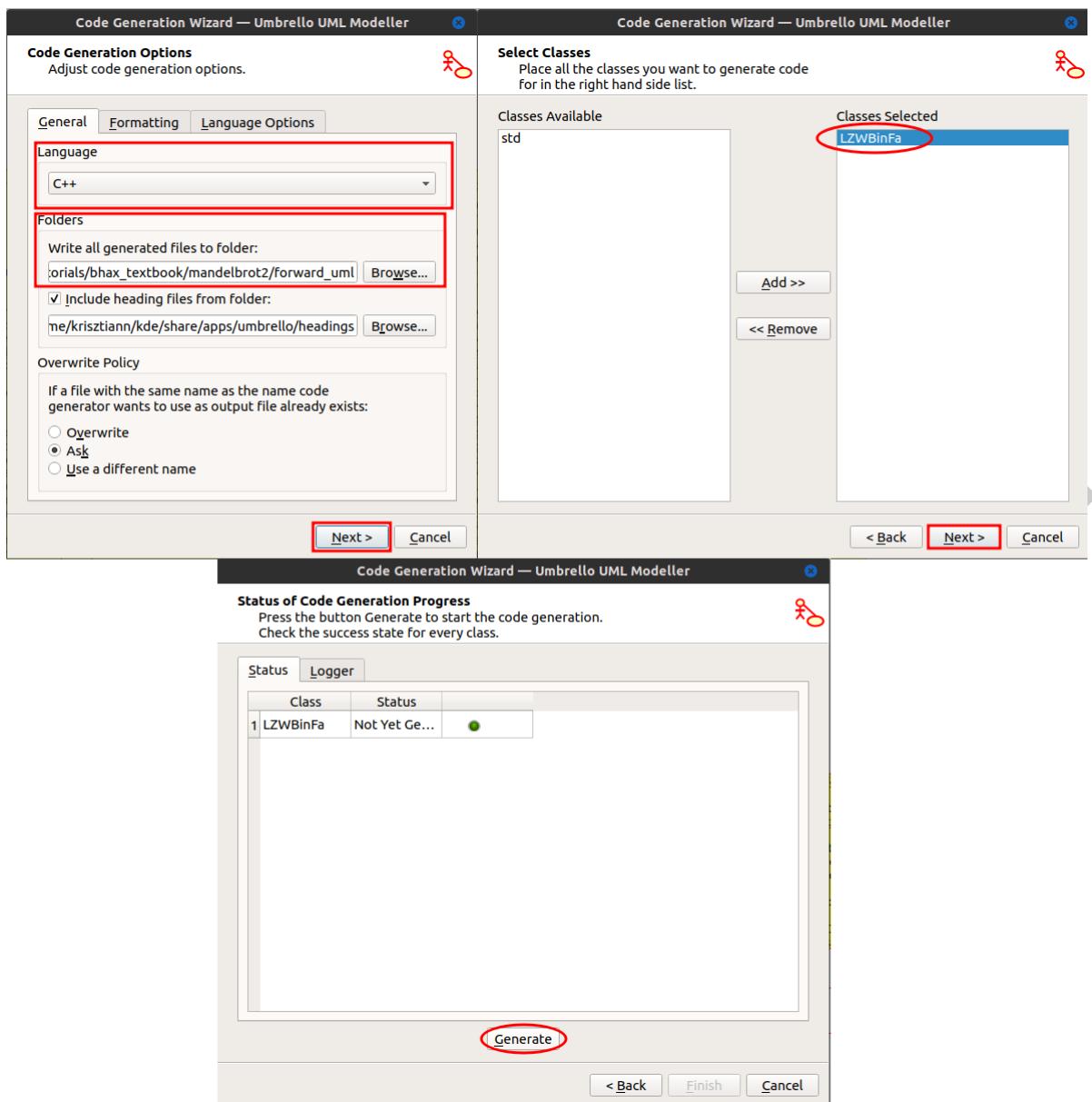
14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/mandelbrot2/forward.uml/LZWBinFa.cpp

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/mandelbrot2/forward.uml/LZWBinFa.h

Most megpróbálunk az előző feladat uml diagramából generálni c++ kódot. Az Umbrello programban a Code menüpont alatt a Code Generating Wizard opcióval megnyílik egy ablak, ahol megadhatjuk a generálandó kód helyét, nyelvét és egyéb formázási beállításokat (pl. akarjuk-e hogy kiegészítse a kódot a kommentjeinkkel vagy sem) hajthatunk végre. Ezután kiválasztjuk legenerálni kívánt osztályokat és egy-két kattintással kész is a forráskód.



A végeredmény egy .cpp és egy .h fájl. A header tartalmazza a mezőket és metódusokat, ám azok üresek, mivel a diagram csupán a metódus nevét, láthatóságát és argumentumait tartalmazza. Ezeken kívül létrehozott get/set metódusokat is a mezők lekérdezéséhez és módosításához.

```
#ifndef LZWBINFA_H
#define LZWBINFA_H

#include string

/**
 * class LZWBinFa
 * osztályon belül csak prototípusok legyenek, <<operátor ←
 * kiiktatása, szeggyük külön
 * a programot 3 részre
 */

class LZWBinFa
```

```
{  
public:  
  
    // Constructors/Destructors  
    //  
  
    /**  
     * Empty Constructor  
     */  
    LZWBinFa ();  
  
    /**  
     * Empty Destructor  
     */  
    virtual ~LZWBinFa ();  
  
    // Static Public attributes  
    //  
  
    // Public attributes  
    //  
  
    // Public attribute accessor methods  
    //  
  
    // Public attribute accessor methods  
    //  
  
    /**  
     * @param b  
     */  
    void operator_ (char b)  
    {  
    }  
  
    /**  
     */  
    void kiir ()  
    {  
    }  
  
    /**  
     * A változatosság kedvéért ezeket az osztálydefiníció (class ←  
     * LZWBinFa {...};) után  
     * definiáljuk,  
     * hogy kénytelen légy az LZWBinFa és a :: hatókör operátorral min ←  
     * ōsítve definiálni  
     * :) l. lentebb  
     * @return int  
     */
```

```
int getMelyseg ()
{
}

/***
 * @return double
 */
double getAtlag ()
{
}

/***
 * @return double
 */
double getszoras ()
{
}

/**
 * @return std::ostream&
 * @param os
 * @param bf
 */
std::ostream& operator_ (std::ostream& os, LZWBinFa& bf)
{
}

/**
 * @param os
 */
void kiir (std::ostream& os)
{
}

/**
 */
void ki ()
{
}

/**
 * @param elem
 */
void szabadit (Csomopont* elem)
{
}

protected:

// Static Protected attributes
```

```
//  
  
// Protected attributes  
//  
  
// A fában tagként benne van egy csomópont, ez erősen ki van ↪  
// tüntetve, ō a gyökér:  
Csomopont* gyoker;  
int maxMelyseg;  
double atlag;  
double szoras;  
public:  
  
// Protected attribute accessor methods  
//  
  
protected:  
  
public:  
  
// Protected attribute accessor methods  
//  
  
/**  
 * Set the value of gyoker  
 * A fában tagként benne van egy csomópont, ez erősen ki van ↪  
 * tüntetve, ō a gyökér:  
 * @param new_var the new value of gyoker  
 */  
void setGyoker (Csmopont* new_var) {  
    gyoker = new_var;  
}  
  
/**  
 * Get the value of gyoker  
 * A fában tagként benne van egy csomópont, ez erősen ki van ↪  
 * tüntetve, ō a gyökér:  
 * @return the value of gyoker  
 */  
Csmopont* getGyoker () {  
    return gyoker;  
}  
  
/**  
 * Set the value of maxMelyseg  
 * @param new_var the new value of maxMelyseg  
 */  
void setMaxMelyseg (int new_var) {  
    maxMelyseg = new_var;  
}
```

```
/**  
 * Get the value of maxMelyseg  
 * @return the value of maxMelyseg  
 */  
int getMaxMelyseg () {  
    return maxMelyseg;  
}  
  
/**  
 * Set the value of atlag  
 * @param new_var the new value of atlag  
 */  
void setAtlag (double new_var) {  
    atlag = new_var;  
}  
  
/**  
 * Get the value of atlag  
 * @return the value of atlag  
 */  
double getAtlag () {  
    return atlag;  
}  
  
/**  
 * Set the value of szoras  
 * @param new_var the new value of szoras  
 */  
void setSzoras (double new_var) {  
    szoras = new_var;  
}  
  
/**  
 * Get the value of szoras  
 * @return the value of szoras  
 */  
double getSzoras () {  
    return szoras;  
}  
protected:  
  
/**  
 * @param elem  
 */  
void rmelyseg (Csomopont* elem)  
{  
}  
  
/**
```

```
* @param elem
*/
void ratlag (Csomopont* elem)
{
}

/**
 * @param elem
 */
void rszoras (Csomopont* elem)
{
}

private:

// Static Private attributes
//

// Private attributes
//

// van nulla
int vann;
// nincs nulla
int nincsn;
// van egy
int vane;
// nincs egy
int nincse;
// Mindig a fa "LZW algoritmus logikája szerinti aktuális" ←
// csomópontjára mutat
Csomopont* fa;
// technikai
int melyseg;
int atlagosszeg;
int atlagdb;
double szorasosszeg;
public:

// Private attribute accessor methods
//

private:

public:

// Private attribute accessor methods
//

/**
```

```
* Set the value of vann
* van nulla
* @param new_var the new value of vann
*/
void setVann (int new_var)  {
    vann = new_var;
}

/**
 * Get the value of vann
* van nulla
* @return the value of vann
*/
int getVann ()  {
    return vann;
}

/**
 * Set the value of nincsn
* nincs nulla
* @param new_var the new value of nincsn
*/
void setNincsn (int new_var)  {
    nincsn = new_var;
}

/**
 * Get the value of nincsn
* nincs nulla
* @return the value of nincsn
*/
int getNincsn ()  {
    return nincsn;
}

/**
 * Set the value of vane
* van egy
* @param new_var the new value of vane
*/
void setVane (int new_var)  {
    vane = new_var;
}

/**
 * Get the value of vane
* van egy
* @return the value of vane
*/
int getVane ()  {
```

```
        return vane;
    }

    /**
     * Set the value of nincse
     * nincs egy
     * @param new_var the new value of nincse
     */
    void setNincse (int new_var) {
        nincse = new_var;
    }

    /**
     * Get the value of nincse
     * nincs egy
     * @return the value of nincse
     */
    int getNincse () {
        return nincse;
    }

    /**
     * Set the value of fa
     * Mindig a fa "LZW algoritmus logikája szerinti aktuális" ←
     * csomópontjára mutat
     * @param new_var the new value of fa
     */
    void setFa (Csomopont* new_var) {
        fa = new_var;
    }

    /**
     * Get the value of fa
     * Mindig a fa "LZW algoritmus logikája szerinti aktuális" ←
     * csomópontjára mutat
     * @return the value of fa
     */
    Csomopont* getFa () {
        return fa;
    }

    /**
     * Set the value of melyseg
     * technikai
     * @param new_var the new value of melyseg
     */
    void setMelyseg (int new_var) {
        melyseg = new_var;
    }
```

```
/**  
 * Get the value of melyseg  
 * technikai  
 * @return the value of melyseg  
 */  
int getMelyseg () {  
    return melyseg;  
}  
  
/**  
 * Set the value of atlagosszeg  
 * @param new_var the new value of atlagosszeg  
 */  
void setAtlagosszeg (int new_var) {  
    atlagosszeg = new_var;  
}  
  
/**  
 * Get the value of atlagosszeg  
 * @return the value of atlagosszeg  
 */  
int getAtlagosszeg () {  
    return atlagosszeg;  
}  
  
/**  
 * Set the value of atlagdb  
 * @param new_var the new value of atlagdb  
 */  
void setAtlagdb (int new_var) {  
    atlagdb = new_var;  
}  
  
/**  
 * Get the value of atlagdb  
 * @return the value of atlagdb  
 */  
int getAtlagdb () {  
    return atlagdb;  
}  
  
/**  
 * Set the value of szorasosszeg  
 * @param new_var the new value of szorasosszeg  
 */  
void setSzorasosszeg (double new_var) {  
    szorasosszeg = new_var;  
}  
/**
```

```
* Get the value of szorasosszeg
 * @return the value of szorasosszeg
 */
double getSzorasosszeg ()  {
    return szorasosszeg;
}
private:

/**
 * @param
 */
LZWBinFa (const LZWBinFa& )
{
}

/**
 * szokásosan: nocopyable
 * @return LZWBinFa&
 * @param
 */
LZWBinFa& operator_ (const LZWBinFa& )
{
}

/**
 * @param elem
 * @param os
 */
void kiir (Csomopont* elem, std::ostream& os)
{
}

void initAttributes () ;

};

#endif // LZWBINFA_H
```

14.3. Egy esettan

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/liskov/antioo/PiBBPBench.java

Feladat: Egy kereskedés számítógép-alkatrészek és számítógép-konfigurációk értékesítésével foglalkozik, Elsődleges feladatunk egy olyan alkalmazás elkészítése, amely lehetőséget biztosít a kereskedés alkatrészeinek és konfigurációinak nyilvántartására. Ennek keretében támogatnia kell a termékek állományból

való betölté- sét, képernyőre történő listázását, állományba való kiírását és az árképzés rugalmas kialakítását.

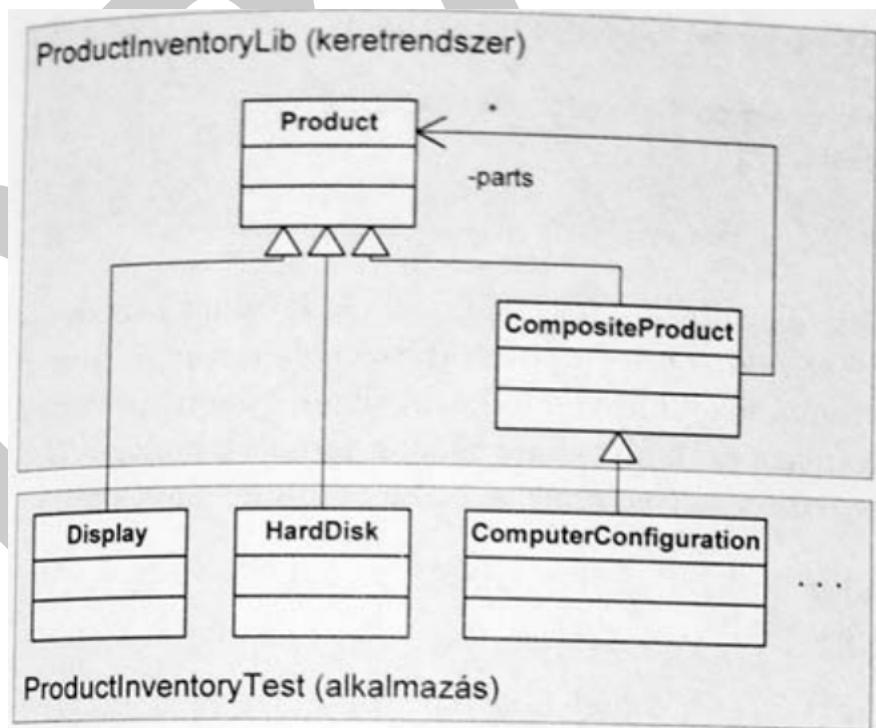
A keretrendszerre vonatkozó követelmények a következők:

- A keretrendszert úgy lehessen felhasználni, hogy ne kelljen kiadni a forráskódját.
- A keretrendszernek támogatnia kell az egyes termékek adatfolyamból való betöltését, adatfolyamba való kiírását és adatfolyamba(jellemzően képernyőre) történő formázott kiírását.
- A termékek fontosabb attribútumai a következők: beszerzési ár, beszerzés dátuma, név és típus. Az aktuális ár a termék attribútumainak függvényében számítandó, a számítás módja fogghat a terméktípustól.
- egy termék lehet elemi vagy összetett. Ez utóbbiak több termékből épülnek fel(például számítógép konfiguráció), amelyek szintén lehetnek elemiek vagy akár összetettek.

Az alkalmazásra vonatkozó követelmények:

- A támogatott termékek a következők (zárójelben az attribútumokkal): kijelző(szélesség és magasság hüvelykben), merevlemezeg egység (sebesség RPM-ben), számítógép-konfiguráció. Az utóbbi összetett termék.

A ProductInventoryLib könyvtár fogja tartalmazni a keretrendszer osztályait, így a csak a lefordított könyvtárra és az osztálydefiníciókat tartalmazó fejlécállományokra van szükség. A Teljes alkalmazás helyett elkészítjük a ProductInventoryTest alkalmazást. A köztük levő kapcsolatokat a következő ábra szemlélteti:



```
// File: product.h
#ifndef PRODUCT_H
#define PRODUCT_H
#include <iostream>
#include <ctime>
class Product
{
protected:
    int initialPrice; //Beszerzési ár
    time_t dateOfAcquisition; // Beszerzés dátuma
    std::string name; // Név
    virtual void printParams(std::ostream& os) const;
    virtual void loadParamsFromStream(std::istream& is);
    virtual void writeParamstoStream(std::ostream& os) const;
public:
    Product();
    Product(std::string name, int initialPrice, time_t
dateOfAcquisition);
    virtual ~Product() {};
    int GetInitialPrice() const;
    std::string GetName() const;
    time_t GetDateOfAcquisition() const;
    int GetAge() const;
    virtual int GetCurrentPrice() const;
    void Print(std::ostream& os) const;
    virtual std::string GetType() const = 0;
    virtual char GetCharCode() const = 0;
    friend std::istream& operator>>(std::istream& is,
Product& product);
    friend std::ostream& operator<<(std::ostream& os,
const Product& product);
};
#endif /* PRODUCT_H */
```

A Product osztály protected tagváltozói az initialPrice(int), dateOfAcquisition(time_t) és a name(std::string). Ezek értékeihez csupán a hozzájuk tartozó get függvényekkel juthatunk hozzá.

```
int GetInitialPrice() const;
std::string GetName() const;
time_t GetDateOfAcquisition() const;
```

Az aktuális dátum és a beszerzési datum alapján a GetAge () visszaadja a termék korát.

```
int Product::GetAge() const
{
    time_t currentTime;
    time(&currentTime);
    double timeDiffInSec = difftime(currentTime, dateOfAcquisition);
```

```
    return (int)(timeDiffInSec/(3600*24));  
}
```

Mivel az aktuális ár kiszámítása termékfüggő, ezért a `GetCurrentPrice()` függvényt virtuálisnak deklártuk, hogy azt majd a leszarmazott felülírja. Ez a `HardDisk` osztály esetén így néz ki:

```
class HardDisk : public Product  
{  
    int speedRPM;  
public:  
    int GetCurrentPrice() const  
    ...  
};  
  
int HardDisk::GetCurrentPrice() const  
{  
    int ageInDays = GetAge();  
    if (ageInDays < 30)  
        return initialPrice;  
    else if (ageInDays >= 30 && ageInDays < 90)  
        return (int)(initialPrice * 0.9);  
    else  
        return (int)(initialPrice * 0.8);  
}
```

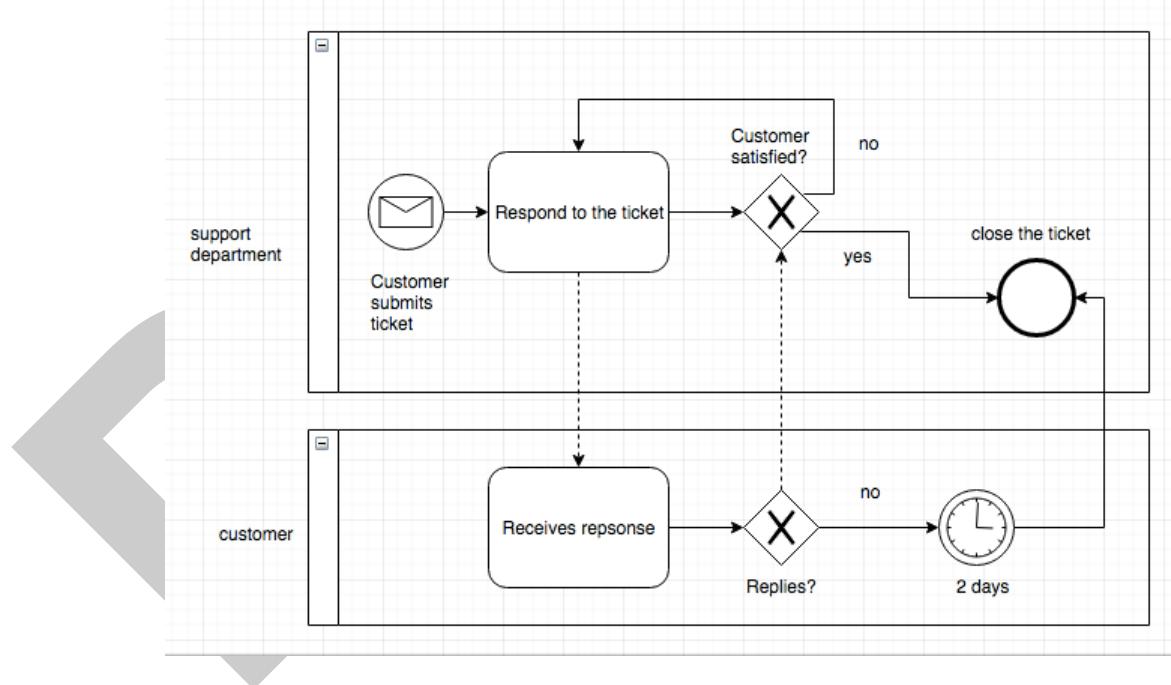
14.4. BPMN

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

A BPMN széles körben használt és elfogadott módja a folyamatok ábrázolásának. Ilyen ábrák készítéséhez a következő eszközök állnak a rendelkezésünkre:

Activity		An Activity is work that is performed within a Business Process
Event		An Event is something that “happens” during the course of a Process
Gateway		A Gateway is used to control the divergence, and convergence of Sequence Flows in a Process
Flow		Two major flow elements are core to BPM: • A Sequence Flow is used to show the order that Activities will be performed in a Process • A Message Flow is used to show the flow of Messages between two Participants of a Process
Message Flow		Represents messages from one process participant to another.
Sequence Flow		Connects flow objects in proper sequential order.
Association		Shows relationships between artifacts and flow objects.

Példa: panaszlevelet küld egy vásárló. A support részleg elküldi válaszát a vásárlónak és vár a visszajelzésre. Ha a vásárló 2 napig nem válaszol, akkor nem foglalkoznak vele többet. Ha viszont válaszol, de válaszában további elégedetlenségét nyilvánítja ki, akkor az ide-oda levelezés tovább folytatódik amíg a vásárló elégedett nem lesz és így le nem zárul az ügy.



14.5. TeX UML

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

Megoldás forrása:

15. fejezet

Helló, Chomsky!

15.1. Encoding

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky2/encoding/MandelbrotHalmazNagyító.java

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky2/encoding/MandelbrotHalmazNagyító.java

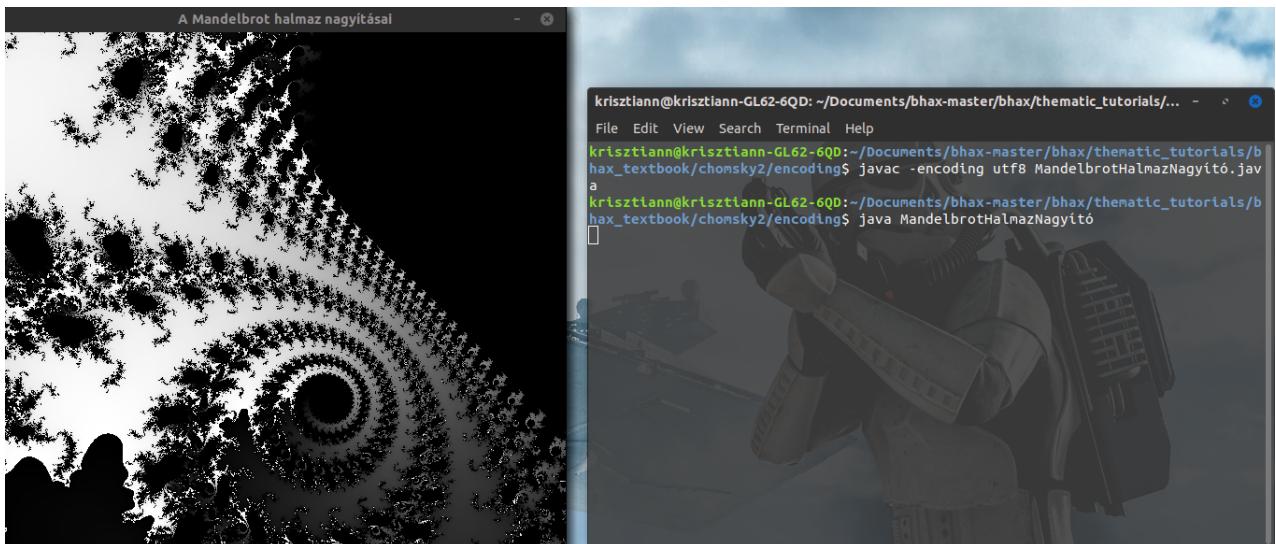
Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky2/encoding/MandelbrotHalmazNagyító.java

Általában nem szokás és nem is ajánlott ékezeteket használnunk a fájlnevekben és a forráskódban (mivel kicsi az esélye annak, hogy a kódodat csak magyarok fogják olvasni és így "mások" nem fogják érteni az azonosítóidat és bizonyos esetben a fordító encode-olási hibába ütközhet a kódod értelmezésekor), de nem árt ha tisztában vagyunk vele, hogy ez is lehetséges.

-encoding encoding Sets the source file encoding name, such as EUC-JP and UTF-8. If the -encoding option is not specified, then the platform default converter is used.

A **javac** parancs után az **-encoding** kapcsolóval megmondhatjuk a fordítónak, hogy melyik encode-olás szerint értelmezze a forráskódunk. Ha ezt nem adjuk meg, akkor az alapértelmezett beállítás szerint végzi el. A támogatott kódolások listáját megtalálod [itt](#).

A fordításhoz az UTF-8-as kódolást használtam, ami biztosítja hogy a fordító értelmezni tudja a magyar nyelv ékezetes betűit.



Ha az alapértelmezett encode-olás a forrásunk kódolásával, akkor az `-encoding` kapcsoló elhanyagolható.

```
krisztiann@krisztiann-GL62-6QD:~/Documents/bhax-master/bhax/thematic_tutorials/... - 
File Edit View Search Terminal Help
Specify where to find annotation processors
-profile <profile>
    Check that API used is available in the specified profile
--release <release>
    Compile for a specific VM version. Supported targets: 6, 7, 8, 9, 10, 11
-s <directory>
    Specify where to place generated source files
-source <release>
    Provide source compatibility with specified release
-source-path <path>, -sourcepath <path>
    Specify where to find input source files
--system <jdk>|none
    Override location of system modules
-target <release>
    Generate class files for specific VM version
--upgrade-module-path <path>
    Override location of upgradeable modules
-verbose
    Output messages about what the compiler is doing
--version, -version
    Version information
-Werror
    Terminate compilation if warnings occur

krisztiann@krisztiann-GL62-6QD:~/Documents/bhax-master/bhax/thematic_tutorials/bhax_textbook/chomsky2/encoding$ javac MandelbrotHalmazNagyító.java
krisztiann@krisztiann-GL62-6QD:~/Documents/bhax-master/bhax/thematic_tutorials/bhax_textbook/chomsky2/encoding$ java MandelbrotHalmazNagyító
krisztiann@krisztiann-GL62-6QD:~/Documents/bhax-master/bhax/thematic_tutorials/bhax_textbook/chomsky2/encoding$ 
```

15.2. OOCWC lexer

Izzítsük be az OOCWC-t és vázoljuk a <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll> lexert és kapcsolását a programunk OO struktúrájába!

15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/chomsky2/pasz_opengl/para6.cpp](https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/carlexer.ll)

A PaRa (Paszigráfia Rapszódia) egy mesterséges nyelv kialakítására való törkvés, amely lehetővé teszi a homunkulusz és a mesterséges homunkulusz közötti kommunikációt, ergo ez az esport kultúra nyelve. Az SMNIST-el ellentétben itt nem a pöttyök száma, hanem azok pontos elhelyezkedése számít.

A vizualizáláshoz az OpenGL könyvtárat fogjuk használni. Az importálást a következő sor teszi lehetővé:

```
#include <GL/glut.h>
```

A színvilág megváltoztatásához a glColor3f() metódusnak kell megadni 3 float típusú értéket, amelyek a vörös, zöld és kék színek (RGB) értékei. Ezeket minden a glBegin() után hívjuk meg, mivel pl.: a glBegin (GL_LINES) után a vonal színét állítjuk amíg azt a glEnd() parancssal be nem zárjuk. Ezek a színezések a drawParaCube() metódusban helyezkednek el.

A háttérszín megváltoztatásához a glClearColor() paramétereit kell átírni, amelyek a 3 RGB + alpha értékek float tipussal. Ez viszont már a draw() metódusban van.

```
void draw ( void )
{
    glClearColor ( .8f, .8f, .8f, .8f );

    ...
}
```

A keyboard() metódust kiegészítettem a fullscreen opcióval, amelyet "f" gombbal tudunk be- és kikapcsolni. A teljes képernyős modba a glutFullScreen() parancssal kerülünk és az ablak újraméretezésével kerülünk ki onnan: glutReshapeWindow(640 , 480) .

```
void keyboard ( unsigned char key, int x, int y )
{
    if ( key == '0' ) {
        index=0;
    } else if ( key == '1' ) {
        index=1;
    } else if ( key == '2' ) {
        index=2;
    } else if ( key == '3' ) {
        index=3;
    } else if ( key == '4' ) {
        index=4;
    } else if ( key == '5' ) {
        index=5;
    } else if ( key == '6' ) {
        index=6;
    } else if ( key == 't' ) {
        transp = !transp;
    } else if ( key == '-' ) {
        ++fovy;

        glMatrixMode ( GL_PROJECTION );
        glLoadIdentity();
        gluPerspective ( fovy, ( float ) w/ ( float ) h, ←
                        .1f, 1000.0f );
        glMatrixMode ( GL_MODELVIEW );

    } else if ( key == '+' ) {
        --fovy;
```

```
glMatrixMode ( GL_PROJECTION );
glLoadIdentity();
gluPerspective ( fovy, ( float ) w/ ( float ) h, ←
    .1f, 1000.0f );
glMatrixMode ( GL_MODELVIEW );

} else if ( key == 'f' ) {

    fullScreen = !fullScreen;
    if (fullScreen) {
        glutFullScreen();
    } else {
        glutReshapeWindow(640 , 480);
    }

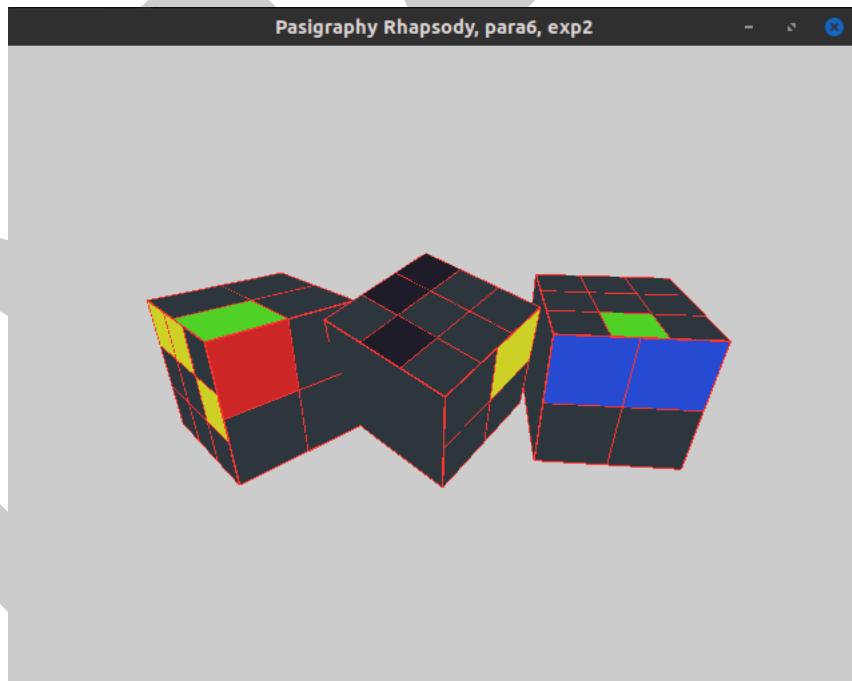
}

glutPostRedisplay();

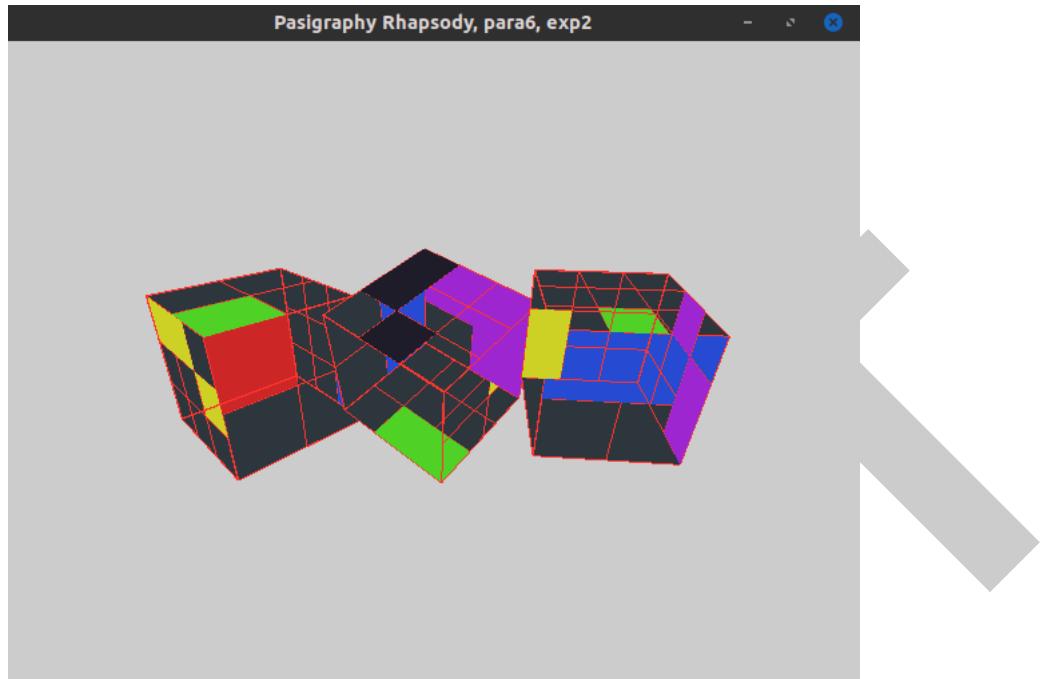
}
```

A program fordításához használt parancs: **g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut**

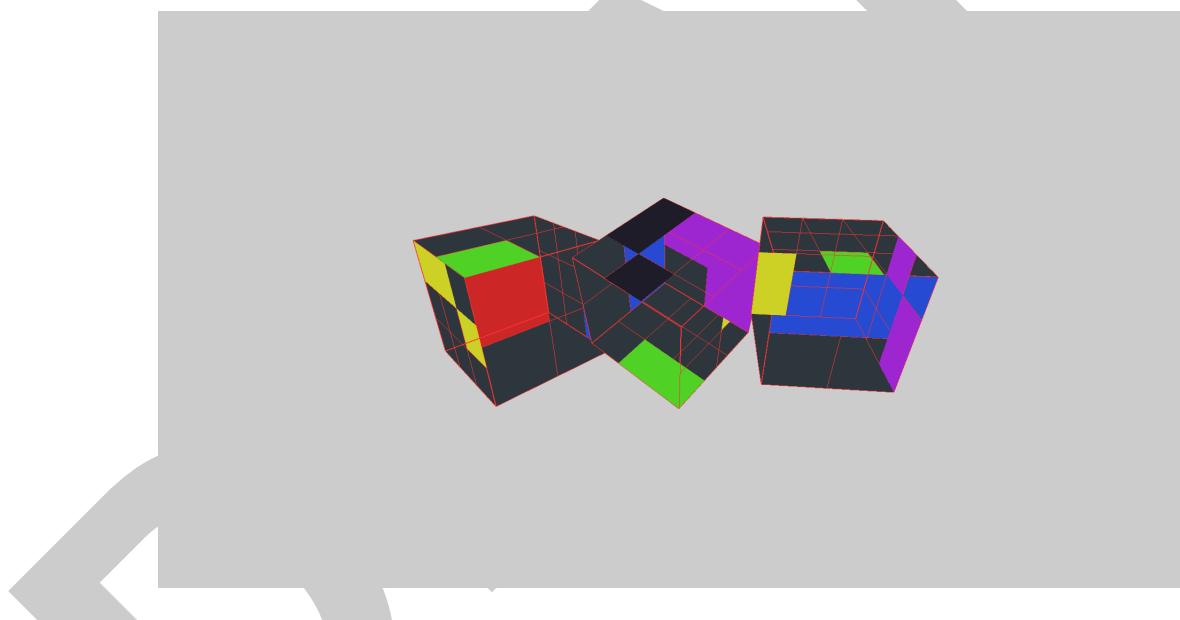
A futtatáshoz pedig: **./para**



Átlátszó módban:



Teljes képernyő módban:



15.4. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás.

15.5. Perceptron osztály

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky2/perceptron_mandel.cpp

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky2/perceptron/main.cpp

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky2/perceptron/mlp.hpp

Gépi tanulásban a perceptron egy bináris osztályozás tanítására használt algoritmus, amely el tudja dönten a bemenetről, hogy az benne van e az adott osztályban vagy sem.

Először lefordítjuk a mandel.cpp-t, hogy aztán létrehozhassuk vele mandel.png-t. Később ez a kép lesz a bemenet a perceptronnak.

```
$ g++ mandel.cpp -lpng16 -O3 -o mandel
$ ./mandel mandel.png
2114
21.1525 sec
mandel.png mentve
```

A libpng és libpng++ könyvtárkat a következő parancsokkal tudjuk feltelepíteni: **sudo apt-get install libpng-dev** és **sudo apt-get install libpng++-dev**. A **-O3** kapcsoló 3-as szintű optimizálást jelent, hogy a program majd gyorsabb legyen, cserébe viszont tovább tart a fordítás.

```
$ g++ mlp.hpp main.cpp -o perc -lpng -std=c++11
$ ./perc mandel.png
0.527632
$ ./perc mandel.png
0.696341
```

Deklaráljuk a képet tartalmazó objektumunkat a parancssori argumentumban megadott fájlnév alapján.

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

Ezután kiszámoljuk és eltároljuk a kép "méretét" (területét), amit felhasználunk a perceptron bemeneti méretének meghatározásához. Ugyígy az image változó deklarálásakor.

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

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

double* image = new double[size];
```

Egyesével végigmegyünk a kép összes pixelén és azok red értékét eltároljuk az image tömbbe, így a ciklusok végére megkapjuk a kép vörös színértékeit egydimenziós formára konvertálva. Erre azért volt szükség, mert a perceptron ilyen formában igényeli a bemenetet.

```
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;
```

Mostmár átadjuk a perceptronnak az értékeket és az eredményt elmentjük a value változóba. Végül már csak a kiíratás és a memória felszabadítása maradt:

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

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

delete p;
delete [] image;

}
```

DRAFT

16. fejezet

Helló, Stroustrup!

16.1. JDK osztályok

Írunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/stroustrup/jdk/jdk_counter.cpp

A feladatban a JDK 11 forrásfájljainak a kilistázását fogjuk most megvalósítani. Ehhez azonban le kell töltenünk az src.zip állományt, mivel ez tartalmazza a Java osztályait. Ubuntu-n a következő parancssal érhetjük ezt el: **sudo apt-get install openjdk-11-source** (különböző java verzió src.zip letöltéséhez elegendő a szám megváltoztatása a parancsban). A "/usr/lib/jvm/openjdk-11/lib/" mappában megtaláljuk a fájlunkat, amit én kicsomagoltam egy másik helyre, hogy az könnyen elérhető legyen a program számára.

A Boost könyvtárból beimportáljuk a fájlkezeléshez szükséges header-eket és az átláthatóbb és egyszerűbb kódolásért névtéret is létrehozunk számára.

```
#include <iostream>
#include <boost/filesystem.hpp>
#include <boost/algorithm/string.hpp>
using namespace boost::filesystem;
```

Definiálunk 2 globális változót: a counter-t long long-nak, hogy biztosan beleférjünk a számtípusok határaiba, ugyanis ide fogjuk tárolni az osztályok számát; az exclude tartalmazza azokat a fájlneveket, amelyek nem Java osztályok. Ezután inicializáljuk a root változót a könyvtár elérési útvonalával. A szöveges útvonalat a Boost könyvtár függvényeinek megfelelő típusúvá konvertáljuk a p változó segítségével, amit át is adunk az Java fájlokat számláló eljárásnak.

```
long long counter;
std::vector<std::string> exclude = {"module-info.java", "package-info.java"};

...

int main() {
```

```
    std::string root = "/home/krisztian/Downloads/javablib/src";
    counter = 0;

    path p (root);
    countJavaFiles(p);
    std::cout << "counted java files: " << counter << std::endl;

}
```

...

```
void countJavaFiles(path pathh) {

    recursive_directory_iterator p(pathh);
    recursive_directory_iterator end;

    while(p != end) {

        std::vector<std::string> result;
        boost::split(result, p->path().string(), boost::is_any_of("/"))
        ;
        std::string j = ".java";

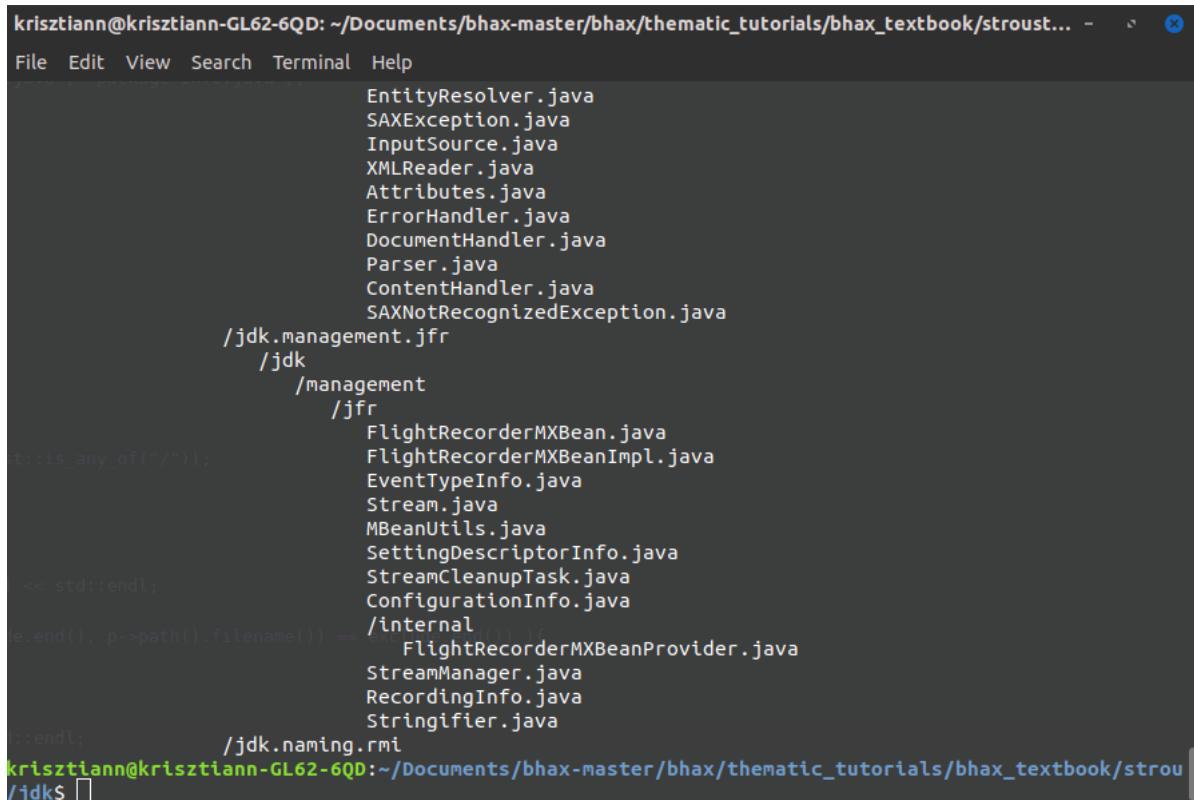
        if( is_directory(p->path()) ) {

            padding(result.size()-1);
            std::cout << "/" << result[result.size()-1] << std::endl;

        } else if( (std::find(exclude.begin(), exclude.end(), p->path() .
        .filename()) == exclude.end()) && (j.compare(p->path() .
        extension().string()) == 0) ) {

            counter++;
            padding(result.size()-1);
            std::cout << result[result.size()-1] << std::endl;

        }
        ++p;
    }
}
```



```
krisztiann@krisztiann-GL62-6QD: ~/Documents/bhax-master/bhax/thematic_tutorials/bhax_textbook/stroust... - ×
File Edit View Search Terminal Help
EntityResolver.java
SAXException.java
InputSource.java
XMLReader.java
Attributes.java
ErrorHandler.java
DocumentHandler.java
Parser.java
ContentHandler.java
SAXNotRecognizedException.java
/jdk.management.jfr
/jdk
/management
/jfr
FlightRecorderMXBean.java
FlightRecorderMXBeanImpl.java
EventTypeInfo.java
Stream.java
MBeanUtils.java
SettingDescriptorInfo.java
StreamCleanupTask.java
ConfigurationInfo.java
/internal
FlightRecorderMXBeanProvider.java
StreamManager.java
RecordingInfo.java
Stringifier.java
d::endl;
/jdk.naming.rmi
krisztiann@krisztiann-GL62-6QD:~/Documents/bhax-master/bhax/thematic_tutorials/bhax_textbook/strou...
/jdk$
```

16.2. Másoló-mozgató szemantika

Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékkadásra!

A c++11 mozgató szemantikája megszabadít minket a felesleges memória foglalásoktól, ami akkor keletkezik, amikor A-t atmasoljuk B-be, de A-ra nincs szükségünk többé.

Például, létrehozzuk az old_house objektumot, meghívódik a konstruktor és lefoglalódik a szükséges memória. Ezután létrehozzuk a new_house objektumot, ekkor meghívódik a másoló konstruktor, ami lefoglalja a ptr-nek szükséges memóriaterületet és ezután átmásolja az old_house értékét.

```
struct A {
    int* ptr;

    A() {
        ptr = new int;
        cout << "Constructor..." << endl;
    }

    A(const A& a1) {
        cout << "Copy const..." << endl;
        this->ptr = new int;
        *this->ptr = *a1.ptr;
    }

    ~A() {
```

```
        cout << "Destructor..." << endl;
        delete ptr;
    }

};

int main() {

    A old_house = A();
    A new_house = old_house;

    return 0;
}

$ ./mozgato
Constructor...
Copy const...
Destructor...
Destructor...
```

De mint amikor új házba költözünk és ahelyett, hogy új bútorokat vennénk, amelyek egyébként pont úgy néznek ki mint a régiek, inkább elhuzzuk őket a régi lakásunkról, ugyanígy megtehetjük a mozgató szemantikával azt, hogy csupán átvesszük az `old_house` `ptr`-jének értékét a két mutató kicserélésével: az új pointer a tárolt értékre mutat, míg a régi pointer sehova.

Ehhez szükség lesz a jobbérték referenciára (jele: `&&`). Mivel az `old_house` balérték referencia, meghívjuk a `move()` függvényt, ami jobbérték referenciát csinál belőle.

```
A(A&& a1) {
    cout << "Move contr..." << endl;
    this->ptr = a1.ptr;
    a1.ptr = nullptr;
}

...
A old_house = A();
A new_house = move(old_house);
...

$ ./mozgato
Constructor...
Move contr...
Destructor...
Destructor...
```

16.3. Hibásan implementált RSA törése

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: https://arato.inf.unideb.hu-batfai.norbert/UDPROG/deprecated/Prog2_3.pdf (71-73 fólia) által készített titkos szövegen.

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky2/pasz_opengl/para6.cpp

```
#include <GL/glut.h>
```

16.4. Változó argumentumszámú ctor

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/stroustrup/ctor/main.cpp

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/stroustrup/ctor/mlp.hpp

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

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

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

```
double* operator() ( double image [] ) // ----- valtozas itt
{
    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
    {

        #ifdef CUDA_PRCPS

            cuda_layer ( i, n_units, units, weights );

        #else

            #pragma omp parallel for
            for ( int j = 0; j < n_units[i]; ++j )
            {
                units[i][j] = 0.0;

                for ( int k = 0; k < n_units[i-1]; ++k )

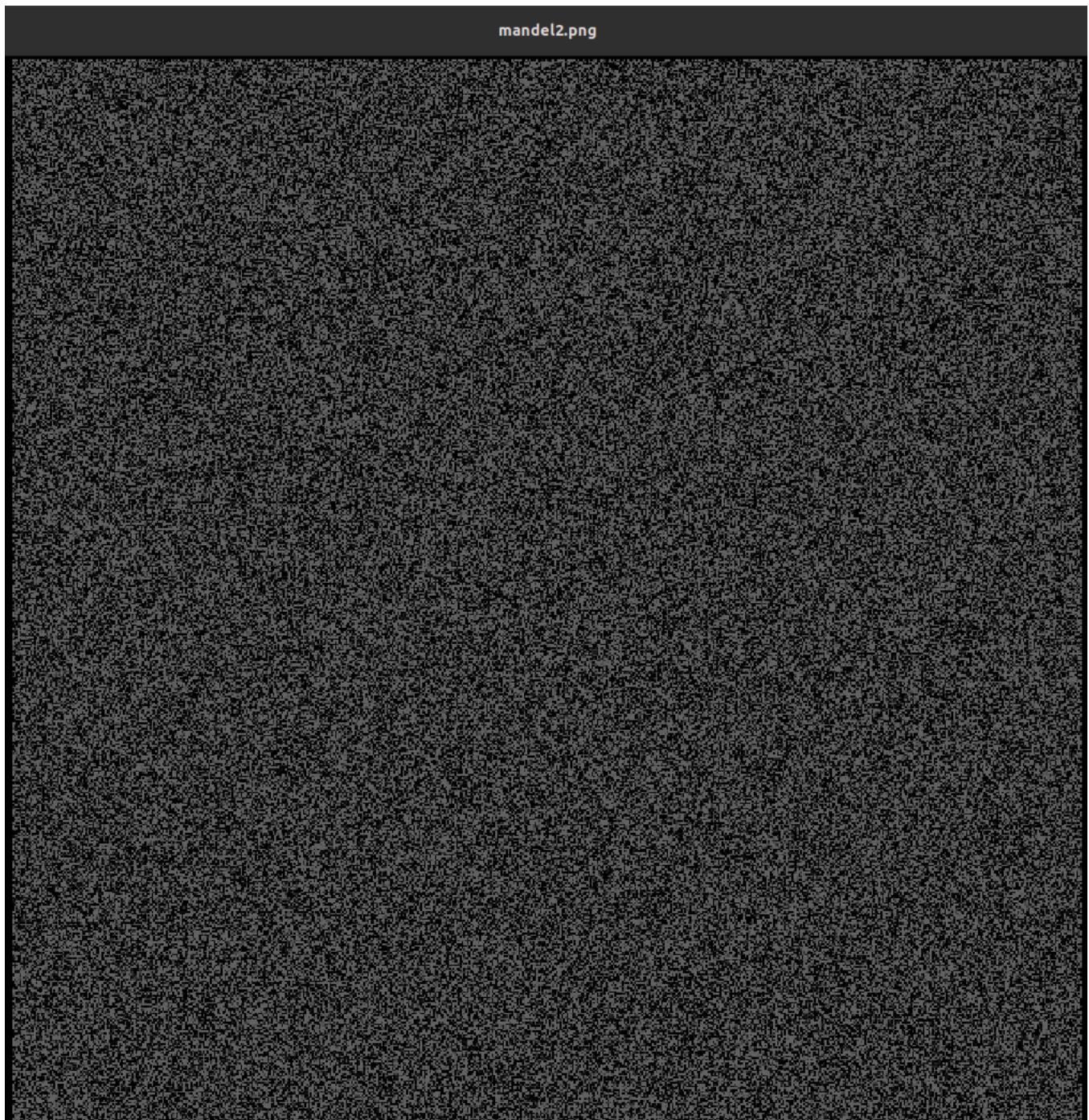
```

```
{  
    units[i][j] += weights[i-1][j][k] * units[i-1][k];  
}  
  
units[i][j] = sigmoid ( units[i][j] );  
  
}  
  
#endif  
  
}  
  
return units[n_layers - 1];//sigmoid ( units[n_layers - 1][0] ) ←  
; ----- es itt  
  
}
```

Összegezve: a mandelbrot halma ról készített kép vörös (red) értékeit átszámoltuk a perceptron osztállyal és most visszaírjuk azokat egy újonnan létrehozott kép (png_image2) red értékeibe. Ha az új kép kék és zöld értékeit az eredeti kép értékeivel pótoljuk, akkor annak egy elszínezett változatát kapjuk.

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





17. fejezet

Helló, Gödel!

17.1. Gengszterek

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

C++11-óta rendelkezésünkre áll az ún. lambda kifejezések létrehozására. Ezek úgy működnek, mint egy névtelen metódus amit ott hozunk létre, ahol éppen szükségünk van rá. Tegyük fel, hogy az egyik metódus egy függvényt kér harmadik paramétereként és ahelyett hogy írnál egy függvénnyt, amire csak egyszer lesz szükséged, megírhatod helyben, névtelenül és a kód továbbra is átlátható marad. Az alábbi példa ezt szemlélteti:

```
// function to count numbers greater than or equal to N
// [=] denotes, can access all variable
int count_N = count_if(v1.begin(), v1.end(), [=](int a)
{
    return (a >= N);
});
```

A lambda kifejezések 3 fő részből állnak:

```
[ captures ] ( params ) { body }
```

A [] zárójelek között megadhatjuk a környezetéből azokat a változókat, amelyeket használni szeretnénk. Elérhetjük őket érték (=) vagy referencia (&) alapján is. Például:

```
[&total, factor]
[factor, &total]
[&, factor]
[factor, &]
[=, &total]
[&total, =]
```

A () zárójelek között definiáljuk a kis függvényünk paramétereit. Ez után jön a visszatérési érték megadása, amely nem kötelező.

```
[] () ->int { }
```

A Robotautó Világ bajnokságban a `sort()` függvény harmadik paramétereként megadtunk egy saját függvényt, amely a szerint fogja rendezni a gengsztereket, hogy azok milyen messze vannak a rendőrtől. Visszatérési értéket nem definiáltunk mert egyértelmű és ilyenkor nincs rá szükség, mert a fordító megállapítja a típust.

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

17.2. C++11 Custom Allocator

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/godel/my_allocator.c](https://bhax.thematic-tutorials.bhax_textbook/godel/my_allocator.c)

A standard allokátor az `std::allocator<T>`, de mi most egy sajátot fogunk írni C-ben.

A memória foglalást az `allocate(size)` függvény végzi. Attribútumként megkapja a tárolni kívánt elemek számát `size_t` típussal, amely az adatok méretét szokta tárolni (és a `sizeof()` függvény visszatérési értéke is ilyen típusú). Az `int` s és `char*` p változókat csupán szemléltetés végett léteznek, hogy kiírassuk az adattípust. A lényeg a `return` kulcsszó után van, ahol a `reinterpret_cast<new_type>(exp)` függvény jelen esetben `<T*>` típusúvá konvertálja amit átadunk neki. Mivel a `char` típus pontosan 1 bájtot foglal, így egy megfelelő méretű `char` típusú tömb pont annyi helyet fog foglalni, amennyire szükségünk van. Ezt úgy számoljuk ki, hogy a típus memóriaigényét megszorozzuk a darabszámmal.

```
using size_type = size_t;
...
pointer allocate(size_type n) {
    int s;
    char* p = abi::__cxa_demangle( typeid(T).name(), 0, 0, &s );
    std::cout << "Allocating "
           << n << " objects of "
           << n * sizeof(T)
           << " bytes. "
           << typeid(T).name() << "=" << p
           << std::endl;
    free(p);
    return reinterpret_cast<T*>( new char[ n * sizeof(T) ] );
}
```

A memória felszabadításához meghívjuk a `delete[]` kulcsszót és utána csinálunk egy kis nyomkövetést:

```
void deallocate(pointer p, size_type n) {
    delete[] reinterpret_cast<char *>(p);
```

```
    std::cout << "Deallocating "
        << n << " objects of "
        << n * sizeof(T)
        << " bytes. "
        << typeid(T).name() << "=" << p
        << std::endl;
}
```

Szemléltetés végett a main() metódusban létrehoztunk egy int típusú vektort és áadtuk neki a saját allokátorunkat, hogy azt használja, valamint feltöltöttük a vektort számokkal 0-tól 14-ig, hogy lássuk, mi is történik. Egy int 4 bájton tárolható és láthatjuk, hogy amint meghaladjuk a lefoglalt memóriát, újra foglalunk, de már 2-szer annyit és a régit felszabadítjuk:

```
$ ./my_allocator
Allocating 1 objects of 4 bytes. i=int
0
Allocating 2 objects of 8 bytes. i=int
Deallocating 1 objects of 4 bytes. i=0x564ff17cbe70
1
Allocating 4 objects of 16 bytes. i=int
Deallocating 2 objects of 8 bytes. i=0x564ff17cc2a0
2
3
Allocating 8 objects of 32 bytes. i=int
Deallocating 4 objects of 16 bytes. i=0x564ff17cbe70
4
5
6
7
Allocating 16 objects of 64 bytes. i=int
Deallocating 8 objects of 32 bytes. i=0x564ff17cc2c0
8
9
10
11
12
13
14
Deallocating 16 objects of 64 bytes. i=0x564ff17cc2f0
```

17.3. STL map érték szerinti rendezése

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

```
#include <GL/glut.h>
```

17.4. Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang Interface Comparable<T> szerepét!

Javában az Interface-ek hasonlóan működnek, mint az osztályok, de az Interface-ekben sosem határozzuk meg a metódusok törzsét, így ez a feladat arra az osztályra hárul, amelyik implementálja azt. Implementálni az implements kulcsszóval lehet és az interface neve után kisebb-nagyobb opeátorok között megadjuk az osztály nevét. C++-al ellentétben, Javában az osztályok egyszerre csupán 1 osztály leszármazottai lehetnek, de interface-ekből többet is implementálhatnak.

```
class Csapat implements Comparable<Csapat> {  
    ...  
}
```

Miután definiáltuk a Csapat osztály mezőit és metódusait, megírjuk a Comparable interface általunk használni kívánt compareTo () függvényének törzsét. A visszatérési érték egy int típusú pozitív vagy negatív szám vagy nulla. Ha a függvényt meghívó objektum erteke valtozója kisebb, mint az argumentumban átadotté, akkor az eredmény -1 lesz, jelezve ezzel, hogy az értékünk kisebb mint akihez hasonlítottuk azt, 1 lesz ha nagyobb értékünk van és 0 ha megegyeznek.

```
class Csapat implements Comparable<Csapat> {  
  
    protected String nev;  
    protected double ertek;  
  
    public Csapat(String nev, double ertek) {  
        this.nev = nev;  
        this.ertek = ertek;  
    }  
  
    public int compareTo(Csapat csapat) {  
        if (this.ertek < csapat.ertek) {  
            return -1;  
        } else if (this.ertek > csapat.ertek) {  
            return 1;  
        } else {  
            return 0;  
        }  
    }  
}
```

17.5. Prolog családfa

Ágyazz be a Prolog családfa programot C++ vagy Java programba! Lásd para_prog_guide.pdf!

18. fejezet

Helló, Anonym!

18.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>
Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

A fordító nem látja a JavaFX-et

18.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/-justine/rcemu/src/carlexer.ll>

18.3. SamuCam

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

```
#include <GL/glut.h>
```

18.4. BrainB

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

A program futtatásához szükség van Qt és opencv telepítéséhez, amit a következő parancsokkal érhetünk el Ubuntu rendszeren:

```
sudo apt-get install libqt4-dev  
sudo apt-get install libopencv-dev
```

Először töltük le a forrásfájlokat a feladatleírásból, majd a következő parancsokkal létrehozzuk, fordítjuk és futtatjuk a projektet:

```
$ qmake -project  
$ qmake BrainB.pro  
$ make  
$ ./BrainB
```

A program 10 percig fut. Addig az a dolgunk, hogy a bal egérgombot nyomva tartva a Samu Entropy-n tartuk az egeret. 10 perc után a program kiírja a statisztikákat.

A forráskódban a slot-signal mechanizmust használtuk. Ez a mechanizmus teljesen független a GUI események ciklusaitól. Ezek különálló void típusú függvények. A slot függvények kezelik a signalokat, amelyeket a connect () függvénytel kapcsolunk össze. Ezért fontos, hogy a signal paraméterlistája megegyezzen az őt fogadó slot paraméterlistájával, ugyanis a signal át kell hogy adja a paraméterként kapott adatait a hozzá kapcsolt slot-oknak.

A signal függvényeket arra használjuk, hogy tudassuk a programmal/figyelővel, hogy változás történt. Az emit kulcsszóval tudjuk meghívni őket. Meghíváskor lefut az összes hozzá kapcsolt slot függvény.

...

Például a BrainB programban összekapcsoltuk a heroesChanged () signalt a updateHeroes () slottal. A connect (sender, signal, receiver, slot) függvénynek először megadjuk a signalt küldő objektumot (ez esetben a brainBThread-ot, és ha megnézzük a forrást, akkor láthatjuk, hogy ő hívja meg a jelet: emit heroesChanged (dest, heroes[0].x, heroes[0].y);), a signalt a paramétereinek típusaival együtt, egy mutatót a slot-ot tartalmazó objektumra (ez esetben this, mert ez a kód a BrainBWin.cpp-ben szerepel, és a slot is itt van definiálva), valamint a slot metódust paramétereinek típusaival együtt.

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ), this ←  
          , SLOT ( updateHeroes ( QImage, int, int ) ) );
```

18.5. OSM térképre rajzolása

Debrecen térképre dobunk rá cuccokat, ennek mintájára, ahol én az országba helyeztem el a DEAC hekkereket: <https://www.twitch.tv/videos/182262537> (de az OOCWC Java Swinges megjelenítőjéből: <https://github.com/emulator/tree/master/justine/rcwin> is kiindulhatsz, mondjuk az komplexebb, mert ott időfejlődés is van...)

IV. rész

Irodalomjegyzék

DRAFT

18.6. Általános

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

18.7. C

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

18.8. C++

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

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

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