

Univerzális programozás

Így neveld a programozód!

Ed. Patrik Kis, DEBRECEN,
2020. szeptember 24, v.
0.0.1

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, 2020, 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>

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Kis, Patrik	2020. november 26.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2020-09-16	0. heti olvasónapló feladat megírásának elkezdése.	pattesz1998
0.0.2	2020-09-26	Prog 1,2 könyv összeállításának tervezett időpontja.	pattesz1998

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]

„Csak kicsi hatást ért el a videójáték-ellenes kampány. A legtöbb iskolában kétműszakos üzemben dolgoznak a számítógépek, értő és áldozatos tanárok ellenőrzése mellett.”

„Minden számítógép-pedagógus tudja a világon, hogy játékokkal kell kezdeni. A játékot követi a játékprogramok írása, majd a tananyag egyes részeinek a feldolgozása.,,

—Marx György, *Magyar Tudomány*, 1987 (27) 12., [MARX]

„I can’t give complete instructions on how to learn to program here — it’s a complex skill. But I can tell you that books and courses won’t do it — many, maybe most of the best hackers are self-taught. You can learn language features — bits of knowledge — from books, but the mind-set that makes that knowledge into living skill can be learned only by practice and apprenticeship. What will do it is (a) reading code and (b) writing code.”

—Eric S. Raymond, *How To Become A Hacker*, 2001., <http://www.catb.org/~esr/faqs/hacker-howto.html>

„I’m going to work on artificial general intelligence (AGI).”

I think it is possible, enormously valuable, and that I have a non-negligible chance of making a difference there, so by a Pascal’s Mugging sort of logic, I should be working on it.

For the time being at least, I am going to be going about it “Victorian Gentleman Scientist” style, pursuing my inquiries from home, and drafting my son into the work.”

—John Carmack, *Facebook post*, 2019., [in his private Facebook post](#)

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket, előadásokat nézzek meg, könyveket olvassak el?	3
II. Tematikus feladatok	5
2. Helló, Turing!	7
2.1. Végtelen ciklus	7
2.2. Lefagyott, nem fagyott, akkor most mi van?	9
2.3. Változók értékének felcserélése	11
2.4. Labdapattogás	11
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	11
2.6. Helló, Google!	11
2.7. A Monty Hall probléma	12
2.8. 100 éves a Brun tétel	12
2.9. Vörös Pipacs Pokol/csigafolytonos mozgási parancsokkal	16
3. Helló, Chomsky!	17
3.1. Decimálisból unárisba átváltó Turing gép	17
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	17
3.3. Hivatkozási nyelv	17
3.4. Saját lexikális elemző	18
3.5. Leetspeak	18

3.6. A források olvasása	20
3.7. Logikus	21
3.8. Deklaráció	22
3.9. Vörös Pipacs Pokol/csigá diszkrét mozgási parancsokkal	25
4. Helló, Caesar!	26
4.1. double ** háromszögmátrix	26
4.2. C EXOR titkosító	28
4.3. Java EXOR titkosító	28
4.4. C EXOR törő	29
4.5. Neurális OR, AND és EXOR kapu	29
4.6. Hiba-visszaterjesztéses perceptron	29
4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve	29
5. Helló, Mandelbrot!	30
5.1. A Mandelbrot halmaz	30
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	31
5.3. Biomorfok	34
5.4. A Mandelbrot halmaz CUDA megvalósítása	38
5.5. Mandelbrot nagyító és utazó C++ nyelven	38
5.6. Mandelbrot nagyító és utazó Java nyelven	38
5.7. Vörös Pipacs Pokol/fel a láváig és vissza	38
6. Helló, Welch!	39
6.1. Első osztályom	39
6.2. LZW	39
6.3. Fabejárás	39
6.4. Tag a gyökér	39
6.5. Mutató a gyökér	40
6.6. Mozgató szemantika	40
6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid	40
7. Helló, Conway!	41
7.1. Hangyaszimulációk	41
7.2. Java életjáték	41
7.3. Qt C++ életjáték	41
7.4. BrainB Benchmark	42
7.5. Vörös Pipacs Pokol/19 RF	42

8. Helló, Schwarzenegger!	43
8.1. Szoftmax Py MNIST	43
8.2. Mély MNIST	43
8.3. Minecraft-MALMÖ	43
8.4. Vörös Pipacs Pokol/javíts a 19 RF-en	44
9. Helló, Chaitin!	45
9.1. Iteratív és rekurzív faktoriális Lisp-ben	45
9.2. Gimp Scheme Script-fu: króm effekt	45
9.3. Gimp Scheme Script-fu: név mandala	45
9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-edén	46
10. Helló, Gutenberg!	47
10.1. Programozási alapfogalmak	47
10.2. C programozás bevezetés	47
10.3. C++ programozás	47
10.4. Python nyelvi bevezetés	47
III. Második felvonás	48
11. Helló, Arroway!	50
11.1. OO szemlélet	50
11.2. Homokózó	52
11.3. „Gagyí”	59
11.4. Yoda	61
11.5. Kódolás from scratch	61
11.6. EPAM: Java Object metódusok	63
11.7. EPAM: Eljárásorientál vs Objektorientált	64
11.8. EPAM: Objektum példányosítás programozási mintákkal	64
12. Helló, Liskov!	65
12.1. Liskov helyettesítés sértése	65
12.2. Szülő-gyerek	66
12.3. Anti OO	68
12.4. Hello, Android!	69

12.5. Hello, SMNIST for Humans!	69
12.6. Ciklomatus komplexitás	69
12.7. EPAM: Interfész evolúció Java-ban	70
12.8. EPAM: Liskov féle helyettesíthetőség elve, öröklődés	70
12.9. EPAM: Interfész, Osztály, Absztrak Osztály	70
13. Helló, Mandelbrot!	71
13.1. Reverse engineering UML osztálydiagram	71
13.2. Forward engineering UML osztálydiagram	72
13.3. Egy esettan	73
13.4. BPMN	74
13.5. TeX UML	75
13.6. EPAM: Neptun tantárgyfelvétel modellezése UML-ben	76
13.7. EPAM: Neptun tantárgyfelvétel UML diagram implementálás	76
13.8. EPAM: OO modellezés	77
14. Helló, Chomsky!	78
14.1. Encoding	78
14.2. OOCWC lexer	79
14.3. l334d1c4 6	79
14.4. Full screen	80
14.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció	82
14.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció	83
14.7. Perceptron osztály	83
14.8. EPAM: Order of everything	83
14.9. EPAM: Bináris keresés és Buborék rendezés implementálása	84
14.10 EPAM: Saját HashMap implementáció	84
15. Helló, Stroustrup!	85
15.1. JDK osztályok	85
15.2. Másoló-mozgató szemantika	86
15.3. Hibásan implementált RSA törése	86
15.4. Változó argumentumszámú ctor	89
15.5. Összefoglaló	92
15.6. EPAM: It's gone. Or is it?	92
15.7. EPAM: Kind of equal	93
15.8. EPAM: Java GC	93

16. Helló, Gödel!	94
16.1. Gengszterek	94
16.2. C++11 Custom Allocator	94
16.3. STL map érték szerinti rendezése	95
16.4. Alternatív Tabella rendezése	96
16.5. GIMP Scheme hack	98
16.6. EPAM: Mátrix szorzás Stream API-val	98
17. Helló, hetedik hét!	101
17.1. FUTURE tevékenység editor	101
17.2. OOCWC Boost ASIO hálózatzkezelése	103
17.3. SamuCam	103
17.4. BrainB	105
17.5. OSM térképre rajzolása	106
17.6. EPAM: XML feldolgozás	106
17.7. EPAM: ASCII Art	106
18. Helló, Lauda!	108
18.1. Port scan	108
18.2. AOP	109
18.3. Android Játék	111
18.4. Junit teszt	113
18.5. EPAM: Kivételkezelés	114
19. Helló, Calvin!	116
19.1. MNIST	116
19.2. Deep MNIST	116
19.3. CIFAR-10	116
19.4. Android telefonra a TF objektum detektálója	116
19.5. SMNIST for Machines	116
19.6. Minecraft MALMO-s példa	116
20. Helló, Berners-Lee! - Olvasónapló	117
20.1. C++ vs Java összehasonlítása	117
20.2. Python - Élménybeszámoló	122

IV. Irodalomjegyzék	124
20.3. Általános	125
20.4. C	125
20.5. C++	125
20.6. Python	125
20.7. Lisp	125

Ábrák jegyzéke

2.1. A B_2 konstans közelítése	15
4.1. A double ** háromszögmátrix a memóriában	28
5.1. A Mandelbrot halmaz a komplex síkon	30
13.1. Binfá UML osztálydiagram	71
13.2. UML forrás generálás	72
13.3. Program futása utáni kimenet	74
13.4. BPMN ételrendelés folyamatábra.	75

Előszó

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

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

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

Hogyan forgasd

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

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám felhasználjuk az egyetemi programozás oktatásban is: a reguláris programozás képzésben minden hallgató otthon elvégzendő labormérési jegyzőkönyvként, vagy kollokviumi jegymegajánló dolgozatként írja meg a saját változatát belőle. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyerekeknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk?

Magam is ezeken gondolkozok. Szerintem a programozás lesz a jegyünk egy másik világba..., hogy a galaxisunk közepén lévő fekete lyuk eseményhorizontjának felületével ez milyen relációban van, ha egyáltalán, hát az homályos...

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán 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
- C kapcsán a [**KERNIGHANRITCHIE**] könyv adott részei.
- C++ kapcsán a [**BMECPP**] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a [The GNU C Reference Manual](#), mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [**BMEPY**] könyv 25-49, kb. 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket, előadásokat nézzek meg, könyveket olvassak el?

A kurzus kultúrájának élvezéséhez érdekes lehet a következő elméletek megismerése, könyvek elolvasása, filmek megnézése.

Elméletek.

- Einstein: A speciális relativitás elmélete.
- Schrödinger: Mi az élet?
- Penrose-Hameroff: Orchestrated objective reduction.
- Julian Jaynes: Breakdown of the Bicameral Mind.

Könyvek.

- Carl Sagan, Kapcsolat.
- Roger Penrose, A császár új elméje.
- Asimov: Én, a robot.
- Arthur C. Clarke: A gyermekkor vége.

Előadások.

- Mariano Sigman: Your words may predict your future mental health, <https://youtu.be/uTL9tm7S1Io>, hihetetlen, de Julian Jaynes kétkamarás tudat elméletének legjobb bizonyítéka információtechnológiai...
- Daphne Bavelier: Your brain on video games, <https://youtu.be/FktsFcoolIG8>, az esporttal kapcsolatos sztereotípiák eloszlatására („The video game players of tomorrow are older adults”: 0.40-1:20, „It is not true that Screen time make your eyesight worse”: 5:02).

Filmek.

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.
 - Rain Man, <https://www.imdb.com/title/tt0095953/>, az [SMNIST] munkát ihlette, melyeket akár az MNIST-ek helyett lehet csinálni.
 - Kódjátzsma, <https://www.imdb.com/title/tt2084970>, benne a **kódtörő feladat** élménye.
 - Interstellar, <https://www.imdb.com/title/tt0816692>.
 - Middle Men, <https://www.imdb.com/title/tt1251757/>, mitől fejlődött az internetes fizetés?
 - Pixels, <https://www.imdb.com/title/tt2120120/>, mitől fejlődött a PC?
-

- Gattaca, <https://www.imdb.com/title/tt0119177/>.
- Snowden, <https://www.imdb.com/title/tt3774114/>.
- The Social Network, <https://www.imdb.com/title/tt1285016/>.
- The Last Starfighter, <https://www.imdb.com/title/tt0087597/>.
- What the #\$*! Do We (K)now!?, <https://www.imdb.com/title/tt0399877/>.
- I, Robot, <https://www.imdb.com/title/tt0343818/>.

Sorozatok.

- Childhood's End, <https://www.imdb.com/title/tt4171822/>.
 - Westworld, <https://www.imdb.com/title/tt0475784/>, Ford az első évad 3. részében konkrétan meg is nevezi Julian Jaynes kétkamarás tudat elméletét, mint a hosztok programozásának alapját...
 - Chernobyl, <https://www.imdb.com/title/tt7366338/>.
 - Stargate Universe, <https://www.imdb.com/title/tt1286039/>, a Desteny célja a mikrohullámú háttér struktúrája mögötti rejtély feltárása...
 - The 100, <https://www.imdb.com/title/tt2661044/>.
 - Genius, <https://www.imdb.com/title/tt5673782/>.
-

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: <https://youtu.be/lvmi6tyz-nI>

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c), [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-w.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-w.c).

Számos módon hozhatunk és hozunk létre végtelen ciklusokat. Vannak esetek, amikor ez a célunk, például egy szerverfolyamat fusson folyamatosan és van amikor egy bug, mert ott lesz végtelen ciklus, ahol nem akartunk. Saját példánkban ilyen amikor a PageRank algoritmus rázza az 1 liter vizet az internetben, de az iteráció csak nem akar konvergálni...

Egy mag 100 százalékban:

```
int
main ()
{
    for (;;) ;

    return 0;
}
```

vagy az olvashatóbb, de a programozók és fordítók (szabványok) között kevésbé hordozható

```
int
#include <stdbool.h>
main ()
{
    while(true);

    return 0;
}
```

Azért érdemes a `for (; ;)` hagyományos formát használni, mert ez minden C szabvánnyal lefordul, másrészt a többi programozó azonnal látja, hogy az a végtelen ciklus szándékunk szerint végtelen és nem szoftverhiba. Mert ugye, ha a `while`-al trükközünk egy nem triviális `1` vagy `true` feltétellel, akkor ott egy másik, a forrást olvasó programozó nem látja azonnal a szándékunkat.

Egyébként a fordító a `for`-os és `while`-os ciklusból ugyanazt az assembly kódot fordítja:

```
$ gcc -S -o infty-f.S infty-f.c
$ gcc -S -o infty-w.S infty-w.c
$ diff infty-w.S infty-f.S
1c1
<  .file "infty-w.c"
---
>  .file "infty-f.c"
```

Egy mag 0 százalékban:

```
#include <unistd.h>
int
main ()
{
    for ( ; ; )
        sleep(1);

    return 0;
}
```

Minden mag 100 százalékban:

```
#include <omp.h>
int
main ()
{
#pragma omp parallel
{
    for ( ; ; );
}
    return 0;
}
```

A `gcc infty-f.c -o infty-f -fopenmp` parancssorral készítve a futtathatót, majd futtatva, közben egy másik terminálban a `top` parancsot kiadva tanulmányozzuk, mennyi CPU-t használunk:

```
top - 20:09:06 up 3:35, 1 user, load average: 5.68, 2.91, 1.38
Tasks: 329 total, 2 running, 256 sleeping, 0 stopped, 1 zombie
%Cpu0 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

```
%Cpu3 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu6 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu7 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem :16373532 total,11701240 free, 2254256 used, 2418036 buff/cache
KiB Swap:16724988 total,16724988 free, 0 used. 13751608 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5850	batfai	20	0	68360	932	836	R	798,3	0,0	8:14.23	infty-f



Werkfilm

- <https://youtu.be/lvmi6tyz-nl>

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

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

Megoldás videó:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: ugyanott.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írd egy olyan programot, ami egy labdát pattogat a karakteres konzolon! (Hogy mit értek pattogatás alatt, alább láthatod a videón.)

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

Megoldás forrása: ugyanott.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.5. Szóhossz és a Linus Torvalds féle BogomIPS

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU, <https://youtu.be/KRZlt1ZJ3qk>, .

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bogomips.c](https://bhaxor.com/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bogomips.c)

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.6. Helló, Google!

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

Megoldás forrása: a második előadás [55-63](#) fólia.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.8. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

A természetes számok építőelemei a prímszámok. Abban az értelemben, hogy minden természetes szám előállítható prímszámok szorzataként. Például $12=2*2*3$, vagy például $33=3*11$.

Prímszám az a természetes szám, amely csak önmagával és eggyel osztható. Eukleidész görög matematikus már Krisztus előtt tudta, hogy végtelen sok prímszám van, de ma sem tudja senki, hogy végtelen sok ikerprím van-e. Két prím ikerprím, ha különbségük 2.

Két egymást követő páratlan prím között a legkisebb távolság a 2, a legnagyobb távolság viszont bármilyen nagy lehet! Ez utóbbit könnyű bebizonyítani. Legyen n egy tetszőlegesen nagy szám. Akkor szorozzuk össze $n+1$ -ig a számokat, azaz számoljuk ki az $1*2*3*\dots*(n-1)*n*(n+1)$ szorzatot, aminek a neve $(n+1)$ faktoriális, jele $(n+1)!$.

Majd vizsgáljuk meg az a sorozatot:

$(n+1)!+2$, $(n+1)!+3$,... , $(n+1)!+n$, $(n+1)!+(n+1)$ ez n db egymást követő szám, ezekre (a jól ismert bizonyítás szerint) rendre igaz, hogy

- $(n+1)!+2=1*2*3*\dots*(n-1)*n*(n+1)+2$, azaz $2*$ valamennyi+2, 2 többszöröse, így ami osztható kettővel
- $(n+1)!+3=1*2*3*\dots*(n-1)*n*(n+1)+3$, azaz $3*$ valamennyi+3, ami osztható hárommal
- ...
- $(n+1)!+(n-1)=1*2*3*\dots*(n-1)*n*(n+1)+(n-1)$, azaz $(n-1)*$ valamennyi+ $(n-1)$, ami osztható $(n-1)$ -el
- $(n+1)!+n=1*2*3*\dots*(n-1)*n*(n+1)+n$, azaz $n*$ valamennyi+ n , ami osztható n -el
- $(n+1)!+(n+1)=1*2*3*\dots*(n-1)*n*(n+1)+(n+1)$, azaz $(n+1)*$ valamennyi+ $(n+1)$, ami osztható $(n+1)$ -el

tehát ebben a sorozatban egy prím nincs, akkor a $(n+1)!+2$ -nél kisebb első prím és a $(n+1)!+(n+1)$ -nél nagyobb első prím között a távolság legalább n .

Az ikerprímszám sejtés azzal foglalkozik, amikor a prímek közötti távolság 2. Azt mondja, hogy az egymástól 2 távolságra lévő prímek végtelen sokan vannak.

A Brun tétel azt mondja, hogy az ikerprímszámok reciprokaiból képzett sor összege, azaz a $(1/3+1/5)+(1/5+1/7)+(1/11+1/13)+\dots$ véges vagy végtelen sor konvergens, ami azt jelenti, hogy ezek a törtek összeadva egy határt adnak ki pontosan vagy azt át nem lépve növekednek, ami határ számot B_2 Brun konstansnak neveznek. Tehát ez nem dönti el a több ezer éve nyitott kérdést, hogy az ikerprímszámok halmaza végtelen-e? Hiszen ha véges sok van és ezek reciprokait összeadjuk, akkor ugyanúgy nem lépjük át a B_2 Brun konstans értékét, mintha végtelen sok lenne, de ezek már csak olyan csökkenő mértékben járulnának hozzá a végtelen sor összegéhez, hogy így sem lépnék át a Brun konstans értékét.

Ebben a példában egy olyan programot készítettünk, amely közelíteni próbálja a Brun konstans értékét. A repó [bhax/attention_raising/Primek_R/stp.r](https://github.com/bhax/attention_raising/Primek_R/stp.r) nevű állománya kiszámolja az ikerprímeket, összegzi a reciprokaikat és vizualizálja a kapott részeredményt.

```
# 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")
```

Soranként értelemezzük ezt a programot:

```
primes = primes(13)
```

Kiszámolja a megadott számig a prímeket.

```
> primes=primes(13)
> primes
[1] 2 3 5 7 11 13
```

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

```
> diff = primes[2:length(primes)]-primes[1:length(primes)-1]
> diff
[1] 1 2 2 4 2
```

Az egymást követő prímek különbségét képz, tehát 3-2, 5-3, 7-5, 11-7, 13-11.

```
idx = which(diff==2)
```

```
> idx = which(diff==2)
> idx
[1] 2 3 5
```

Megnézi a diff-ben, hogy melyiknél lett kettő az eredmény, mert azok az ikerprím párok, ahol ez igaz. Ez a diff-ben lévő 3-2, 5-3, 7-5, 11-7, 13-11 különbségek közül ez a 2., 3. és 5. indexűre teljesül.

```
tlprimes = primes[idx]
```

Kivette a primes-ból a párok első tagját.

```
t2primes = primes[idx]+2
```

A párok második tagját az első tagok kettő hozzáadásával képezzük.

```
rt1plust2 = 1/tlprimes+1/t2primes
```

Az $1/tlprimes$ a $tlprimes$ 3,5,11 értékéből az alábbi reciprokokat képz:

```
> 1/tlprimes
[1] 0.33333333 0.20000000 0.09090909
```

Az $1/t2primes$ a $t2primes$ 5,7,13 értékéből az alábbi reciprokokat képz:

```
> 1/t2primes
[1] 0.20000000 0.14285714 0.07692308
```

Az $1/tlprimes + 1/t2primes$ pedig ezeket a törtet rendre összeadja.

```
> 1/tlprimes+1/t2primes
[1] 0.5333333 0.3428571 0.1678322
```

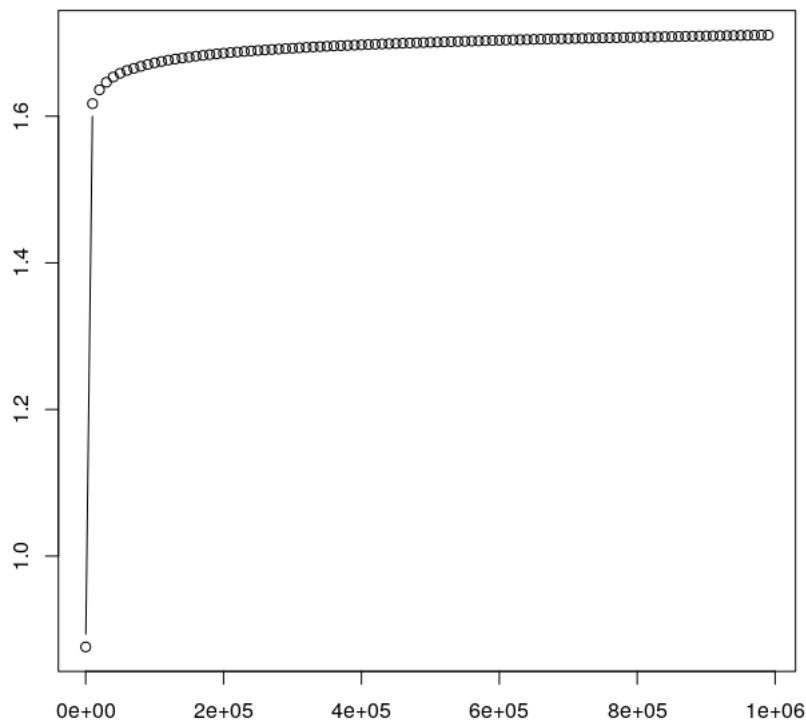
Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvénnyel.

```
sum(rt1plust2)
```

```
> sum(rt1plust2)
[1] 1.044023
```

A következő ábra azt mutatja, hogy a szumma értéke, hogyan nő, egy határértékhez tart, a B_2 Brun konstanshoz. Ezt ezzel a csipettel rajzoltuk ki, ahol először a fenti számítást 13-ig végezzük, majd 10013, majd 20013-ig, egészen 990013-ig, azaz közel 1 millióig. Vegyük észre, hogy az ábra első köre, a 13 értékhez tartozó 1.044023.

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



2.1. ábra. A B_2 konstans közelítése



Werkfilm

- <https://youtu.be/VkMFrgBhN1g>
- <https://youtu.be/aF4YK6mBwf4>

2.9. Vörös Pipacs Pokol/csiga folytonos mozgási parancsokkal

Megoldás videó: <https://youtu.be/uA6RHzXH840>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás forrása: az első előadás [27 fólia](#).

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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 forrása: az első előadás [30-32 fólia](#).

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.3. Hivatkozási nyelv

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

Megoldás forrása: az első előadás [63-65 fólia](#).

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.4. Saját lexikális elemző

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1)

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás videó: https://youtu.be/06C_PqDpD_k

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.1)

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>

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

struct cipher {
    char c;
```

```

    char *leet[4];
} l337d1c7 [] = {

    {'a', {"4", "4", "@", "/-\\"}},
    {'b', {"b", "8", "|3", "|"}},
    {'c', {"c", "(", "<", "{"}},
    {'d', {"d", "|)", "|]", "|"}},
    {'e', {"3", "3", "3", "3"}},
    {'f', {"f", "|=", "ph", "|#"}},
    {'g', {"g", "6", "[", "[+"}},
    {'h', {"h", "4", "|-|", "[-"}},
    {'i', {"1", "1", "|", "!"}},
    {'j', {"j", "7", "_|", "_/"}},
    {'k', {"k", "|<", "1<", "|{"}},
    {'l', {"l", "1", "|", "|_"}},
    {'m', {"m", "44", "(V)", "\\|"}},
    {'n', {"n", "\\|", "/\\/", "/V"}},
    {'o', {"0", "0", "()", "[]"}},
    {'p', {"p", "/o", "|D", "|o"}},
    {'q', {"q", "9", "O_", "(,)"}},
    {'r', {"r", "12", "12", "|2"}},
    {'s', {"s", "5", "$", "$"}},
    {'t', {"t", "7", "7", "'|'"}},
    {'u', {"u", "|_|", "(_)", "[_]"}},
    {'v', {"v", "\\|/", "\\|/", "\\|/"}},
    {'w', {"w", "VV", "\\|\\|/", "(/\\|)"}}},
    {'x', {"x", "%", ")(", ")("}},
    {'y', {"y", "", "", ""}},
    {'z', {"z", "2", "7_", ">_"}},

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

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

}%
%%
. {

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

```



```
{

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

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

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

        found = 1;
        break;
    }

}

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

}

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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 megváránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás videó: BHAX 357. adás.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```
- ```
int *b = &a;
```
- ```
int &r = a;
```
- ```
int c[5];
```
- ```
int (&tr)[5] = c;
```
- ```
int *d[5];
```
- ```
int *h ();
```
- ```
int *(*l) ();
```
- ```
int (*v (int c)) (int a, int b)
```

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

Megoldás videó: BHAX 357. adás.

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

```
#include <stdio.h>

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

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

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

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

    f = sum;

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

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

    g = sumormul;

    f = *g (42);

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

```
    return 0;
}

#include <stdio.h>

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

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

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

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

int
main ()
{
    F f = sum;

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

    G g = sumormul;

    f = *g (42);

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

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.9. Vörös Pipacs Pokol/csiga diszkrét mozgási parancsokkal

Megoldás videó: <https://youtu.be/Fc33ByQ6mh8>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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

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

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

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

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

    for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
```

```
        tm[i][j] = i * (i + 1) / 2 + j;

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

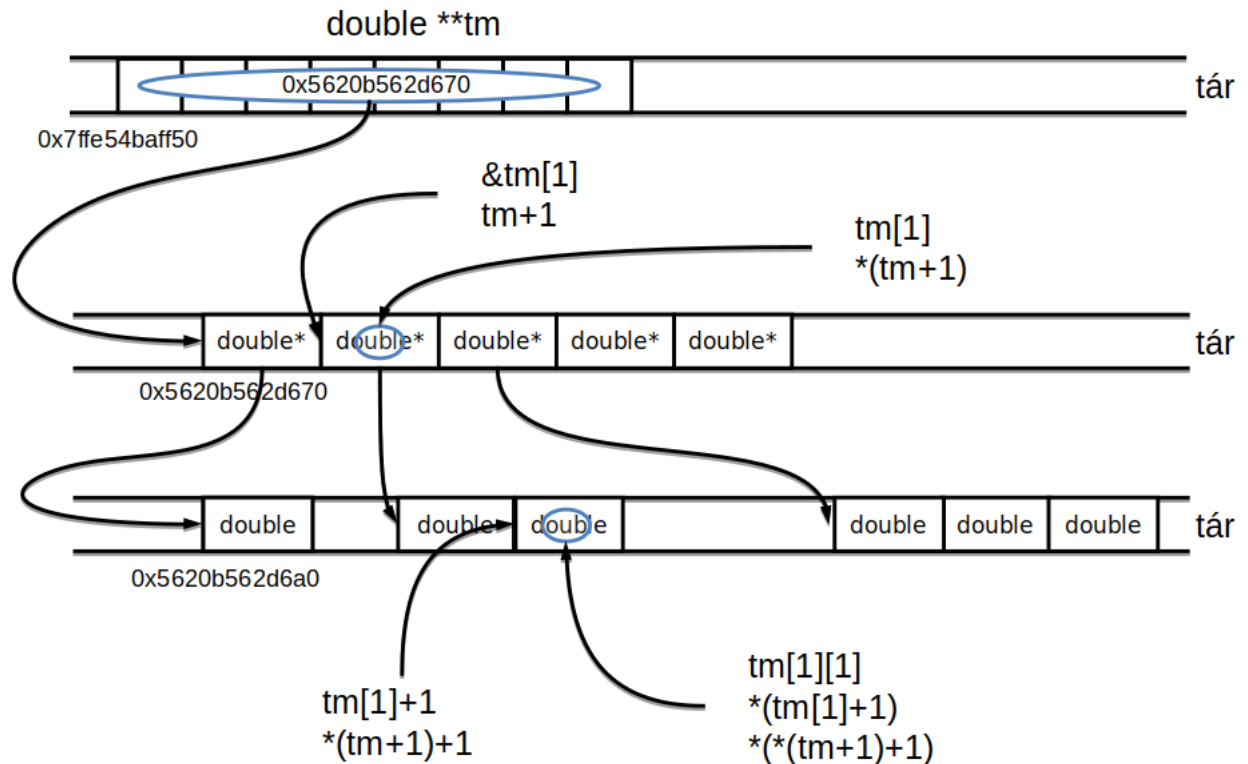
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

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

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

free (tm);

return 0;
}
```

4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: egy részletes feldolgozása az [posztban](#), lásd az e.c forrást.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.4. C EXOR törő

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

Megoldás forrása: egy részletes feldolgozása az [posztban](#), lásd az t.c forrást.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.6. Hiba-visszaterjesztéses perceptron

Fontos, hogy ebben a feladatban még nem a [neurális paradigma](#) megismerése a cél, hanem a többretegű perceptron memóriakezelése (lásd majd a változó argumentumszámú konstruktorban a double *** szerkezetet).

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve

Megoldás videó: <https://youtu.be/-GX8dzGqTdM>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

5. fejezet

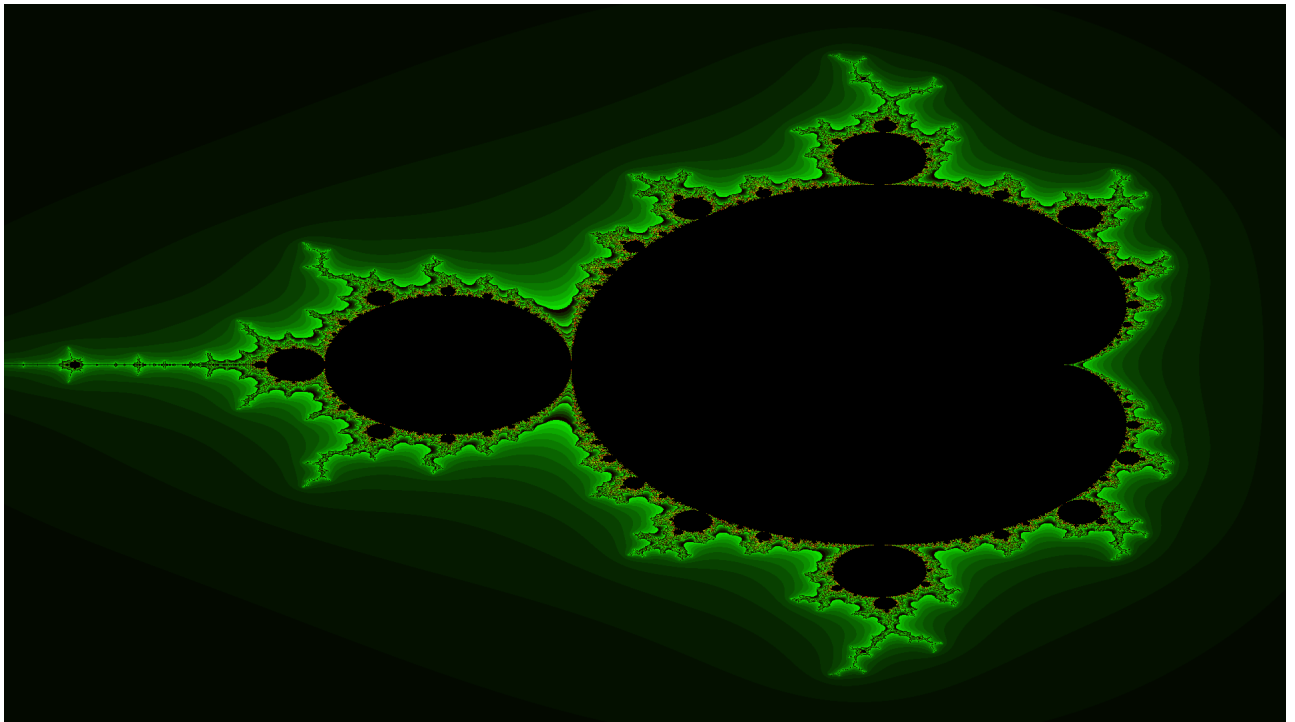
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása: [bhax/attention_raising/CUDA/mandelpngt.c++](https://github.com/bhax/attention_raising_CUDA_mandelpngt.c++) nevű állománya.



5.1. ábra. A Mandelbrot halmaz a komplex síkon

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

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1} = z_n^2 + c$, ($0 \leq n$) képlet alapján úgy, hogy a c az éppen vizsgált rácspon. A z_0 az origó. Alkalmazva a képletet a

- $z_0 = 0$
- $z_1 = 0^2 + c = c$
- $z_2 = c^2 + c$
- $z_3 = (c^2 + c)^2 + c$
- $z_4 = ((c^2 + c)^2 + c)^2 + c$
- ... s így tovább.

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

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

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

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

Megoldás forrása: BHAX repó, https://gitlab.com/nbatfai/bhax/-/blob/master/attention_raising/Mandelbrot/-3.1.2.cpp

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhax/attention_raising/Mandelbrot/3.1.2.cpp](https://gitlab.com/nbatfai/bhax/-/blob/master/attention_raising/Mandelbrot/3.1.2.cpp) nevű állománya.

```
// 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 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/>.

#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/IJMbgRzY76E>

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

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

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

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

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

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

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

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

            ++iteracio;
        }
    }
}
```

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

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

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

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

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// 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
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
```



```
//  
// Version history  
//  
// https://youtu.be/IJMbgRzY76E  
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↵  
    Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf  
//  
  
#include <iostream>  
#include "png++/png.hpp"  
#include <complex>  
  
int  
main ( int argc, char *argv[] )  
{  
  
    int szelesseg = 1920;  
    int magassag = 1080;  
    int iteraciosHatar = 255;  
    double xmin = -1.9;  
    double xmax = 0.7;  
    double ymin = -1.3;  
    double ymax = 1.3;  
    double reC = .285, imC = 0;  
    double R = 10.0;  
  
    if ( argc == 12 )  
    {  
        szelesseg = atoi ( argv[2] );  
        magassag = atoi ( argv[3] );  
        iteraciosHatar = atoi ( argv[4] );  
        xmin = atof ( argv[5] );  
        xmax = atof ( argv[6] );  
        ymin = atof ( argv[7] );  
        ymax = atof ( argv[8] );  
        reC = atof ( argv[9] );  
        imC = atof ( argv[10] );  
        R = atof ( argv[11] );  
  
    }  
    else  
    {  
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↵  
            d reC imC R" << std::endl;  
        return -1;  
    }  
  
    png::image < png::rgb_pixel > kep ( szelesseg, magassag );  
  
    double dx = ( xmax - xmin ) / szelesseg;
```

```
double dy = ( ymax - ymin ) / magassag;

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

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

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

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

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

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

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

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

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

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

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: [bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu](https://github.com/bhax/attention_raising_CUDA/blob/master/mandelpngc_60x60_100.cu) nevű állománya.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása: az ötödik előadás 26-33 fólia, illetve <https://sourceforge.net/p/udprog/code/ci/master/-tree/source/binom/Batfai-Barki/frak/>.

5.6. Mandelbrot nagyító és utazó Java nyelven

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

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518>

5.7. Vörös Pipacs Pokol/fel a láváig és vissza

Megoldás videó: <https://youtu.be/I6n8acZoyoo>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás forrása: a második előadás [17-22 fólia](#).

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

6.2. LZW

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

Megoldás forrása: https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_p

6.3. Fabejárás

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

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó: https://youtu.be/_mu54BDkqiQ

Megoldás forrása: ugyanott.

6.5. Mutató a gyökér

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

Megoldás videó: https://youtu.be/_mu54BDkqiQ

Megoldás forrása: ugyanott.

6.6. Mozgató szemantika

Írj az előző programhoz másoló/mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva, a másoló értékadás pedig a másoló konstruktorra!

Megoldás videó: <https://youtu.be/QBD3zh5OJ0Y>

Megoldás forrása: ugyanott.

6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/-/tree/master/attention_raising%2FMyrmecologist

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.2. Java életjáték

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

Megoldás forrása: <https://regi.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apb.html#conway>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: https://bhaxor.blog.hu/2018/09/09/ismerkedes_az_eletjatekkal

Megoldás videó: a hivatkozott blogba ágyazva.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.4. BrainB Benchmark

Megoldás videó: initial hack: <https://www.twitch.tv/videos/139186614>

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.5. Vörös Pipacs Pokol/19 RF

Megoldás videó: <https://youtu.be/VP0kfvRYD1Y>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python (lehet az SMNIST [SMNIST] is a példa).

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

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa
https://progater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8.2. Mély MNIST

Python (MNIST vagy SMNIST, bármelyik, amely nem a softmaxos, például lehet egy CNN-es).

Megoldás videó: https://bhaxor.blog.hu/2019/03/10/the_semantic_mnist

Megoldás forrása: SMNIST, <https://gitlab.com/nbatfai/smnist>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8.3. Minecraft-MALMÖ

Most, hogy már van némi ágensprogramozási gyakorlatod, adj egy rövid általános áttekintést a MALMÖ projektről!

Megoldás videó: initial hack: <https://youtu.be/bAPSu3Rndi8>. Red Flower Hell: <https://github.com/nbatfai/-RedFlowerHell>.

Megoldás forrása: a Red Flower Hell repójában.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8.4. Vörös Pipacs Pokol/javíts a 19 RF-en

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása: ugyanott.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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_hackelese_a_scheme_programozasi_nyelv

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-edén

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

Rövid olvasónapló a [PICI] könyvről.

10.2. C programozás bevezetés

Rövid olvasónapló a [KERNIGHANRITCHIE] könyvről.

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

10.3. C++ programozás

Rövid olvasónapló a [BMECPP] könyvről.

10.4. Python nyelvi bevezetés

Rövid olvasónapló a [BMEPY] könyvről.

III. rész

Második felvonás

**Bátf41 Haxor Stream**

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

11. fejezet

Helló, Arroway!

11.1. OO szemlélet

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

Lásd még fóliák!

Ismétlés: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf (16-22 fólia)

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

Megoldás forrása: <https://github.com/Pattesz1998/Prog2/blob/master/polargen.java>

Első Prog2-es Hello, Arroway fejezethez tartozó feladatunk: a módosított polártranszformációs normális generátor Java nyelvre történő megírása. Működése rendkívül egyszerű: legenerál két pseudorandom számot melyből az egyiket kiírjuk a standard outputra, míg a másikat a **"double tarolt"** változónkban tároljuk el.

Ezek a pseudorandom számok kísértetiesen hasonlítanak a random (véletlen) számokra, különbségük csupán abban rejlik, ezen számok a véletlen számoktól eltérően egy matematikai képlet segítségével kerülnek generálásra egy szorzat részletéből, így ez visszafejthető.

A Java polártranszformációs generátor két ilyen random számot hoz létre, melyből az egyik tárolásra kerül. A program működési elvét szemlélve, láthatjuk, hogy a **kovetkezo** függvény 10 alkalommal hívódik meg a programunk futása során. A függvény elvégzi magát a generálást, akkor ha nincsen tárolt szám, majd kiírja az eredményt. Amennyiben van korábban eltárolt számunk, abban az esetben nem történik meg a generálás, csak kiírjuk a már eltárolt számot. Ezen megoldási mód azért praktikus, mivel így ugyan a program futtatása során 10 alkalommal kerül kiíratásra szám, azonban generálás viszont mindösszesen 5 alkalommal megy végbe (ezáltal is gyorsítva a folyamatot.). Ilyenkor a processzornak nem kell végigmennie minden egyes alkalommal a generálás lépésein.

```
public class polargen {  
    boolean nincsTarolt = true;
```

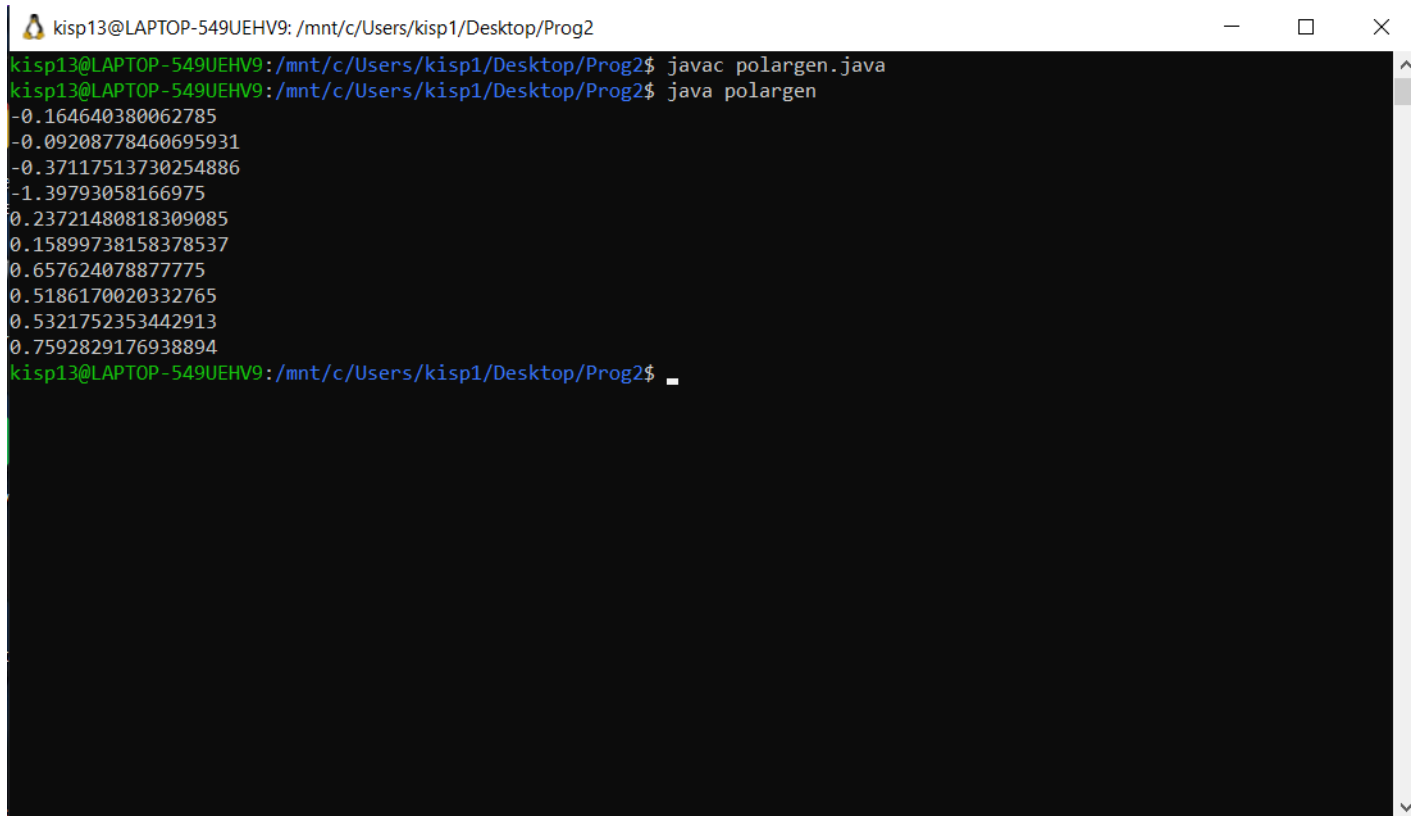
```
double tarolt;

public polargen() {
    nincsTarolt = true;
}

public double kovetkezo(){
    if(nincsTarolt){
        double u1, u2, v1, v2, w;
        do{
            u1 = Math.random();
            u2 = Math.random();
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        } while(w > 1);
        double r = Math.sqrt((-2 * Math.log(w)) / w);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
        return r * v1;
    }
    else {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
}

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

A programunk futtatásakor az alábbi kimenetet kapjuk eredményül:



```
kisp13@LAPTOP-549UEHV9: /mnt/c/Users/kisp1/Desktop/Prog2
kisp13@LAPTOP-549UEHV9: /mnt/c/Users/kisp1/Desktop/Prog2$ javac polargen.java
kisp13@LAPTOP-549UEHV9: /mnt/c/Users/kisp1/Desktop/Prog2$ java polargen
-0.164640380062785
-0.09208778460695931
-0.37117513730254886
-1.39793058166975
0.23721480818309085
0.15899738158378537
0.657624078877775
0.5186170020332765
0.5321752353442913
0.7592829176938894
kisp13@LAPTOP-549UEHV9: /mnt/c/Users/kisp1/Desktop/Prog2$
```

A következőkben meg fogjuk vizsgálni, hogy mi is a különbség a JDK -ban lévő generátor, valamint a mi generátorunk között. A JDK-s generátor javarészt úgy működik, mint a mi, módosított polártranszformációs generátorunk, viszont a különbség abban rejlik, hogy a JDK esetében gondoltak a párhuzamosítás megvalósítására is. Ezt mi sem támasztja alá jobban, mintsem az, hogy az osztályban (classban) lévő metódus létrehozásánál a **synchronized** kulcsszóval találkozhatunk. Feladata: a párhuzamos futások felügyelete, melyet úgy valósít meg, hogy amíg az egyik szálon fut maga a generálás, addig a többi szál számára nem teszi lehetővé a generáláshoz való hozzáférést.

Létezik azonban egy másik megoldás, mely szintén része a JDK -nak. Ez a megoldás a **StrictMath** osztályt használja. Erre azért van szükségünk, hogy amennyiben bármilyen környezetben van implementálva a kódunk, az eredmény mindig azonos legyen, hiszen ha csak a Math osztályt használnánk, abban az esetben az eredmény a környezettől függően változhatna.

11.2. Homokozó

Írjuk át az első védelési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden máris működik.

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

(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 Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Megoldás forrása: <https://github.com/Pattesz1998/Prog2/blob/master/binfa.java>

Maga a LZW binfa program bizonyára mindannyiunknak ismerős lehet a magasszintű programozási nyelvek 1 kurzusról, ahol megírtuk a Binfa C++ os verzióját, template osztály segítségével. A mostani feladatunk szerint az LZW binfa C++ -os forráskódját kell átírnunk Java nyelvre. Mielőtt elkezdenénk megvalósítani a Java nyelvű Binfánkat, fontos tisztázni pár dolgot: A C++ os forráskódunkból ki kell szednünk a pointereket valamint a referenciákat, mivel a Java nyelv esetében minden referenciaként van kezelve, itt nincsenek pointerek, valamint a program header részében a #include helyett az import kulcsszót kell használnunk.

Nézzük is meg magát a megvalósítást:

```
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;
class JavaBinfa
{
    public JavaBinfa ()
    {
        root = new Node('/');
        currentNode = root;
    }

    public void insert (char b)
    {
        if (b == '0')
        {
            if (currentNode.getLeftChild () == null)
            {
                Node n = new Node ('0');

                currentNode.newLeftChild (n);

                currentNode = root;
            }
            else
            {
                currentNode = currentNode.getLeftChild ();
            }
        }
        else
```

```
{
    if (currentNode.getRightChild () == null)
    {
        Node n = new Node ('1');
        currentNode.newRightChild (n);
        currentNode = root;
    }
    else
    {
        currentNode = currentNode.getRightChild ();
    }
}

}

public int getDepth ()
{
    depth = maxDepth = 0;
    recDepth (root);
    return maxDepth - 1;
}

public double getMean ()
{
    depth = sumOfMean = numberOfNodes = 0;
    recMean (root);
    mean = ((double) sumOfMean) / numberOfNodes;
    return mean;
}

public double getVariance ()
{
    mean = getMean ();
    sumOfVar = 0.0;
    depth = numberOfNodes = 0;

    recVar (root);

    if (numberOfNodes - 1 > 0)
        variance = Math.sqrt (sumOfVar / (numberOfNodes - 1));
    else
        variance = Math.sqrt (sumOfVar);

    return variance;
}

public void recDepth (Node n)
{
    if (n != null)
    {
        ++depth;
        if (depth > maxDepth)
```

```
        maxDepth = depth;
        recDepth (n.getRightChild ());

        recDepth (n.getLeftChild ());
        --depth;
    }
}

public void recMean (Node n)
{
    if (n != null)
    {
        ++depth;
        recMean (n.getRightChild ());
        recMean (n.getLeftChild ());
        --depth;
        if (n.getRightChild () == null && n.getLeftChild () == null)
        {
            ++numberOfNodes;
            sumOfMean += depth;
        }
    }
}

public void recVar (Node n)
{
    if (n != null)
    {
        ++depth;
        recVar (n.getRightChild ());
        recVar (n.getLeftChild ());
        --depth;
        if (n.getRightChild () == null && n.getLeftChild () == null)
        {
            ++numberOfNodes;
            sumOfVar += ((depth - mean) * (depth - mean));
        }
    }
}

public void printTree (PrintWriter os)
{
    depth = 0;
    printTree (root, os);
}

class Node
{
```

```
Node (char b)
{
    if(b == '0' || b == '1')
        value = b;
    else
        value = '/';
};

Node getLeftChild ()
{
    return leftChild;
}

Node getRightChild ()
{
    return rightChild;
}

void newLeftChild (Node gy)
{
    leftChild = gy;
}

void newRightChild (Node gy)
{
    rightChild = gy;
}

char getValue ()
{
    return value;
}

private char value;

private Node leftChild;
private Node rightChild;
};

Node currentNode;

private int depth, sumOfMean, numberOfNodes;
private double sumOfVar;

public void printTree (Node n, PrintWriter os)
```

```
{

    if (n != null)
    {
        ++depth;
        printTree (n.getLeftChild (), os);

        for (int i = 0; i < depth; ++i)
            os.print("---");
        os.print(n.getValue () + "(" + depth + ")\n");
        printTree (n.getRightChild (), os);
        --depth;
    }
}

protected Node root;
protected int maxDepth;
protected double mean, variance;

public static void usage ()
{
    System.out.println("Usage: lzwtree in_file -o out_file");
}

public static void main (String[] args) throws FileNotFoundException, ↵
    IOException
{

    if (args.length < 3)
    {
        usage ();
        return;
    }

    String inFile = args[0];

    if(!args[1].equals("-o"))
    {
        System.out.println(args[1]);
        usage ();
        return;
    }

    java.io.FileInputStream input = new java.io.FileInputStream(new ↵
        java.io.File(inFile));
```

```
if (input == null)
{
    System.out.println("Nem létezik");
    usage ();
    return;
}
PrintWriter output = new PrintWriter(args[2]);

byte[] b = new byte[1];
JavaBinfo binfo = new JavaBinfo();
while (input.read(b) != -1)
{
    if (b[0] == 0x0a)
    {
        break;
    }
}

boolean inComment = false;
while (input.read(b) != -1)
{
    if (b[0] == 0x3e)
    {
        inComment = true;
        continue;
    }
    if (b[0] == 0x0a)
    {
        inComment = false;
        continue;
    }
    if (inComment)
    {
        continue;
    }
    if (b[0] == 0x4e)
    {
        continue;
    }
    for (int i = 0; i < 8; ++i)
    {
        if ((b[i] & 0x80) != 0)
        {
            binfo.insert('1');
        }
        else
        {
            binfo.insert('0');
        }
    }
}
```

```

        b[0] <= 1;
    }
}

bincurrentNode.printTree(output);

output.print("depth " + bincurrentNode.getDepth () + "\n");
output.print("mean " + bincurrentNode.getMean () + "\n");
output.print("var " + bincurrentNode.getVariance () + "\n");

output.close ();
input.close ();

}

};

```

A kódunkat alaposabban megfigyelve láthatjuk a Java nyelv egyik erősségét, mivel ő automatikus szeméthyűjtőt tartalmaz (ún. Garbage Collector.) Ez arról gondoskodik, hogy a már nem használt objektum helyét felszabadítja a memóriában. Maga a Garbage Collector nem található meg a C++ nyelvben, ott a tárhely felszabaitásának céljából a destruktorokat kell használnunk.

11.3. „Gagyí”

Az ismert formális „while” tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikon meg az objektum referenciája) „mélyebb” választ, írd 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 forrása: <https://github.com/Pattesz1998/Prog2/blob/master/gagyí.java>

A következő feladatban egy roppant egyszerű példán keresztül bemutatjuk, hogy a Java nyelvben, az Integer osztályban a gyorsítótárban előre legenerált számok szerepelnek (-128 és 127 között). Ez abból is látszik, hogy mind a két Integer típusú változó értéke 6 ámba a program még sem lép be a végtelen ciklusba.

```

        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 =
sun.misc.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() {}
}
```

Ez azért van mert a ciklus feltétel vizsgálatakor, az Integer osztály esetében a memóriacím vizsgálata történik, ebből kifolyólag pedig, mivel a 6-os előre le van generálva, így a két példány ugyanarra a memóriacímre címez, így az utolsó feltétel nem teljesül, tehát nem fogunk belépni a ciklusba.

```
        public class gagy {
public static void main(String[] args) {
    Integer x = 6;
    Integer t = 6;
    System.out.println(x);
    System.out.println(t);
    while (x <= t && x >= t && t != x) ;
}
}
```

Mi történik abban az esetben, ha olyan számokat választunk amelyek már nincsenek előre legenerálva. Ha például a megadott két értéket átírjuk 600-ra, abban az esetben az összes feltétel teljesülni fog, így be fogunk lépni a végtelen ciklusba. Ez azért következik be, mivel két új példányt hoz létre a program amiknek a memóriacíme már nem egyezik meg és így az utolsó feltétel is teljesülni fog.

```
        public class gagy {
public static void main(String[] args) {
    Integer x = 600;
    Integer t = 600;
    System.out.println(x);
    System.out.println(t);
}
```

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

11.4. Yoda

Írjunk olyan Java programot, ami `java.lang.NullPointerException`-el leáll, ha nem követjük a Yoda conditions-t!
https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás forrása: <https://github.com/Pattesz1998/Prog2/blob/master/yoda.java>

A feladat lényege hogy megmutassuk mi is az a Yoda conditions. Ezt könnyű szemléltetni egy egyszerű példán keresztül. Fel kell vennünk egy null értékű változót és egy feltétel vizsgálatot. A lényeg az lesz, hogy ha egy null pointerre hívunk meg egy feltétel vizsgálatot, akkor az `java.lang.NullPointerException`-el leáll, tehát hibát kapunk eredményül. Abban az esetben viszont, ha a Yoda conditions-t követjük és felcseréljük a sorrendet akkor a programunk hiba nélkül fog lefutni. Az mindegy hogy egy konstansra vagy literálra hívjuk meg a vizsgálatot a lényeg az hogy ne null pointer legyen. Az eredmény amit vissza fog adni, az `false` lesz, viszont nem történik hiba, mivel nem a null pointerre hívtuk meg a feltétel vizsgálatot, hanem csak paraméterként adtuk át a módszernek.

```
public class Yoda  
{  
    public static void main (String[] args)  
    {  
        String myString = null;  
        if (myString.equals("proba"))  
        {  
            System.out.println("egyenlő a probával");  
        }  
        /*if ("proba".equals(myString))  
        {  
            System.out.println("egyenlő a null értékkel");  
        }*/  
    }  
}
```

Amennyiben paraméterként van átadva a módszernek, abban az esetben a vizsgálat eredménye **false** lesz.

11.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapebbpalg.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-tar>

[apbs02.html#pi_jegyei](#) (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

A BBP algoritmus arra szolgál, hogy a Pi értékét meghatározzuk, hexadecimális formában a $d+1$. helytől. Mindezt úgy tesszük, hogy nincs tudomásunk a korábbi jegyekről.

A program PiBBP nevű függvénye arra szolgál, hogy segítségével kiszámítsuk a $\{16^d \cdot \pi\} = \{4 \cdot \{16^d \cdot S_1\} - 2 \cdot \{16^d \cdot S_4\} - \{16^d \cdot S_5\} - \{16^d \cdot S_6\}\}$ képletet, ahol is a $\{ \}$ a törtrészt jelöli, továbbá ezen függvényben történik meg a hexadecimális számrendszerbe történő átváltása és az eredmény behelyezése a bufferbe is. Az átváltás és a bufferbe helyezés mind a while ciklusban történnek meg.

```
public PiBBP(int d) {
    double d16Pi = 0.0d;
    double d16S1t = d16Sj(d, 1);
    double d16S4t = d16Sj(d, 4);
    double d16S5t = d16Sj(d, 5);
    double d16S6t = d16Sj(d, 6);
    d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;
    d16Pi = d16Pi - StrictMath.floor(d16Pi);
    StringBuffer sb = new StringBuffer();
    Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};
    while(d16Pi != 0.0d) {
        int jegy = (int)StrictMath.floor(16.0d*d16Pi);
        if(jegy<10)
            sb.append(jegy);
        else
            sb.append(hexaJegyek[jegy-10]);
        d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
    }
    d16PiHexaJegyek = sb.toString();
}
```

A következő metódus a d16Sj metódus lesz. Ez egy segéd metódus, mely arra szolgál, hogy a fenti képletben lévő S1, S4, S5 és S6 értéket kiszámítsa.

```
public double d16Sj(int d, int j) {
    double d16Sj = 0.0d;
    for(int k=0; k<=d; ++k)
        d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);
    return d16Sj - StrictMath.floor(d16Sj);
}
```

Az utolsó metódus a main-en kívül amiről szót kell ejtenünk, az a n16modk metódus. Ez a metódus bináris hatványozást végez, azért hogy a S értékeket ki lehessen számítani.

```
public long n16modk(int n, int k) {
```

```
int t = 1;
while(t <= n)
t *= 2;
long r = 1;
while(true) {
if(n >= t) {
r = (16*r) % k;
n = n - t;
}
t = t/2;
if(t < 1)
break;
r = (r*r) % k;
}
return r;
}
```

Végül a main függvényben egy PiBBP objektum létrehozása és a d értékének meghatározása történik, ami jelen helyzetben 1000000. Így tehát a Pi értékei az 1000001. helytől kezdődően lesznek kiszámítva.

```
public static void main(String args[]) {
System.out.println(new PiBBP(1000000));
}
```

11.6. EPAM: Java Object metódusok

Mutasd be a Java Object metódusait és mutass rá mely metódusokat érdemes egy saját osztályunkban felüldefiniálni és miért. (Lásd még Object class forráskódja)

Az Object osztály esetében érdemes megemlítenünk, hogy minden osztály, ebből az osztályból származik, ez áll a képzeletbeli piramisunk tetején. Az object osztály úgynevezett ősosztály, mely minden Java osztály őseinek metódusa. Ez lényegében azt jelenti, hogy a metódussal minden Java osztály rendelkezik.

Az Object osztály definiálja az alapvető működést, mely minden objektumnál rendelkezésre áll. Legfontosabb metódusai a következők: clone, equals, hashCode, finalize, toString, getClass metódusok.

A **clone** egy metódus a Java programozási nyelvben, melyet az objektumok másolására használunk. A Java-ban az objektumokat referenciaváltozókon keresztül manipulálják, és nincs operátor az objektum lemásolásához - a hozzárendelési operátor másolja a referenciát, nem az objektumot. A klón () módszer biztosítja ezt a hiányzó funkciót. Azoknak az osztályoknak, amelyek másolási funkciót akarnak, végre kell hajtaniuk valamilyen módszert. Bizonyos mértékben ezt a funkciót a(z) **Object.clone()** biztosítja. A **equals** metódus feladata, hogy összehasonlítsa két megadott karakterláncot a karakterláncok tartalma alapján. Amennyiben bármely két karakter nem egyezik meg, akkor "hamis" értéket fogunk eredményül kapni. Amennyiben az összes karakter megegyezik, abban az esetben "igaz" értéket kapunk vissza. A Stringequals() metódus felülírja az Objektum osztály equals metódusát. A **hashCode** metódus feladata

11.7. EPAM: Eljárásorientál vs Objektumorientált

Fealadatléírás

11.8. EPAM: Objektum példányosítás programozási mintákkal

Fealadatléírás

12. fejezet

Helló, Liskov!

12.1. Liskov helyettesítés sértése

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

A Liskov elv lényege az objektumorientált programok helyes tervezése. A könnyű érthetőség kedvéért egy példán keresztül nézzük meg ezt. Vegyük például azt hogy van egy madár osztályunk. Ebben létrehozunk egy tulajdonságot ami kimondja hogy a madarak tudnak repülni. Ez után ha származtatunk két osztályt a madár osztályból, legyen egy sas és egy pingvin osztály, akkor az megsérti a liskov elvet. A elv szerint ugyanis rossz a tervezés. Ez abból derül ki hogy mivel származtatott osztályról van szó a pingvin osztály megkapja a reülés tulajdonságot. Ez viszont hiba, mivel a pingvin nem tud repülni. Ha nem akarjuk megsérteni az elvet akkor a megoldás az hogy újra kell tervezni a programot. Ezt például úgy lehet megtenni ha a repülés tulajdonságát elkülönítjük egy repülő madarak osztályba és ez után a sas a repülő madarak osztályába fog tartozni a pingvin pedig csak a madarak osztályába. Így a tervezés már jó lesz és nem lesz megsérve a Liskov elv.

```
class Madar {
public void repul() {}
};
class Program {
public void fgv ( Madar madar ) {
    madar.repul();
}
};
class Sas extends Madar
{};
class Pingvin extends Madar
{};
class Liskov{
public static void main(String[] args)
{
```

```
Program kod = new Program();
Madar mad = new Madar();
kod.fgv ( mad );
Sas sass = new Sas();
kod.fgv ( sass );
Pingvin pingvin = new Pingvin();
kod.fgv ( pingvin );
}
```

```
class Madar {
public:
virtual void repul() {};
};
class Program {
public:
void fgv ( Madar &madar ) {
madar.repul();
}
};
class Sas : public Madar
{};
class Pingvin : public Madar
{};
int main ( int argc, char **argv )
{
Program program;
Madar madar;
program.fgv ( madar );
Sas sas;
program.fgv ( sas );
Pingvin pingvin;
program.fgv ( pingvin );
}
```

12.2. Szülő-gyerek

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

A feladat megoldása során szükség van két osztályra a kódban ahhoz hogy bemutathassuk a feladatot. Az egyik osztály az ős osztály lesz, ennek a neve **Szülő**. A másik osztály egy származtatott osztály amely a **Gyerek** nevet kapta.

Az ős osztály lényege hogy egy általánosabb egységet hozzunk létre, amely egy nagyobb egységbe zárja a kódban létrehozott példányokat. Így a benne szereplő tulajdonságok egy tágabb tartományt hoznak létre. Ilyen például az, ha egy programban le akarjuk modellezni az emlősöket. Az emlős lesz az ős osztály, amely egy tágabb tartomány, és minden benne szereplő tulajdonság igaz lesz az emlősökre.

A származtatott osztály egy szűkebb tartomány, amely már több megkötést tartalmaz, viszont rendelkezik az ős osztály tulajdonságaival is. Erre példa ha mondjuk származtatott osztályként létrehozunk egy ember vagy delfin osztályt. Mindkettő több megkötést tartalmaz mint az emlős, de akár az embert, akár a delfint nézzük mind a kettő rendelkezik az emlős osztály tulajdonságaival is.

A feladat megoldása során egy egyszerűbb példa kerül bemutatásra. Egyszer lesz nekünk, a korábban már említett Szülő és Gyerek osztály. Az ős osztály a szülő lesz, a származtatott osztály pedig a gyerek. A gyerek osztály tudja használni a szülő osztály metódusát is, viszont a szülő nem tudja a gyereket.

Most pedig nézzük meg a kódokat. Először is a Java nyelvű megvalósítást:

```
class Szulo
{
public void szulo_vagyok()
{
System.out.println("Én vagyok a szülő.");
}
};
class Gyerek extends Szulo
{
public void gyerek_vagyok()
{
System.out.println("Én vagyok a gyerek.");
}
public static void main(String[] args)
{
Szulo apa = new Szulo();
Gyerek joska = new Gyerek();
apa.szulo_vagyok();
joska.gyerek_vagyok();
joska.szulo_vagyok();
apa.gyerek_vagyok(); //ez így nem működik mivel az ős ←←
osztály nem fér hozzá a származtatott osztályhoz ezáltal ←←
a benne lévő metódusokhoz sem
}
};
```

Létrehozásra kerül a két osztály, majd a mainben mindegyik osztályból egy-egy példány. Láthatjuk hogy a kódban jelölve van a származtatás. A Gyerek osztály létrehozásánál ott van az **extends** kulcsszó. Ez arra szolgál hogy mögötte feltüntessünk egy másik osztályt, amely által az meg lesz jelölve ősnek. Mivel van

ős osztálya ő egy származtatottá válik. Végül személtetés céljából a példányokra meghívásra kerül, a szülő osztály, valamint a gyerek osztály metódusa is. A gyerek osztály tudja használni a szülő osztály függvényét, míg a szülő nem tudja használni a gyerek osztályét.

Azonban, ha mégis megpróbáljuk az ős osztálynak a példányára meghívni a származtatott osztály függvényét akkor az hibát fog eredményezni.

Most pedig következzen a C++-os megoldás. Látható hogy lényeges különbség nincs a Javához képest. Ami változik az mindösszesen annyi, hogy másként van megjelölve a származtatás. A futás során is ugyanarra az eredményre jutunk.

```
#include <iostream>
class Szulo
{
public:
void szulo_vagyok()
{
std::cout << "Én vagyok a szülő." << std::endl;
}
};
class Gyerek : public Szulo
{
public:
void gyerek_vagyok()
{
std::cout << "Én vagyok a gyerek." << std::endl;
}
};
int main ( int argc, char **argv )
{
Szulo szulo;
Gyerek gyerek;
szulo.szulo_vagyok();
gyerek.gyerek_vagyok();
gyerek.szulo_vagyok();
//szulo.gyerek_vagyok(); ez így nem működik mivel az "os ←-
osztály nem fér hozzá a származtatott osztályhoz ezáltal a ←-
benne lévő metódusokhoz sem
}
```

12.3. Anti OO

A BBP algoritmussal a Pi hexadecimális kifejtésének a 0. pozíciótól számított 106, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási idöket! <https://www.tankonyvtar.hu/-hu/tartalom/tkt/javat-tanito-k-javat/apas03.html#id561066>


```
Iteracio...  
norma = 0.05918759643574294  
osszeg = 1.0  
Iteracio...  
norma = 0.014871507967965858  
osszeg = 0.9999999999999999  
Iteracio...  
norma = 0.006686856768932613  
osszeg = 1.0  
Iteracio...  
norma = 0.007776749198562133  
osszeg = 1.0  
Iteracio...  
norma = 0.007661483555477314  
osszeg = 0.9999999999999999  
Iteracio...  
norma = 0.007581657116770109  
osszeg = 0.9999999999999998  
Iteracio...  
norma = 0.007532798726590474  
osszeg = 0.9999999999999998  
Iteracio...  
norma = 0.007538144256251714  
osszeg = 0.9999999999999998  
Iteracio...  
norma = 0.007535825315190922  
osszeg = 1.0  
Iteracio...  
norma = 0.00753510193655861  
osszeg = 0.9999999999999997  
Iteracio...  
norma = 0.0075353297234758716  
osszeg = 0.9999999999999998  
Iteracio...  
norma = 0.007535284725802345  
osszeg = 0.9999999999999999  
Iteracio...  
norma = 0.007535275242694286  
osszeg = 0.9999999999999996  
Iteracio...  
norma = 0.0075352801728795745  
osszeg = 0.9999999999999999  
Iteracio...  
norma = 0.007535280078749908  
osszeg = 0.9999999999999998  
Iteracio...  
norma = 0.007535279718816022  
osszeg = 0.9999999999999998  
Iteracio...  
norma = 0.007535279786665789  
osszeg = 0.9999999999999998  
Iteracio...  
norma = 0.007535279802432719
```

12.7. EPAM: Interfész evolúció Java-ban

Fealadatleírás

12.8. EPAM: Liskov féle helyettesíthetőség elve, öröklődés

Fealadatleírás

12.9. EPAM: Interfész, Osztály, Absztrak Osztály

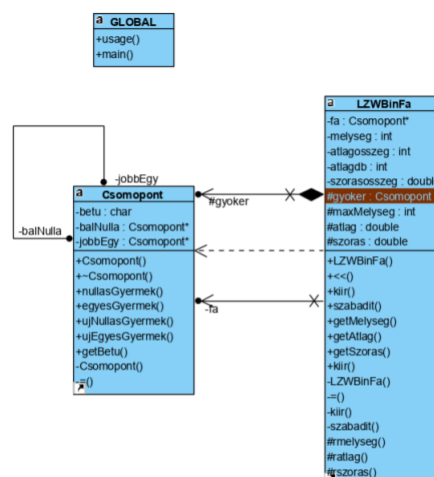
Fealadatleírás

13. fejezet

Helló, Mandelbrot!

13.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nIERIEOs.



13.1. ábra. Binfu UML osztálydiagram

A feladat megoldása során a Visual Paradigm - ezt a programot ajánlom mindenkinek a feladat megoldásához - programot használtam a kódból való **UML** diagram elkészítésére. Az UML diagrammok lényege az, hogy egy ábrában szemléltesse egy adott program tervezetét. A Visual Paradigm kiválóan alkalmas arra, hogy segítségével megtervezzünk egy komplett programot, majd a diagramm alapján elkészítsük annak kódbéli verzióját. Létrehozás esetén kétféle verzió létezik: Az egyik az, hogy a terv alapján kézzel megírjuk a kódot, a másik esetben pedig az ábra alapján generáljuk a kódot. Ennyire előre azért ne fussunk, hiszen erről az ábra alapján történő kódgenerálásról a következő feladat során ejtek bővebben szót.

Jelen feladatban, egy már kész kódból generáltuk le magát a diagrammot. Ez hasznos lehet abban az esetben, ha például egy bemutató alakalmával el kell magyarázni egy adott program működését. A kód alapján generálni kell egy diagrammot, majd az így legenerált, könnyen átlátható ábrán keresztül könnyedén prezentálhatjuk a programunk felépítését.

Itt a védési programból (LZWBinfa kódból) készítettük el a diagrammot. A diagrammon láthatjuk, hogy milyen osztályokból épül fel maga a program és azok között milyen kapcsolat van, továbbá látható az is, hogy az osztályok milyen változókkal és metódusokkal rendelkeznek. Egy osztály két részre van osztva a diagrammon. A felső részben található a változók listája, az alsó részben pedig a metódusok listája. Minden változó és metódus előtt látható egy +, - vagy # jel. Ez jelöli azt, hogy milyen a hozzáférhetősége. A + jelenti a public-ot, a - a private-ot, a # pedig a protected jelzőt jelenti.

A kapcsoltokat, nyilak segítségével jelöljük az osztályok között. Elsőnek vegyük a sima nyilat. Ez az asszociációt jelöli a két osztály között. Látható hogy a nyílra rá van írva a -fa felirat. Ez azt jelenti, hogy azon keresztül áll fent a kapcsolat. A következő nyíl a szaggatott. Ez a dependency, vagyis függőség nyíl. Ez azt jelenti, hogy az LZWBinfa osztály függ a csomópont osztálytól, vagyis ha a Csomópont osztály megváltozik akkor vele együtt változik az LZWBinfa osztály is. Végezetül pedig lássuk az utolsó nyilat. Ez a kompozíciót jelenti. Lényegében két osztály kompozícióban van, ha a Csomópont osztály csak az LZWBinfa osztállyal együtt tud létezni, tehát, ha az LZWBinfa osztály törlésre kerül akkor a Csomópont osztály is elveszíti a létjogosultságát, mivel a csomópont nem létezhet a binfa nélkül.

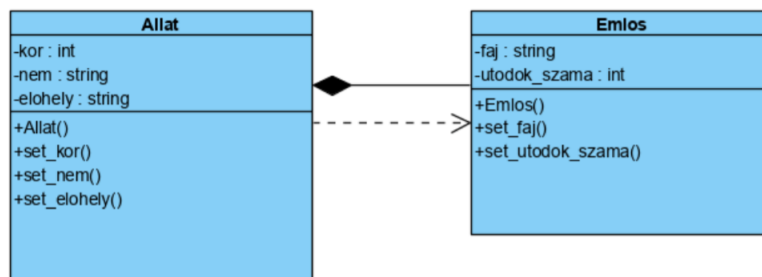
Létezik azonban egy másik kapcsolat fajta is, ami a jelen diagrammunkon nem szerepel, ez pedig az aggregáció. Az aggregáció nagyon hasonlít a kompozícióra, különbség csupán annyi, hogy ott a részosztálynak akkor is megvan a létjogosultsága ha az osztály eltűnik.

13.2. Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generáljunk belőle forrást!

Ebben a feladatban egy UML diagrammból kell kódot készíteni. (ez a feladat pontosan az előző ellentétje.) Az UML diagrammot a Visual Paradigm programmal készítjük el majd ezt követően kódot generálunk belőle. Első lépésként, meg kell adnunk az osztályokat, majd belehelyezni a kívánt változókat és metódusokat. Miután ez megvan, be kell jelölnünk a közöttük lévő kapcsolatokat, majd elvégezzük magát a kód generálást.

A kapott eredmény:



13.2. ábra. UML forrás generálás

13.3. Egy esettan

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

Megoldás forrása: <https://github.com/Pattesz1998/Prog2/tree/master/Esettan>

A könyv esettana egy számítógépes bolt nyilvántartó rendszeréhez készít egy programot, melyben eltárolásra kerülnek a boltban megtalálható termékek a neve, típusa, beszerzési ára, beszerzési ideje, és számított értéként megadja az eladási árát valamint az eszköz korát. A könyv részletesen tárgyalja a program tervezésének a menetét, hogy miként kell felépíteni a programot objektumorientált formában és milyen részeket tartalmazzon maga a programkódunk. A tervezetet UML diagram formájában bemutatja számunkra, majd lépésről lépésre kitárgyalja a kód felépítését.

A tervezet alapján létre kell hoznunk egy ős osztály, amely az általános tulajdonságokat tárolja minden - a boltban fellelhető - termékről. Mivel a származtatott osztályok módosításokat eszközölnek, így virtuális metódusok használata szükséges számunkra. Tökéletes példa egy ilyen megvalósításra, ha mondjuk az eszköz típusát tároljuk el. Ezt az alábbi metódus segítségével tudjuk megtenni: `virtual std::string GetType() const = 0;` Mivel a változók `protected`ként vannak létrehozva így a hozzáférésükhöz szükségünk van metódusok meglétére is.

A származtatott osztályok arra szolgálnak, hogy egy új eszköz esetén kezelni tudják annak értékeit. Mivel az adatok olvasása, illetve írása adatfolyamon keresztül zajlik, így miután eldöntésre kerül hogy milyen típusú eszközről van szó, a vezérlés átkerül a származtatott osztályokba és azok elvégzik a beolvasást.

A könyvben részletesen bemutatásra kerül továbbá az is, hogy az adatfolyamban miként vannak tárolva az értékek, azonban most nem erre térnek ki. Ami jelen esetben számunkra érdekesebb, az az, hogy lehetőségünk van arra, hogy összetettebb termékeket tároljunk el. Ez abban az esetben igaz, amennyiben például valaki egy számítógép konfiguráció vásárlásán gondolkodik. Ebben az esetben az alkatrészek külön-külön el vannak tárolva, ám jelezzük az adatfolyamban, hogy ezek egy konfiguráció részét képezik.

Mivel, már megvan hogy a termékeket hogyan kell beolvasnunk, illetve kezelnünk, következő lépésként azt kell megvizsgálnunk, hogy hogyan is tudjuk kezelni magát a nyilvántartást. Erre is ad példát a könyvünk. A **ProductInventory** lehetővé teszi hogy olvassuk a nyilvántartást, szerkeszthessük (írjunk) bele, kiírjuk azt egy kimenetre és új elemet hozzunk benne létre.

Végül pedig ahhoz, hogy a feladat feltételeinek eleget tegyen a programunk, néhány módosítást kell eszközölnünk a kódban a beolvasás terén. Ahhoz, hogy megfelelő legyen a feltételeknek és az objektumorientált elveknek is néhány módosításra van szükségünk. Ilyen például az hogy switch-case szerkezet ne szerepeljen az ős osztályban.

A fejezet utolsó pontjában pedig arról van szó, hogy miként lehet elegánsabbá tenni a kódunkat. A könyvben erre olyan megoldást találhatunk, hogy a programunk, a dátumok kezelését egy külön osztályra bízza.

A program futtatása után az alábbi konzol kimenetet kapjuk:

```
kisp13@LAPTOP-549UEHV9: /mnt/c/Users/kisp1/Desktop/Prog2/Esettan$ g++ compositeproduct.h compositeproduct.cpp computerconfiguration.h computerproductfactory.h computerproductfactory.cpp display.h disk.h harddisk.cpp product.h product.cpp productfactory.h productfactory.cpp productinventory.h productinventory.cpp productinventorytest.cpp -o proba
kisp13@LAPTOP-549UEHV9: /mnt/c/Users/kisp1/Desktop/Prog2/Esettan$ ./proba
Test1: create inventory and printing it to the screen.
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20201015, Age: 0, Current price: 30000, InchWidth: 13, InchHeight: 12
1.: Type: HardDisk, Name: WD, Initial price: 25000, Date of acquisition: 20201015, Age: 0, Current price: 25000, SpeedRPM: 7500
Press any key to continue...a

Test2: loading inventory from a file (computerproducts.txt), printing it, and then writing it to a file (computerproducts_out.txt).
End of reading product items.The content of the file is:
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20011001, Age: 6953, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 5128, Current price: 28000, InchWidth: 10, InchHeight: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age: 5128, Current price: 70000
Items:
0.: Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20011001, Age: 6953, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: HardDisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 5128, Current price: 28000, SpeedRPM: 7000
3.: Type: HardDisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20050228, Age: 5707, Current price: 20000, SpeedRPM: 7000

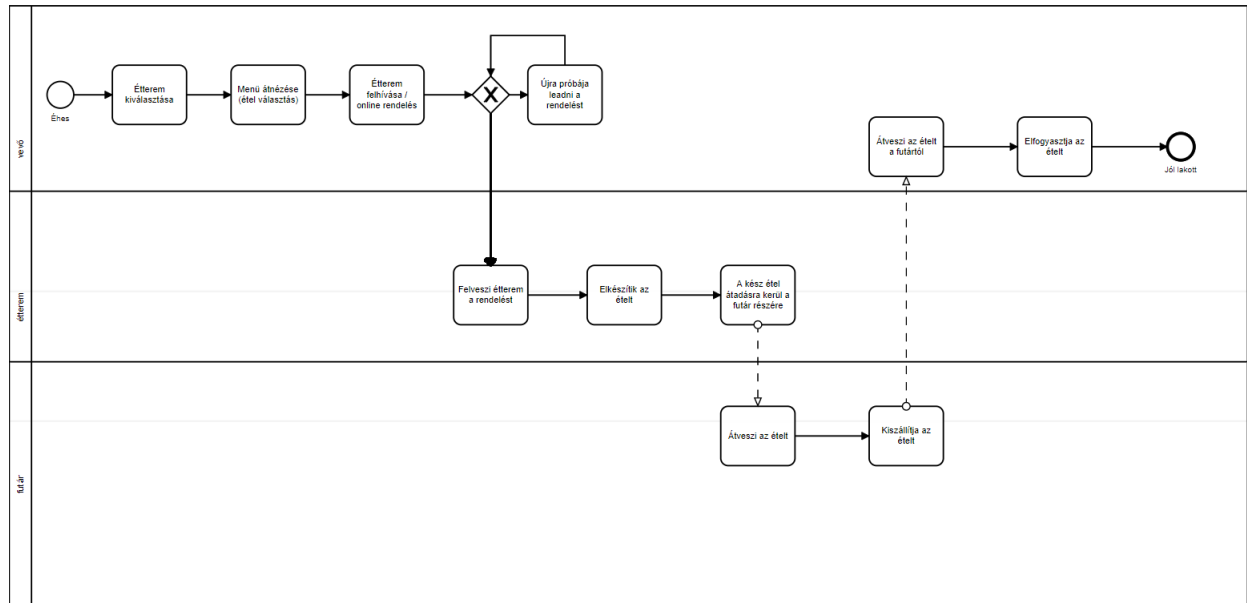
The content of the inventory has been written to computerproducts_out.txt
Done.kisp13@LAPTOP-549UEHV9:/mnt/c/Users/kisp1/Desktop/Prog2/Esettan$
```

13.3. ábra. Program futása utáni kimenet

13.4. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben!

https://arato.inf.unideb.hu/batfai.norbert/PROG2/Prog2_7.pdf (34-47 fólia)



13.4. ábra. BPMN ételrendelés folyamatára.

A BPMN - teljes nevén Business Process Model and Notation - egy modellezési forma, amely kiválóan alkalmas arra hogy egy folyamat lépéseit szemléltessünk benne (folyamatábra, pseudocode segítségével). Az ábrák segítségével be tudjuk mutatni azt, hogy egy adott folyamat során milyen lépési lehetőségek vannak és ezek hová is vezethetnek. Kiválóan alkalmas folyamatok megtervezésére és arra hogy a folyamatok minden irányú kimenetelét szemléltethessük.

Jelen ábrán egy étteremből való ételrendelés folyamata látható. A kiinduló állapotot az első kör jelöli amikor is, a vásárló éhes. Ábránkon a rendelési folyamat lépéseit a téglalapok, míg a folyamat irányát a nyilak jelölik. A rombusz az elágazásokat jelenti, ahonnan a kimeneteltől függően halad tovább a folyamat. A folyamatábra végén található kör a végső állapotot jelenti, ahol a folyamatunk véget ér.

13.5. TeX UML

Fejlesztési leírás

Ebben a feladatban az OOCWC projektről kellett UML diagrammot készíteni, azonban jelen esetben nem grafikusán, hanem kódolás segítségével. A végeredmény ugyanúgy egy diagramm lesz, valamint a jelölés rendszere is ugyanaz lesz, mint az első két feladat esetében. A megvalósítása viszont nem a Visual Paradigm programmal történt, hanem kódok segítségével. Az osztályok generálását a következőképpen tehetjük meg:

```
Class.A("MyShmClient")
("# nr_graph: NodeRefGraph*", "# m_teamname: string", "- nr2v: <--
map<unsigned_object_id_type, NRGVertex>")
("+MyShmClient()",
"+~MyShmClient()",
"+start()",
"+start10()",
```



```

"+num_vertices()",
"+print_edges()",
"+print_vertices()",
"+bgl_graph()",
"+hasDijkstraPath()",
"+hasBellmanFordPath()",
"-foo()",
"-init()",
"-gangsters()",
"-initcops()",
"-pos()",
"-car()",
"-route()");

```

Ezen osztály esetében látható, hogy a változók és a metódusok miként lettek belehelyezve az osztályba. Az első zárójelben az osztály nevét definiáltuk, a másodikban a változók nevét és típusát, valamint a hozzáférhetőségüket adtuk meg, a harmadik zárójel esetében pedig, a metódusok lettek létrehozva. A diagramm minden osztályát így módon hoztuk létre.

Léteznek továbbá olyan osztályok is, melyek át lettek alakítva, azért, hogy más szerepet tölthessenek be. Ilyen például: a SmartCar a sampleclient package-ben. Látható hogy ott szerepel egy **Struct** kulcsszó a SmartCar név fölött, ez arra szolgál, hogy megjelenítsük azt, hogy ez egy struktúra nem pedig egy osztály. Megvalósításhoz az alábbi kódot használjuk: `classStereotypes.B("Struct");`

Az osztályok elhelyezkedését a diagrammon belül is kóddal kell megadni, erre több fajta megoldás is létezik. Az egyik megoldási mód az-az, hogy megadjuk az osztályok egymáshoz való viszonyát relatívan. Ezt így módon tehetjük meg: $J.w = H.e + (100, 0)$;. A másik megoldás az talán egyszerűbb a fent említett megoldási módnál. Ebben az esetben, azt kell megadni, hogy egymáshoz képest milyen irányba helyezkedjenek el és mennyi legyen a térköz közöttük.

```

topToBottom.midx(50)(L, K, M);
leftToRight.midy(50)(O, M);

```

A program végén pedig, még be kell rajzolnunk közéjük a köztük fennálló kapcsolatokat. Ezt a **link** utasítással tehetjük meg. Az első zárójelben adhatjuk meg a kapcsolat típusát: `link(nest)(B.n -- A.s);`.

A diagramm kirajzolása után az alábbi eredményt kapjuk: <https://github.com/Pattesz1998/Prog2/blob/master/robocar-1.pdf>

13.6. EPAM: Neptun tantárgyfelvétel modellezése UML-ben

Modellezd le a Neptun rendszer tárgyfelvételéhez szükséges objektumokat UML diagramm segítségével.

13.7. EPAM: Neptun tantárgyfelvétel UML diagram implementálás

Fealadatleírás

13.8. EPAM: OO modellezés

Fealadatléírás

14. fejezet

Helló, Chomsky!

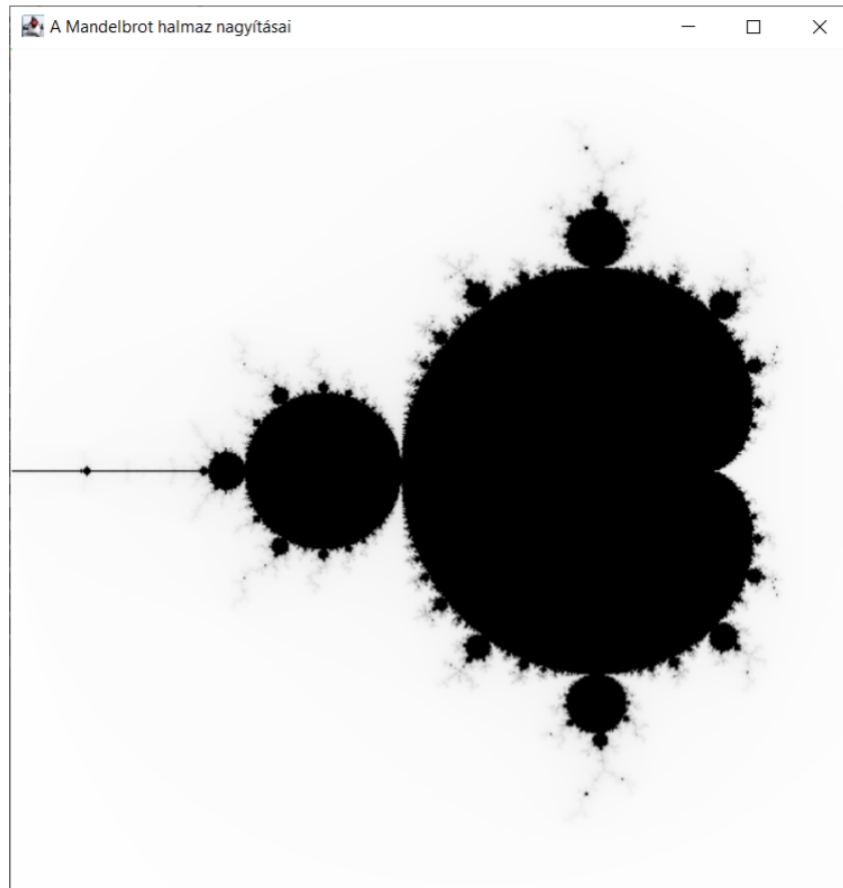
14.1. Encoding

Fordítsuk le és futtassuk a Javat tanítók 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>

A feladat során a MandelbrotHalmazNagyító.java fájlt kellett lefordítanunk úgy, hogy az ékezetes karakterekkel együtt is le tudjon fordulni a programunk. Ezt úgy tudjuk megtenni, ha a program fordítása során megadunk egy encoding kapcsolót, a parancssori paraméterek közzé: **javac -encoding "ISO-8859-1" MandelbrotIterációk.java MandelbrotHalmazNagyító.java.**

Az encoding kulcsszó segítségével adhatjuk meg a karakterkódolást, mely segítségével értelmezni szeretnénk a kódot. Jelen helyzetben ez a karakterkészlet az ISO-8859-1 lesz. Ez egy szabvány, amely tartalmazza az ékezetes karaktereket, így már a megfelelő módon fog működni a programunk.

Programunk futása során kapott kimenet:



14.2. OOCWC lexer

14.3. l33tConverter

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet> (Ha ezt első részben nem tetted meg, akkor írasd ki és magyarázd meg a használt struktúrákban memórafoglalását!)

Megoldás forrása: <https://github.com/Pattesz1998/Prog2/blob/master/L33tConverter.java>

A feladatunk megoldása során létrehoztunk egy java kódot, amely megvalósítja magát a helyettesítést. A program első részében a `toLeetCode` metódus található. Ez a metódus felel a helyettesítésért a megadott karakterláncra. Először a `pattern`-ben megadjuk, hogy milyen karaktereknek létezik megfelelője, amin a helyettesítést végre tudjuk majd hajtani, majd a `map`-ban eltároljuk magát a helyettesítéshez használt táblázatot. Ezután egy `for` ciklus segítségével végig megyünk a `buffer`-en és beolvassuk a karakterláncot. A `key` változókban tároljuk el az éppen vizsgált karaktert nagybetűs (uppercase) formában, majd ezt követően megvizsgáljuk a `key`-ben tárolt karaktert.

Amennyiben a `key`-ben tárolt karakter szerepel a táblázatban, abban az esetben kicseréljük a megfelelőjére, ha viszont nem szerepel a táblázatban akkor a `key`-ben lévő karaktert kapjuk vissza eredményül. A metódus végén a `result`-ban eltárolt eredményt kapjuk vissza.

A main függvényben példányosítjuk az objektumot, majd beolvassuk a karakterláncot a bufferből, végül erre a láncra meghívjuk a helyettesítő metódust, és kiíratjuk az eredményt. Mivel ez a folyamat egy ciklusba van ágyazva, így többször is lehetőségünk van elvégezni a cserét.

14.4. Full screen

A feladatunk az, hogy készítsünk egy Java Full screen programot.

Screen.java

```
import java.awt.*;
import javax.swing.JFrame;
public class Screen {
    private GraphicsDevice vc;
    public Screen() {
        GraphicsEnvironment env = GraphicsEnvironment. <--
        getLocalGraphicsEnvironment();
        vc = env.getDefaultScreenDevice();
    }
    public void setFullScreen(DisplayMode dm, JFrame window) {
        window.setUndecorated(true);
        window.setResizable(true);
        vc.setFullScreenWindow(window);
        if(dm != null && vc.isDisplayChangeSupported()) {
            try {
                vc.setDisplayMode(dm);
            } catch (Exception ex) {}
        }
    }
    public Window getFullScreenWindow() {
        return vc.getFullScreenWindow();
    }
    public void restoreScreen() {
        Window w = vc.getFullScreenWindow();
        if(w != null) {
            w.dispose();
        }
        vc.setFullScreenWindow(null);
    }
}
```

bucky.java

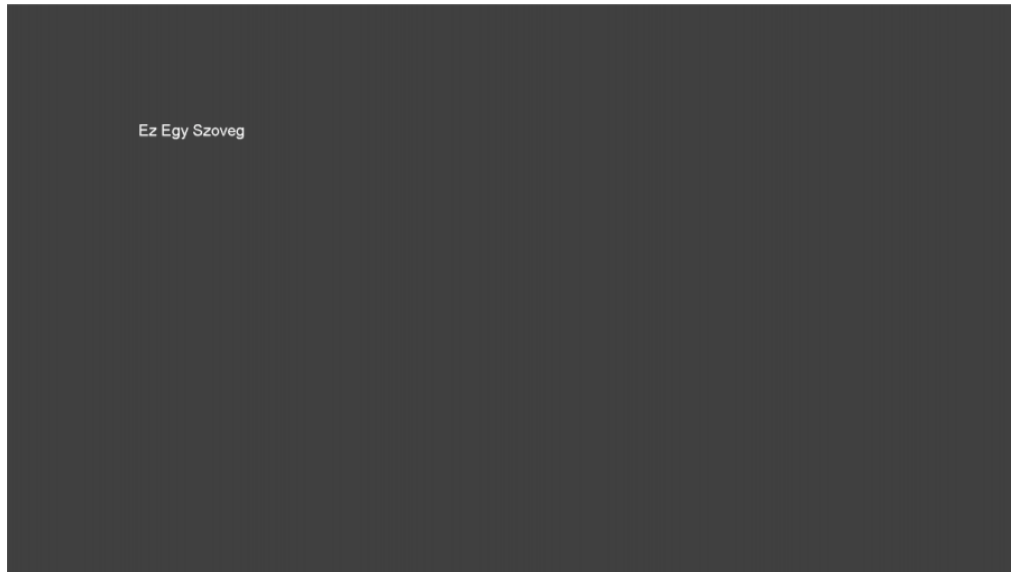
```
import java.awt.*;
import javax.swing.JFrame;
public class bucky extends JFrame {
    public static void main(String[] args) {
```

```
DisplayMode dm = new DisplayMode(800,600,16, ←-
DisplayMode.REFRESH_RATE_UNKNOWN);
bucky b = new bucky();
b.run(dm);
}
public void run(DisplayMode dm) {
getContentPane().setBackground(Color.DARK_GRAY);
setForeground(Color.WHITE);
setFont(new Font("Arial", Font.PLAIN, 24));
Screen s = new Screen();
try {
s.setFullScreen(dm, this);
try {
Thread.sleep(5000);
}catch(Exception ex) {}
}finally {
s.restoreScreen();
}
}
public void paint(Graphics g) {
super.paint(g);
g.drawString("Ez Egy Szoveg", 200, 200);
}
}
```

Láthatjuk, a fullscreen alkalmazásunk két fájlból áll. A Screen.java-ban létrehozunk egy ablakot, ami a `window.setUndecorated(true)`; sor miatt válik fullscreenné, ugyanis abban az esetben ha az értékét `false`-ra állítjuk, akkor egy szabadon átméretezhető ablakot fogunk kapni. Hamár itt tartunk érdemes megemlítenünk azt is, hogy az átméretezés is `true/false`-al állítható `window.setResizable()`; segítségével. Ez nekünk `true`-ra (tehát igaz értékre) van állítva, de mivel mi fullscreennel dolgozunk, így az átméretezés lehetőségét nem fogjuk érezni. Fontos még továbbá megemlítenünk, a Screen.java álmányban, a `restoreScreen` metódustami a kijelző visszaállításáért felelős, ugyanis nem elegendő a fullscreen ablak létrehozásáról gondoskodnunk, annak eltüntetéséért is mi leszünk a felelősek.

Most, hogy képesek vagyunk egy FullScreen ablak létrehozására csinálnunk is kéne vele valamit. Arra, hogy mit is kezdjünk vele, egy egyszerű példát láthatunk a bucky.java álmányban. Annyi fog mindösszesen történni, hogy kiírunk egy szöveget a kijelzőre 5 másodpercre, majd ezt követően leáll a programunk. Az ablakunk kinézetét valamint a futási idejét, egy `run` metódusban határoztuk meg. A `getContentPane().setBackground(Color.DARK_GRAY)`; sötétszürke színűre állítja a háttérünket az előteret pedig a `setForeground(Color.WHITE)`;-tal fehérre állítjuk. A `setFont` command segítségével állítjuk be a használandó betűtípust és méretet. A `run` metódusnak a második fele fogja megadni, azt, hogy mennyi ideig fusson a programunk. Ezt egy `try catch` segítségével oldottuk meg. Amennyiben megkapja a displayt, abban az esetben altatásra kerül a szál 5 másodpercig, majd a `restoreScreen`-nel eltünteti az ablakunkat. Végezetül a `paint` metódusunk maradt hátra, mely a megadott szöveg kiírásáért lesz felelős.

Futtatás után az alábbi eredményt kapjuk:



14.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció

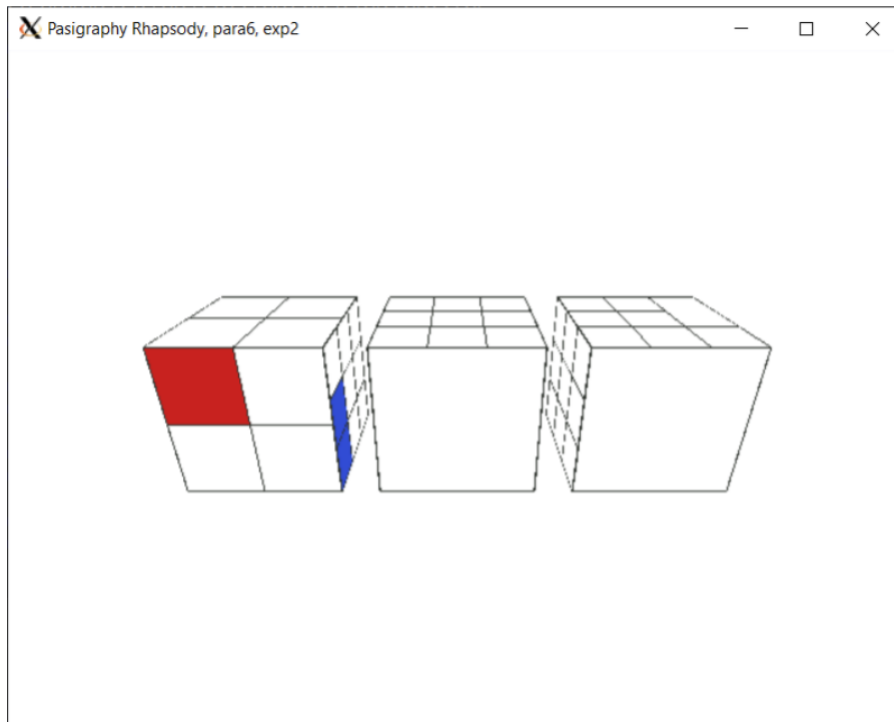
Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Feladatunk első fordítása során hibát tapasztaltunk, ez annak volt köszönhető, hogy a feladat felélesztéséhez szükség van a **sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-commondev** telepítésére. Amennyiben ezzel megvagyunk utána már képesek leszünk fordítani és futtatni a para6.cpp nevű fájlt.

Én személy szerint az irányításon módosítottam a programban, mivel nekem nem volt igazán kézhezálló a billentyűzetes irányítás, ellentétesen volt beállítva. Ezen probléma megoldásához az irányításon kellett módosítanunk, az irányokat meg kellett fordítanunk. Ezt az alábbi kódrészlet segítségével tehetjük meg:

```
void keyboard ( int key, int x, int y )
{
    if ( key == GLUT_KEY_UP ) {
        cubeLetters[index].rotx -= 5.0;
    } else if ( key == GLUT_KEY_DOWN ) {
        cubeLetters[index].rotx += 5.0;
    } else if ( key == GLUT_KEY_RIGHT ) {
        cubeLetters[index].roty += 5.0;
    } else if ( key == GLUT_KEY_LEFT ) {
        cubeLetters[index].roty -= 5.0;
    } else if ( key == GLUT_KEY_PAGE_UP ) {
        cubeLetters[index].rotz -= 5.0;
    } else if ( key == GLUT_KEY_PAGE_DOWN ) {
        cubeLetters[index].rotz += 5.0;
    }
    glutPostRedisplay();
}
```

A színeken is eszközölhetünk változtatásokat, hogy nagyobb kontrasztot érjünk el, ezt úgy próbáltam ki, hogy a `glColor3f` paramétereit állítottam át, a 66., 188., illetve a 220. sorban. Az eredmény az alábbi lett:



14.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció

14.7. Perceptron osztály

14.8. EPAM: Order of everything

Collection-ok rendezése esetén jellemzően futási időben derül ki, ha olyan típusú objektumokat próbálunk rendezni, amelyeken az összehasonlítás nem értelmezett (azaz `T` típus esetén nem implementálják a `Comparable` interface-t). Pl. `ClassCastException` a `Collections.sort()` esetében, vagy `ClassCastException` a `Stream.sorted()` esetében. Írj olyan metódust, amely tetszőleges `Collection` esetén vissza adja az elemeket egy `List`-ben növekvően rendezve, amennyiben az elemek összehasonlíthatók velük azonos típusú objektumokkal. Ha ez a feltétel nem teljesül, az eredményezzen `syntax error`-t. Például: `List Integer actualOutput = createOrderedList(input);` Ahol az `input` `Collection Integer` típusú. Természetesen más típusokkal is működnie kell, feltéve, hogy implementálják a `Comparable` interface-t

Ezt a részt csak elkezdtem mármint az első EPAMOS feladatot, viszont a szünetben még ki fogom egészíteni, illetve amiket nem csináltam meg még eddig feladatokat meg fogom csinálni, hogy teljes legyen a könyv :)

Programunk során generikusokkal fogunk dolgozni. De, hogy mire is jó valójában a generikus? Egyik nagy előnye, az-az, hogy nem kell előre megadnunk azt hogy milyen paramétert várunk, megadhatunk számára egy általános jelölést is, ezáltal képesek leszünk példányosítás esetén ráhivatkozni. Ebben az esetben specifikáljuk ezt az értéket, emiatt egy általánosabb metódust vagy osztályt kapunk majd eredményül. A

fentiekből adódóan képesek leszünk létrehozni egy összehasonlító alkalmazást, ugyanis nem kell megadunk előre, azt, hogy mely elemeket szeretnénk összehasonlítani, hanem generikusok segítségével hivatkozhatunk majd rá. Ezen okból kifolyólag viszont, a létrehozás során külön figyelmet kell fordítanunk az elemek összehasonlíthatóságának vizsgálására is.

14.9. EPAM: Bináris keresés és Buborék rendezés implementálása

14.10. EPAM: Saját HashMap implementáció

15. fejezet

Helló, Stroustrup!

15.1. JDK osztályok

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

A feladat megoldásához szükség van arra hogy telepítsük a boost könyvtárat. Ezt a következő módon tudjuk megejteni: **sudo apt-get install libboost-all-dev** . A telepítés után már lehet is használni a boost könyvtárat, természetesen ehhez meg kell őt hívni. A boost könyvtárra azért van szükség, mivel ennek segítségével határozzuk meg az src elérési útját és annak segítségével megyünk végig az összes almappán és fájlön is. Továbbá a boost segítségével határozzuk meg a fájlok kiterjesztését és azt hogy a vizsgált elem-e, egy fájl-e vagy mappa-e?.

Most, hogy megvan az, hogy mi kell a boostból, kövtekezzen maga a kód. Első lépésként meghatározásra kerül az elérési út. Ha ez megvan akkor egy while ciklus segítségével végig megy a program az összes mappán, almappán és fájlön is. A ciklusban meghatározásra kerül a kiterjesztés majd egy vizsgálat következik. A feltétel melyet vizsgál az hogy a kiterjesztés az .java-e és hogy amit vizsgál az fájl-e. Erre azért van szükség , hogy megvizsgálja hogy fájl-e mert van egy mappa ami .java-ra végzódik. Így ha hiányozna ez a feltétel akkor az **org.graalvm.compiler.java** mappa is megszámlálásra kerülne. Végül pedig kiírja a standart outputra hogy hány darab .java fájlt talált, ezzel megadva a JDK-ban található java osztályok számát.

```
#include <boost/filesystem.hpp> // includes all needed Boost.  ←→
←→
Filesystem declarations
#include <iostream> // for std::cout
using namespace std;
int main(int ac, char** av)
{
    string extension;
    int count = 0;
    boost::filesystem::recursive_directory_iterator iterator(string ←→
    ("/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Stroustrup/ ←→
    jdk_osztaly/src"));
    while(iterator != boost::filesystem:: ←→
```

```
recursive_directory_iterator())
{
    string extension = boost::filesystem::extension(iterator->
    path().filename());
    if(boost::filesystem::is_regular_file(iterator->path()) &&
    extension == ".java"){
        count++;
    }
    ++iterator;
}
cout << count << endl;
return 0;
}
```

15.2. Másoló-mozgató szemantika

Fealadatléírás

15.3. Hibásan implementált RSA törése

Készítsünk 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.

A feladat az, hogy az elrontott RSA kódolással kódolt szöveget betű gyakoriság alapon törjük fel. Az elrontott RSA kódolás azt jelenti, hogy a szöveget nem egybe kódolja le a program hanem karakterenként végzi a kódolást. Így a kódolás nem megfelelően működik. A betű gyakoriság alapú törés pedig azt jelenti, hogy a törő program megvizsgálja a kódolt szöveget majd egy táblázat segítségével, amely tartalmazza a betűk gyakoriságát, visszafejti a kódolt szöveget.

Nézzük meg alaposabban hogy mit is jelent az RSA kódolás. Az RSA kódolás az aszimmetrikus kódolások közé tartozik. Ennek lényege hogy a kódolás és dekódolás során két kulcsra van szükségünk. Egy nyilvános kulcsra és egy privát kulcsra. A nyilvános kulcs segítségével lehet titkosítani az üzeneteket, míg a megfejtéshez a privát kulcsra van szükségünk. Egy üzenet küldése az alábbi módon néz ki:

Van két ember A és B. Ha A üzenetet akar küldeni B-nek akkor A elkéri B nyilvános kulcsát. Ennek segítségével titkosítja a küldeni kívánt üzenetet majd a kódolt szöveget elküldi. A feltöréshez pedig B-nek a privát kulcsra van szükség, ugyanis csak azzal lehet visszafejti a kódolt szöveget. Ha valaki egy másik kulccsal próbálja meg megfejteni a szöveget akkor az eredmény hibás lesz.

Az RSA eljárás rendkívül hatékony titkosítási módszer ám a megfejtése bonyolultabb mint egy szimmetrikusan kódolt szövegé. Ebből adódik hogy biztonságosabb is. Az eljárás apáját a moduláris számelmélet és a prímszámelmélet néhány tétele adja.

Most lássuk a titkosító kódot:

```
import java.math.BigInteger;
```

```
import java.security.SecureRandom;
import java.util.*
public class RSA {
    private final static BigInteger one = new BigInteger("1");
    private final static SecureRandom random = new SecureRandom();
    private BigInteger privateKey;
    private BigInteger publicKey;
    private BigInteger modulus;
    RSA(int N) {
        BigInteger p = BigInteger.probablePrime(N/2, random);
        BigInteger q = BigInteger.probablePrime(N/2, random);
        BigInteger phi = (p.subtract(one)).multiply(q.subtract(one));
        modulus = p.multiply(q);
        publicKey = new BigInteger("65537");
        privateKey = publicKey.modInverse(phi);
    }
    BigInteger encrypt(byte[] bytes) {
        BigInteger swap = new BigInteger(bytes);
        return swap.modPow(publicKey, modulus);
    }
}
```

Látható hogy a kód elején importolva lett a `BigInteger`, a `SecureRandom` és a `util`. A `BigInteger` ahhoz szükséges hogy nagy számokkal lehessen dolgozni. Erre azért van szükség mert a kulcsok amikkel a program dolgozik túl nagyok és az `Integer` határain belül nem lehetne eltárolni a szükséges számokat. A `SecureRandom` egy a kódoláshoz használható biztonságos számot állít elő. Ez arra jó, hogy a szám amit generál nehezen feltörhető. Végül a `util.*`-ra azért van szükség mert a mainben az eredmény egy `List`-ben kerül tárolásra.

Az RSA eljárásban kerül létrehozásra a publikus kulcs és a privát kulcs. Elsőnek a `p` és `q` változóknál eltárolunk két valószínűleg prím számot a `probablePrime` segítségével. Ez után meghatározzuk a `phi` és a modulus értékét. Végül pedig létrehozuk a publikus kulcsot és meghatározzuk a privát kulcsot úgy, hogy vesszük a publikus kulcs reciprokát moduló `phi`.

Az `encrypt` függvény végzi a titkosítást. Megkapja a `byte`-ba átváltott karaktert majd azt eltárolja a `swap`-ben. Erre azért van szükség mert a `modPow` csak `BigInteger`-en alkalmazható. A visszatérítésnél pedig megtörténik a titkosítás a `modPow` segítségével. A `swap`-be lévő értéket a publikus kulcsra emeli, majd modust hajt végre.

```
public static void main(String[] args) {
    int N = Integer.parseInt(args[0]);
    RSA key = new RSA(N);
    System.out.println("key = " + key);
    String s = "test";
    byte[] bytes = s.getBytes();
    List<BigInteger> result = new ArrayList<BigInteger>();
    byte[] atmenet = new byte[1];
    for(int i = 0; i < bytes.length; i++)
    {
        atmenet[0] = bytes[i];
    }
}
```

```
result.add(key.encrypt(atmenet));
}
System.out.println("message = " + s);
System.out.println("encrypted = " + result);
}
```

Most pedig lássuk a main-t. Eloszor a parancssorban megadott paramétert eltároljuk az N-ben majd ezt felhasználva legeneráljuk a kulcsot. Ha ez megvan akkor az s változóban megadjuk, hogy mi az a szöveg amit titkosítani akarunk. Ezt követően a bytes tömbbe belekonvertáljuk az s-ben lévő szöveget úgy, hogy byte típusú legyen. A resultban kerül majd tárolásra az eredmény. A for ciklus segítségével végig megyünk a bytes tömbön és egyesével hozzáfűzzük a result-hoz a kódolt karaktereket. A ciklus végére az egyesével kódolt karakterek lesznek eltárolva a resultba. A program végén pedig kiírjuk az eredményt.

Ezzel végeztünk a kódolással. Most nézzük meg a szöveg dekódolását.

```
class KulcsPar{
private String values;
private char key = '_';
private int freq = 0;
public KulcsPar(String str){
this.values = str;
}
public void setValue(String str){
this.values = str;
}
public void setKey(char k){
this.key = k;
}
public String getValue(){
return this.values;
}
public char getKey(){
return this.key;
}
public void incFreq(){
freq += 1;
}
public int getFreq(){
return freq;
}
}
```

A dekódoló program két fájlból áll. Az egyik a KulcsPar.java a másik az RsaTores.java. Elsőnek lássuk a KulcsPar.java-t. Ez a fájl egy osztályt tartalmaz melyben három private változó található. Továbbá tartalmaz metódusokat melyek arra szolgálnak hogy a változók értékeit az osztályon kívülről is lehessen módosítani illetve lekérni.

```
BufferedReader inputStream = new BufferedReader(new FileReader ←  
    ("be2 ←←  
    .txt"));  
int lines = 0;  
String line[] = new String[10000];  
while((line[lines] = inputStream.readLine()) != null) {  
    lines++;  
}  
inputStream.close();
```

Most pedig lássuk az RsaTores.java-t. Először a `BufferedReader` segítségével a `be2.txt` fájlból beolvassuk a titkosított szöveget. Ez úgy történik, hogy a `while` ciklus segítségével soronként olvassuk a fájlt amíg az nem null vagyis véget nem ér a fájl. A ciklusmagban pedig növeljük a `lines` változót ami megadja a sorok számát.

```
KulcsPar kp[] = new KulcsPar[100];  
boolean volt = false;  
kp[0] = new KulcsPar(line[0]);  
int db = 1;  
for(int i = 1; i < lines; i++) {  
    volt = false;  
    for(int j = 0; j < db; j++) {  
        if(kp[j].getValue().equals(line[i])) {  
            kp[j].incFreq();  
            volt = true;  
            break;  
        }  
    }  
    if(volt == false) {  
        kp[db] = new KulcsPar(line[i]);  
        db++;  
    }  
}
```

Ezt követően létrehozunk `kp` nevű tömböt ami a `KulcsPar` osztályba tartozik. Ennek első elemére beállítjuk az első titkosított karaktert és létrehozuk a `db` változót aminek kezdőértéke egy. Ha ez megvan akkor végig megyünk a sorokon és végignézzük a beolvasott fájlt. Ha egy adott karaktert megtalál akkor annak a darab számát növeli. Ha olyan karaktert talált amilyen eddig még nem szerepelt akkor azt új értéként felveszi a `kp` tömbbe és annak is növeli a darabszámát eggyel. Ez addig meg míg a ciklus végig nem ér az összes soron.

15.4. 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/Per-

ceptron osztály feladatát is.)

A feladat során a tavalyi Perceptron feladatot kellett tovább fejleszteni. A lényeg az hogy a tavalyi Perceptron feladat a számítások során egyetlen értékre szűkítette le a bemeneti képet, most viszont ezen módosítani kellett. A bemeneti képet eloször le kell szűkíteni 256 értékre majd ebből kell visszaalakítani egy az eredetivel megegyező méretű képpé az értékeket.

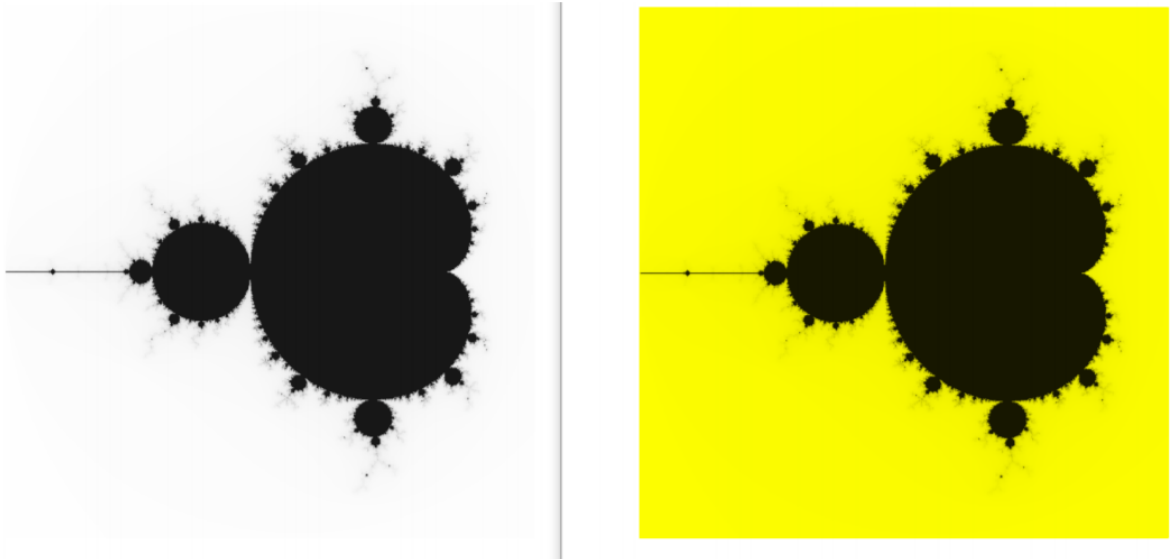
```
#include <iostream>
#include "mlp.hpp"
#include <png++/png.hpp>
#include <fstream>
int main (int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width() *png_image.get_height();
    Perceptron* p = new Perceptron (3, size, 256, size);
    double* image = new double[size];
    for (int i {0}; i<png_image.get_width(); ++i)
    for (int j {0}; j<png_image.get_height(); ++j)
    image[i*png_image.get_width() +j] = png_image[i][j].red <--
    ;
    double* newimage = (*p) (image);
    for (int i = 0; i<png_image.get_width(); ++i)
    for (int j = 0; j<png_image.get_height(); ++j)
    png_image[i][j].blue = newimage[i*png_image.get_width() <--
    +j];
    png_image.write("output.png");
    delete p;
    delete [] image;
}
```

Látható hogy a mainben létrehozott perceptron osztályú példány létrehozáskor meghívja a konstruktort. A konstruktornak 4 paramétert adunk meg. Az első érték azt adja meg hogy hány értéket kap a konstruktor. Majd ezt követően megadjuk a paramétereket. Az első paraméter a size. Ez azt határozza meg hogy mekkora a bementi kép. Majd a 256 megadja hogy hány értékre szűkítse a perceptron a képet, végül pedig megadjuk hogy az eredményként visszaadott kép mekkora méretű legyen. Most viszont nézzük meg magát a konstruktort. A main további részében látható ahogy a kép red értékei mentésre kerülnek az image változóban, majd ezt felhasználva az eredeti kép blue értékeit felülírjuk a kimentett értékekkel. Végül az output.png-be kiírjuk az új képet és a perceptron példányt valamint az image tömböt töröljük. Ahhoz hogy megfelelően működjön az új képet generáló Perceptron program a header fájlban is kell módosítanunk. Elsőnek a konstruktoron kell módosítani.

```
Perceptron ( int nof, ... )
{
    n_layers = nof;
    units = new double*[n_layers];
    n_units = new int[n_layers];
    va_list vap;
```

```
va_start ( vap, nof );
for ( int i {0}; i < n_layers; ++i )
{
    n_units[i] = va_arg ( vap, int );
    if ( i )
        units[i] = new double [n_units[i]];
}
va_end ( vap );
weights = new double**[n_layers-1];
#ifdef RND_DEBUG
    std::random_device init;
    std::default_random_engine gen {init() };
#else
    std::default_random_engine gen;
#endif
std::uniform_real_distribution<double> dist ( -1.0, 1.0 );
for ( int i {1}; i < n_layers; ++i )
{
    weights[i-1] = new double *[n_units[i]];
    for ( int j {0}; j < n_units[i]; ++j )
    {
        weights[i-1][j] = new double [n_units[i-1]];
        for ( int k {0}; k < n_units[i-1]; ++k )
        {
            weights[i-1][j][k] = dist ( gen );
        }
    }
}
```

Látható hogy a módosított konstruktor már nem fix paraméterkészlettel dolgozik hanem változó számúval. Elsonek beolvassa a változók számát, majd létrehoz egy akkora tömböt melyben el tudja tárolni a kapott értékeket. Az `n_units`-ban eltároljuk a szintekhez tartozó unitok számát majd ennek segítségével feltöltésre kerül a `weights`. A másik lényegi változás a `double*` operator()-nál történt. Ez most már egy tömbbel dolgozik és azt is térít vissza. Erre azért volt szükség hogy módosítani tudjunk a képen. Végül az eredmény az abábbi módon néz ki:



15.5. Összefoglaló

RSA törőbe integrálva.

15.6. EPAM: It's gone. Or is it?

Adott a következő osztály. Mutassunk példát olyan esetre, amikor meghívjuk az osztály `writeStuff()` metódusát, de a program lefutása után, a fájl, amibe írtunk mégis üres.

```
public class BugousStuffProducer {  
  
    private Writer writer;  
  
    public BugousStuffProducer(String outputFileName) throws IOException {  
        writer = new FileWriter(outputFileName);  
    }  
  
    public void writeStuff() throws IOException {  
        writer.write("Stuff");  
    }  
  
    public void finishProducingStuff() throws IOException {  
        writer.close();  
    }  
  
    public static void main(String[] args) throws IOException {  
        BugousStuffProducer stuffProducer = new BugousStuffProducer("file. ↵  
        txt");  
    }  
}
```

```
        stuffProducer.writeStuff();  
  
    }  
  
}
```

Adjunk meg rögtön az osztályunknak egy main metódust:

```
public static void main(String[] args) throws IOException {  
  
    BugousStuffProducer stuffProducer = new BugousStuffProducer("file. ↵  
        txt");  
    stuffProducer.writeStuff();  
  
}
```

A kódunk futása során létrejött file üres lesz, hasonlóan az alap filehoz, ennek az az oka, hogy a `FileWriter` objektumunknak nem hívtuk meg a `close()` metódusát, amely lezárná az adatfolyamot. Egy file írásra való megnyitása során, az adatok bufferelve vannak, mely azt jelenti, hogy addig nem lesznek beleírva a tényleges fájlba, míg a rendszer nem bizonyosodott meg arról, hogy többet már nem szeretnénk írni a programunkba. Amennyiben ezt nem jelezzük egy lezárással, abban az esetben, az adatok semelyik esetben sem fognak bekerülni célállományba.

Erre a problémára egyfajta megoldás lehet, amennyiben megpróbálunk nagyobb figyelmet fordítani a meghívni a `close()` metódus meghívására, vagy ezen kívül használhatjuk a java interfészét (ez is egy létező megoldás lehet..). Nézzük is meg a kódunkat picikét átalakítva:

15.7. EPAM: Kind of equal

Fealadatleírás

15.8. EPAM: Java GC

Fealadatleírás

16. fejezet

Helló, Gödel!

16.1. Gengszterek

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

A feladatban az OOCWC projektben lévő gengsztereket kell rendeznünk. A rendezés szempontja az, hogy melyik található a legközelebb a rendőrhöz. Lássuk az ehhez használt kódrészletet:

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

Mint azt láthatjuk, a sorban metódus segítségével kerülnek rendezésre a gengszterek egy lambda segítségével. A metódus, két gengszter osztályú változót kap értékül, ennek segítségével dolgozik. A lambda testében egy feltétel található. Abban az esetben kerül igaz (true) érték visszaadásra, ha az x gengszter közelebb van a rendőrhöz - vagyis cop-hoz - mint az y. Tehát, amikor a metódus végig megy a gengszterek vektoron, abban az esetben, az kerül a vektor elejére, aki a legközelebb lesz a cop-hoz.

16.2. C++11 Custom Allocator

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

Ezen feladatunkban, egy saját allokátor elkészítése volt a feladat. Az allokátoroknak a lényege tulajdonképpen az, hogy helyet biztosítson a memóriában egy adott típus számára. Erre létezik alapértelmezett allokátor is, viszont amennyiben úgy gondoljuk, hogy szeretnénk egy saját allokátort készíteni, erre is lehetőségünk van.. Lássuk tehát a saját allokátorunkat:

```
template<typename T>
```

```

class CustomAlloc
{
public:
    CustomAlloc() {}
    CustomAlloc(const CustomAlloc&) {}
    ~CustomAlloc() {}
    using size_type = size_t;
    using value_type = T;
    using pointer = T*;
    using const_pointer = const T*;
    using reference = T&;
    using const_reference = const T&;
    using difference_type = ptrdiff_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;
        delete p;
        return reinterpret_cast<T*>(new char[n*sizeof(T)]);
    }
    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;
    }
};

```

A saját allokátorkunk előtt található egy template. Ez arra szolgál, hogy egy típust paraméterként át tudjunk adni, ezáltal létrehozva egy általános metódust, mely bármilyen típusal használható lesz. Az allokátor előtt létrehozott változók típusa a using. Ez egy típus alternatíva létrehozásához használható. Erre példa a reference. Mostantól ha ez kerül a kódba az a T és-t fog jelenteni. Most viszont lássuk az allokátorunkat: A char* p-ben eltároljuk a típust mellyel jelenleg dolgozik az allokátor. Ehhez a __cxa_demangle metódust használjuk. Ez arra szolgál hogy a typeid (T).name() által meghatározott mangolt nevet visszafejtsük és meghatározzuk belőle a típust. Erre a nyomkövetés miatt van szükségünk. Ezután, a nyomkövetés céljából kiírásra kerül egy sor majd a return-höz érve a reinterpret_cast segítségével megtörténik a memóriában a helyfoglalás. A deallocate az allocate ellentéte. Arra szolgál, hogy a már használaton kívüli memória területet felszabadítsa. A delete végzi a felszabadítást, és a nyomkövetés céljából itt is történik egy kiírás.

16.3. STL map érték szerinti rendezése

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

A feladat az hogy érték szerint rendezzük a map-ot. Ebben az a csavar hogy a map alapvetően kulcs szerint rendez. Ahhoz hogy érték szerint lehessen rendezni egy kis trükköt kell alkalmaznunk:

```
std::vector<std::pair<std::string, int>> sort_map ( std::map < ↵
    std::
string, int> &rank )
{
    std::vector<std::pair<std::string, int>> ordered;
    for ( auto & i : rank ) {
        if ( i.second ) {
            std::pair<std::string, int> p {i.first, i.
            second};
            ordered.push_back ( p );
        }
    }
    std::sort (
        std::begin ( ordered ), std::end ( ordered ),
        [= ] ( auto && p1, auto && p2 ) {
            return p1.second > p2.second;
        }
    );
    return ordered;
}
```

A trükk az hogy először áteszi a map kulcs érték párait egy vektorba ami használja a pair-t. A pair segítségével lehetőségünk van különböző típusú értékek páronkénti tárolására. Erre azért van szükség, mert a vektorban eltárolva már végre lehet hajtani az érték szerinti rendezést. A vektorba történő áthelyezést a for ciklus végzi. Végig megyünk a paraméterként kapott mapon és ha a kulcshoz tartozik érték is, akkor a p-be beletöltjük a párt majd a push_back segítségével a p-t hozzáfűzzük a vektorhoz. Miután áthelyezésre kerülnek a párok a sort metódus segítségével elvégzésre kerül a rendezés. Ahhoz viszont hogy érték szerint legyen rendezve a Gengszterek feladatban megismert lambdára fogunk támaszkodni. Végig megyünk a létrehozott vektoron az elejétől a végéig, úgy hogy az értékeket átmásoljuk nem pedig referenciaként hivatkozunk rájuk. A lambda segítségével meg kell határozni hogy a rendezés az érték szerint legyen. Ez úgy történik hogy a return a second-ot vizsgálva egy bool értéket ad meg az alapján, hogy az összehasonlított p1.second nagyobb-e mint a p2.second. Ha ez igaz akkor a bool is igaz értéket ad.

16.4. Alternatív Tabella rendezése

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

A feladat során a Cspat osztály definíciójához kellett felhasználnunk a Comparable interfészt. Erre azért volt szükség, hogy a rendezCspatok metódusban a sort függvény kulcsként használhassa, úgy hogy nem kell külön megadni összehasonlított. Lássuk hogy hogyan néz ki a programban.:

```
public static void rendezCsapatok(String[] s, double h[], ←
    String[] e, int ep[]) {
    System.out.println("\nCsapatok rendezve:\n");
    int csapatNevekMeret = h.length;
    Csapat csapatok[] = new Csapat[csapatNevekMeret];
    for (int i = 0; i < csapatNevekMeret; i++) {
        csapatok[i] = new Csapat(s[i], h[i]);
    }
    java.util.List<Csapat> rendezettCsapatok = java.util.Arrays. ←
        asList(csapatok);
    java.util.Collections.sort(rendezettCsapatok);
    java.util.Collections.reverse(rendezettCsapatok);
    java.util.Iterator iterv = rendezettCsapatok.iterator();
    ....
    ....
    ....
    class Csapat implements Comparable<Csapat> {
    protected String nev;
    protected double ertekek;
    public Csapat(String nev, double ertekek) {
    this.nev = nev;
    this.ertekek = ertekek;
    }
    public int compareTo(Csapat csapat) {
    if (this.ertekek < csapat.ertekek) {
    return -1;
    } else if (this.ertekek > csapat.ertekek) {
    return 1;
    } else {
    return 0;
    }
    }
    }
}
```

A Comparable interfész arra szolgál hogy általunk létrehozott típusokat tudjunk vele rendezni. Az Interface Comparable T-ben a T helyére lehet behelyettesíteni azt az objektum típust amihez hasonlítani szeretnénk az interfészt használó objektumot. Jelen esetben ez azt jelenti hogy a Csapat-hoz szeretnénk hasonlítani. A rendezést a compareTo metódus végzi amely paraméterként megkapja a csapatot. A compareTo három különböző értékkel térhet vissza. Ha az aktuális objektum nagyobb mint a paraméterben kapott, akkor a visszatérési érték 1 lesz. Ha kisebb akkor -1, ha pedig egyenlő, abban az esetben 0. Egy lista vagy egy tömb amit ezekből az objektumokból készítünk, ha a sort-tal rendezzük akkor fel fogja használni a compareTo-t is.

```
iteracio...
norma = 0.05918759643574294
osszeg = 1.0
iteracio...
norma = 0.014871507967965858
osszeg = 0.9999999999999999
iteracio...
norma = 0.006686856768932613
osszeg = 1.0
iteracio...
norma = 0.007776749198562133
osszeg = 1.0
iteracio...
norma = 0.007661483555477314
osszeg = 0.9999999999999999
iteracio...
norma = 0.007581657116770109
osszeg = 0.9999999999999998
iteracio...
norma = 0.007532798726590474
osszeg = 0.9999999999999998
iteracio...
norma = 0.007538144256251714
osszeg = 0.9999999999999998
iteracio...
norma = 0.007535825315190922
osszeg = 1.0
iteracio...
norma = 0.00753510193655861
osszeg = 0.9999999999999997
iteracio...
norma = 0.0075353297234758716
osszeg = 0.9999999999999998
iteracio...
norma = 0.007535284725802345
osszeg = 0.9999999999999999
iteracio...
norma = 0.007535275242694286
osszeg = 0.9999999999999996
iteracio...
norma = 0.0075352801728795745
osszeg = 0.9999999999999999
iteracio...
norma = 0.007535280078749908
osszeg = 0.9999999999999998
iteracio...
norma = 0.007535279718816022
osszeg = 0.9999999999999998
iteracio...
norma = 0.007535279786665789
osszeg = 0.9999999999999998
iteracio...
norma = 0.007535279802432719
```

16.5. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témát (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

A **GIMP**(GNU Image Manipulation Program) egy képszerkesztő program. A legtöbb platformon elérhető, ami elősegítette világraszóló terjedését. A programban, különböző képszerkesztési eszközök állnak rendelkezésünkre, pl: rajzolhatunk és színezhetünk akár átlátszóvá is tehetünk egy képet, támogatja a vektorgrafikát, valamint fontos, hogy rétegekkel dolgozik. A **Script-Fu** a Scheme szkriptnyelven alapul, melyet már korábban (a 9. fejezetben) tárgyalt **Lisp** programozási nyelvekhez tartozik. A program maga ezen nyelven íródott, melynek lényege, hogy a bemenetként kapott szöveget átalakítsa, ezen keresztül effektet tegyen rá.

A GIMP programot fogjuk használni a mandala szerkesztéséhez. A mandala.scm fájlt berakjuk a usr/share/gimp/2.0/scripts mappába, majd a GIMP-ben megnyitjuk

16.6. EPAM: Mátrix szorzás Stream API-val

Matematikából már tudhatjuk, hogy mátrixszorzás feltétele, az első mátrix oszlopainak száma megegyezzen a másik mátrix sorainak számával. Az eredménymátrix sorainak a száma az első mátrixé lesz, és oszlopainak száma a másodiké. Nézzük is meg a kódot: Egy Matrix osztályt hozunk létre, és ennek lesznek metódusai illetve attribútumai. Láthatjuk, hogy van egy kétdimenziós tömbünk, ez lesz a mátrixunk, emellett van két egészünk (intünk) is, ezek a sor és oszlopok számát fogják megadni.

```
public class Matrix {
```

```
public int[][] matrix;  
public int rowsLenght;  
public int columnsLenght  
  
public Matrix(int[][] matrix) {  
    this.matrix = matrix;  
    this.rowsLenght = matrix.length;  
    this.columnsLenght = matrix[0].length;  
}
```

Nézzük a szorzó metódust:

```
public Matrix multiply(Matrix input) {  
    int[][] result = Arrays.stream(this.matrix)  
        .map(r -> IntStream.range(0, input.columnsLenght)  
            .map(i -> IntStream.range(0, input.rowsLenght).map(↔  
                j -> (r[j] * input.matrix[j][i])).sum())  
            .toArray())  
        .toArray(int[][]::new);  
    return new Matrix(result);  
}
```

A feladat azt kérte, hogy for és while ciklusok használata nélkül írjuk meg ezt a methodot. Ezt streamekkel, és lambda kifejezésekkel tehetjük meg. Először is átkonvertáljuk streammé az első mátrixunkat, ezek által lesz egy streamünk, aminek minden eleme egy tömb lesz, és a számossága a streamnek egyenlő lesz a végeredmény mátrix sorainak a számával. Ezek után a map methoddal a streamünk minden elemét tudjuk módosítani. Célunk, hogy a meglévő tömbök számát ne, csak számosságát és értékeit változtassuk meg. Ha a számosságát változtatjuk meg ezeknek a tömböknek a második mátrixunknak oszlopainak számára akkor az egyenlő lesz a végeredményként kapni kívánt mátrixunknak oszlopainak számával. Először is az IntStream.range methoddal egy olyan streamet hozunk létre, aminek az elemei egyesével növekednek, a két attribútumként megadott szám között. Az első attribútum szereplni fog ebben a streamben, a második viszont már nem. Majd ezeket az értékeket (oszlopokat) fogjuk módosítani ugyanúgy a map methoddal. Felveszünk egy i változót, ami 0-tól az első mátrixunk oszlopainak a számáig (vagy a második mátrixunk sorainak a számáig) fog menni, ugyanúgy az IntStream.range methoddal. Ebben lesznek majd azok az elemek, amiket páronként össze fogunk szorozni (a map methoddal módosítani), majd később ezeket összeadni (a sum methoddal). Ezekért ez a kódcsipet fog felelni:

```
.map(i -> IntStream.range(0, input.rowsLenght).map(j -> (r[j] * input.↔  
    matrix[j][i])).sum())
```

Ezzel a kódrészlettel az oszlopokat, vagyis már a tényleges értékeket fogjuk módosítani, emiatt egy számot kell értékül kapnunk, és a sum miatt azt is fogunk kapni. Az e feletti mappal pedig a sorokat fogjuk módosítani amik tömbök lesznek, emiatt a stream tömbbé fogjuk átalakítani a toArray() methoddal. Majd végül a

tömbökből álló streamet visszakonvertáljuk egy kétdimenziós tömbbé, egy mátrixxá `.toArray(int[][]::new)` methoddal.

17. fejezet

Helló, hetedik hét!

17.1. 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 feladat megoldásához szükségünk van a JavaFX könyvtárra. A JavaFX azonban az sdk8-ban volt elérhető, tehát a használatához szükségünk lesz az sdk8-ra. Ahhoz, hogy a meglévő sdk-t lecseréljük a megfelelőre bátran használhatjuk az SDKMAN-t. Ennek segítségével képesek leszünk átváltani sdk8-ra, így könnyen megoldhatóvá válik a feladat.

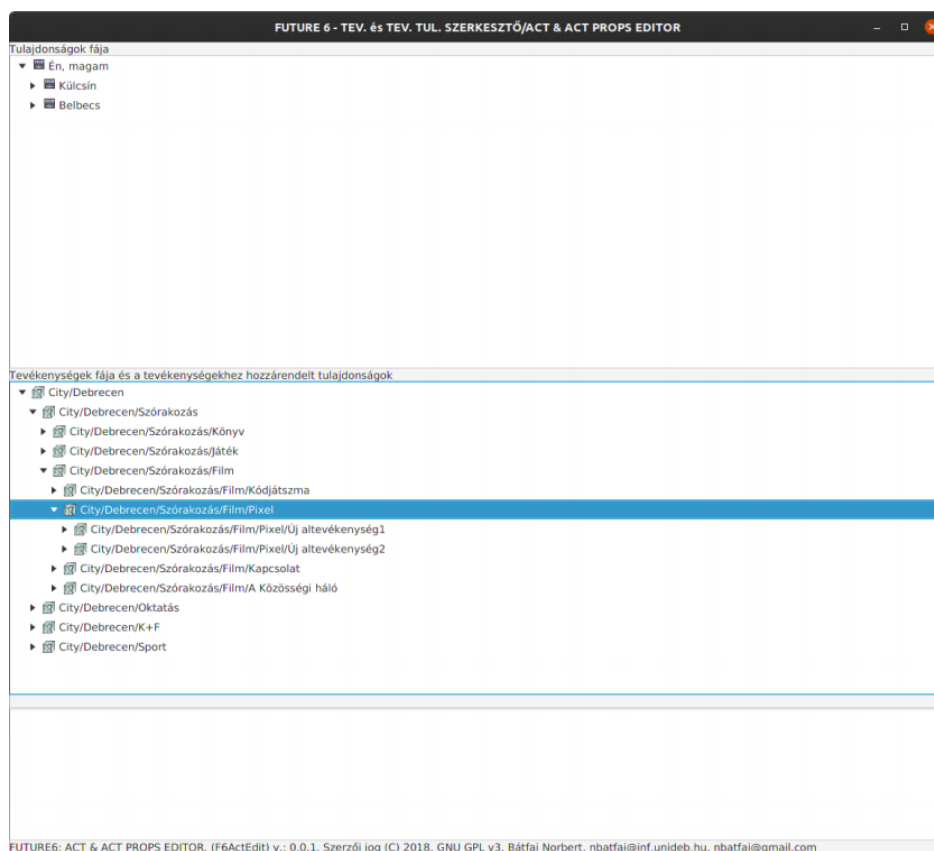
Miután már rendelkezünk a feladat megoldásához szükséges sdk verzióval a program futtathatóvá válik. A futtatást sikeresen elvégezhetjük, mivel nem kapunk szintaktikai hibát. Azonban ezzel szemben található benne szemantikai hiba. Az egyik ilyen hiba, új altevékenység létrehozásánál ütközik ki, ugyanis a programunk nem tud több új altevékenységet létrehozni. Ahhoz hogy ez megoldható legyen szükségünk lesz némi módosításra a kódban. Az altevékenység létrehozásnál meg kell adnunk egy egy while ciklust és növelnünk kell az altevékenységek számát is. Ha sikeresen létrejön az altevékenység, akkor kilépünk a ciklusból egy break utasítással, ha viszont sikertelen a létrehozás akkor növeljük az i értéket és újra indul a ciklusunk.

```
javafx.scene.control.MenuItem subaMenuItem = new javafx. ↵  
    scene.  
control.MenuItem("Új altevékenység");// "New  
//  
  
subactivity  
")  
;  
addMenu.getItems().add(subaMenuItem);  
subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> ↵  
    {  
        java.io.File file = getTreeItem().getValue();  
        int i = 1;  
        while (true) {  
            java.io.File f = new java.io.File(  
                file.getPath() + System.getProperty("file.separator
```

```

    ") + "Új altevékenység" + i); // PLACEHOLDER
    //
    HERE
    if (f.mkdir()) {
        javafx.scene.control.TreeItem<java.io.File> newAct
        // rr.println("Cannot create " + f.getPath())rr.println
        ("Cannot create " +
        // f.getPath())rr.println("Cannot create " + f.getPath
        ())rr.println("Cannot
        // create " + f.getPath()) = new javafx.scene.control.
        TreeItem<java.io.File>(f,
        // new javafx.scene.image.ImageView(actIcon));
        = new FileTreeItem(f, new javafx.scene.image.ImageView(
        actIcon));
        getTreeItem().getChildren().add(newAct);
        break;
    } else {
        i++;
        System.err.println("Cannot create " + f.getPath());
    }
    }
    });

```



17.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocaremulator/blob/master/justine/rcemu/src/carlexer.ll>

```
while ( std::sscanf ( data+nn, "<OK %d %u %u %u>%n", &idd, ←
    &f, &t, ←
    &s, &n ) == 4 )
{
    nn += n;
    gangsters.push_back ( Gangster {idd, f, t, s} );
}
```

Ebben a kódrészletben az sscanf található meg. Ez abban különbözik a scanf-tól hogy a formázott szövegből olvasunk be. Az hogy a szöveg milyen módon van megformázva az idézőjelek között található meg: **OK %d %u %u %u** Ez alól azonban kivétel, az idézőjelben lévő szöveg végén található %n. Ez ugyanis azt mutatja meg, hogy hány paraméter lett beolvasva. A sor végi feltétel vizsgálat is ehhez kapcsolódik. Ugyanis, ha az egyenlőség teljesül, abban az esetben ez az azt jelenti, hogy mind a négy paraméter beolvasása sikeres. A sscanf elején található data+nn azt adja meg, hogy honnan kezdődjön meg a beolvasás. Ahhoz hogy ez mindig más értéket adjon és ne olvassuk többször ugyan azt a sort, az nn értékét növelnünk kell. Ezt a ciklusmagban tehetjük meg. Az nn-hez mindig hozzá kell adnunk a beolvasott adatok mennyiségét, ezáltal a data+nn, mindig a beolvasandó rész elejére fog mutatni. A ciklusmagban pedig a push_back segítségével a kódunk a gangsters-hez hozzáfűzi a beolvasott adatokat.

17.3. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuCam>

```
#include "SamuCam.h"
SamuCam::SamuCam ( std::string videoStream, int width = ←
    176, int
    height = 144 )
: videoStream ( videoStream ), width ( width ), height ( ←
    height )
{
    openVideoStream();
}
SamuCam::~SamuCam ()
{
}
void SamuCam::openVideoStream()
{
    videoCapture.open ( videoStream );
    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
```

```
videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
videoCapture.set ( CV_CAP_PROP_FPS, 10 );
}
```

A program azzal kezdődik, hogy a parancssorban megkapott IP címet a használni kívánt webkamerának átadjuk paraméterként. Ezután a `videoCapture.open (videoStream);` segítségével megnyitjuk a webkamerát majd a `.set` metódusok segítségével beállítjuk a szélességet, a magasságot és végül megadjuk az FPS értékét is.

```
void SamuCam::run()
{
    cv::CascadeClassifier faceClassifier;
    std::string faceXML = "lbpcascade_frontalface.xml"; // ↵
    https://
    github.com/Itseez/opencv/tree/master/data/lbpcascades
    if ( !faceClassifier.load ( faceXML ) )
    {
        qDebug() << "error: cannot found" << faceXML.c_str();
        return;
    }
    cv::Mat frame;
```

Miután a webkamera meg lett nyitva és az értékei be lettek állítva, szükség van egy `CascadeClassifier`-re amely a beolvasa `lbpcascade_frontalface.xml`-t. Ennek segítségével lehet elvégezni a képek elemzését. A `.load` metódus olvassa be az emberi arcot, és ha ez nem sikerül akkor hibát ad eredményül.

```
while ( videoCapture.isOpened() )
{
    QThread::msleep ( 50 );
    while ( videoCapture.read ( frame ) )
    {
        if ( !frame.empty() )
        {
            cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0,
            cv::INTER_CUBIC );
            std::vector<cv::Rect> faces;
            cv::Mat grayFrame;
            cv::cvtColor ( frame, grayFrame, cv::COLOR_BGR2GRAY );
            cv::equalizeHist ( grayFrame, grayFrame );
            faceClassifier.detectMultiScale ( grayFrame, faces,
            1.1, 4, 0, cv::Size ( 60, 60 ) );
            if ( faces.size() > 0 )
            {
                cv::Mat onlyFace = frame ( faces[0] ).clone();
                QImage* face = new QImage ( onlyFace.data,
                onlyFace.cols,
                onlyFace.rows,
```

```

        onlyFace.step,
        QImage::Format_RGB888 )
    ;
    cv::Point x ( faces[0].x-1, faces[0].y-1 );
    cv::Point y ( faces[0].x + faces[0].width+2, faces
    [0].y + faces[0].height+2 );
    cv::rectangle ( frame, x, y, cv::Scalar ( 240, 230,
    200 ) );
    emit faceChanged ( face );
}
QImage* webcam = new QImage ( frame.data,
    frame.cols,
    frame.rows,
    frame.step,
    QImage::Format_RGB888 )
;
emit webcamChanged ( webcam );
}
QThread::msleep ( 80 );
}
if ( ! videoCapture.isOpened() )
{
    openVideoStream();
}
}
}

```

Miután a beolvasás megtörténik a képet átméretezi és interpolálja a kód. Ezután a színét szürkére állítja majd az `equalizeHist` metódus segítségével kiegyenlíti a képek hisztogramját. Ezt követően a `detectMultiScale` segítségével arcokat keres a képen. Ha talál arcot akkor létrehoz belőle egy `QImage`-t majd az `emit` segítségével küld egy jelet amit a `SamuBrain` fog feldolgozni. Ezt követően létrejön még egy `QImage` amit még egy signal küldés is követ. Végül 80 milliszekundum várakozás után a ciklus visszatér az elejére. A ciklusból abban az esetben fogunk kilépni, ha a `videoCapture` bezárásra kerül.

17.4. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esport-talentsearch>

Ebben a feladatban slot-signal mechanizmussal imerkedünk meg. A slot-signal mechanizmus arra szolgál, hogy objektumok között lehessen kommunikálni. A `BrainBWin.cpp`-ben található meg ennek a kód részlete.

```

connect ( brainBThread, &BrainBThread::heroesChanged ,
    this, &BrainBWin::updateHeroes );
connect ( brainBThread, &BrainBThread::endAndStats,
    this, &BrainBWin::endAndStats );

```

Alapvetően egy eseményhez kötött esemény zajlik le, melyet a connect köt össze. Tehát a brainBThread küld egy SIGNAL-t az emit segítségével, amit a connect felfog és ennek hatására fut le a megadott metódus. Jelen példában kettő connect is található. Az első esetben érkezik egy heroesChanged signal melynek hatására lefut az updateHeroes metódus. A másik esetben pedig az endAndStats signal érkezik melynek hatására az endAndStats metódus fut le.



17.5. OSM térképre rajzolása

Fealadatleírás

17.6. EPAM: XML feldolgozás

Fealadatleírás

17.7. EPAM: ASCII Art

Az ASCII vel készített képek (Art-ok) napjainkban igencsak népszerűvé váltak, számos érdekes, akár vicces képet is készíthetünk ASCII segítségével.

Nem lehetetlen megcsinálnunk, azonban sok időt venne igénybe, ezért inkább nézzük meg kód segítségével: A kódunk beolvassa a képet, figyelniünk kell arra, hogy jó filenevet adjunk meg, mivel gyakran, ilyen okokból kifolyólag fog rosszul működni a programunk..

```
try {  
    img = ImageIO.read(new File("teszt2.jpg"));  
} catch (IOException e) {  
}
```

Ezután színenként "szétbontjuk a képet" és minden színhez rendelünk egy Ascii karaktert is helyettesítés-ként. Nézzünk meg egy példát: Pl, ha a színkód nagyobb, mint 240(ergó fehér),akkor oda nem tesz semmit, míg ha kisebb, mint 60(fekete), oda egy "@" karakter kerül. Ennek az eredményét pedig egy txt fájlba menti ki számunkra a program.

```
for (int i = 0; i < img.getHeight(); i++) {
    for (int j = 0; j < img.getWidth(); j++) {
        Color pixcol = new Color(img.getRGB(j, i));
        pixval = (((pixcol.getRed() * 0.30) + (pixcol.getBlue() * 0.59) + (pixcol
            .getGreen() * 0.11)));
        print(strChar(pixval));
    }
    try {
        prntwrt.println("");
        prntwrt.flush();
        filewrt.flush();
    } catch (Exception ex) {
    }
}

public String strChar(double g) {
    String str = " ";
    if (g >= 240) {
        str = " ";
    } else if (g >= 210) {
        str = ".";
    } else if (g >= 190) {
        str = "*";
    } else if (g >= 170) {
        str = "+";
    } else if (g >= 120) {
        str = "^";
    } else if (g >= 110) {
        str = "&";
    } else if (g >= 80) {
        str = "8";
    } else if (g >= 60) {
        str = "#";
    } else {
        str = "@";
    }
    return str;
}
```


18. fejezet

Helló, Lauda!

18.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! <https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/ch01.html#id527287>

```
public class KapuSzkennel {

    public static void main(String[] args) {

        for(int i=0; i<1024; ++i)

            try {

                java.net.Socket socket = new java.net.Socket(args[0], i);

                System.out.println(i + " figyel!");

                socket.close();

            } catch (Exception e) {

                System.out.println(i + " nem figyel!");

            }

    }

}
```

A feladatban használt kódunk arra szolgál, hogy a parancssori argumentumként kapott IP címen megpróbálunk TCP kapcsolatokat létrehozni. A kapcsolatot az alábbi kóddal próbáljuk meg felépíteni: **java.net.Socket socket = new java.net.Socket(args[0], i);**. Ha sikeresen kiépítjük a kapcsolatot akkor kiírja, hogy figyel-e a portot és utána bezárjuk a socketet. Abban az esetben, viszont ha sikertelen a kapcsolat építés, akkor egy

IOException-t kapunk és kiírjuk hogy nem figyeli a portot. Az alábbi képen láthatjuk, hogy melyik portot figyeljük, míg a többinél IOException lép fel.

```
kisp13@LAPTOP-549UEHV9: /mnt/c/Users/kisp1/Desktop/Prog2
kisp13@LAPTOP-549UEHV9:/mnt/c/Users/kisp1/Desktop/Prog2$ java KapuScanner teszt
Error: Could not find or load main class KapuScanner
Caused by: java.lang.ClassNotFoundException: KapuScanner
kisp13@LAPTOP-549UEHV9:/mnt/c/Users/kisp1/Desktop/Prog2$ javac KapuScanner.java
KapuScanner.java:1: error: class KapuSzkennen is public, should be declared in a file named KapuSzkennen.java
public class KapuSzkennen {
      ^
1 error
kisp13@LAPTOP-549UEHV9:/mnt/c/Users/kisp1/Desktop/Prog2$ javac KapuScanner.java
error: file not found: KapuScanner.java
Usage: javac <options> <source files>
use --help for a list of possible options
kisp13@LAPTOP-549UEHV9:/mnt/c/Users/kisp1/Desktop/Prog2$ javac KapuSzkennen.java
kisp13@LAPTOP-549UEHV9:/mnt/c/Users/kisp1/Desktop/Prog2$ java KapuSzkennen
0 nem figyeli
1 nem figyeli
2 nem figyeli
3 nem figyeli
4 nem figyeli
5 nem figyeli
6 nem figyeli
7 nem figyeli
8 nem figyeli
9 nem figyeli
10 nem figyeli
11 nem figyeli
12 nem figyeli
13 nem figyeli
14 nem figyeli
15 nem figyeli
16 nem figyeli
17 nem figyeli
18 nem figyeli
19 nem figyeli
20 nem figyeli
21 nem figyeli
22 nem figyeli
23 nem figyeli
24 nem figyeli
25 nem figyeli
26 nem figyeli
27 nem figyeli
28 nem figyeli
29 nem figyeli
30 nem figyeli
31 nem figyeli
32 nem figyeli
33 nem figyeli
34 nem figyeli
35 nem figyeli
```

18.2. AOP

Szój bele egy átszövő vonatkozást az első védeli programod Java átíratába! (Sztenderd védeli feladat volt korábban.)

A feladatunk az volt, hogy az AspectJ segítségével az LZWBinfa java verziójához/átíratához (védeli programhoz) fűzzünk hozzá valamilyen átszövő vonatkozást. Én azt a megoldást választottam, hogy a binfához hozzáfűzöm a kiírás preorder és postorder változatát. Nézzük is meg ennek a módszernek a megvalósítását.

```
int melyseg = 0;
public pointcut kiir(LZWBinFa.Csomopont elem, java.io. ←
    PrintWriter os)
```

```

: call(public void kiir(LZWBInFa.Csomopont, java.io.PrintWriter ←
    )) && args(elem,os);
after (LZWBInFa.Csomopont elem, java.io.PrintWriter os) : kiir( ←
    elem,os)
{
try {
preOrder(elem,new PrintWriter("pre-order.txt"));
} catch (FileNotFoundException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
melyseg = 0;
try {
postOrder(elem,new PrintWriter("post-order.txt"));
} catch (FileNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}

```

Elsőként a pointcut segítségével megadjuk azt a helyet, ahol a binfában szeretnénk végrehajtani magát a szövést. Jelen esetben ezt a kiir függvény segítségével fogjuk megejteni. Ha ezzel megvagyunk, akkor a call utasítás segítségével megadjuk azt, hogy mi az a metódus amit hozzá akarunk fűzni az eredeti kódhoz. Ezt követően meg kell határoznunk, hogy a pointcut elé vagy után szeretnénk hozzáfűzni a hozzáfűzendő kódot. Erre szolgál majd az after utasítás, mely a kiir függvény után fog lefutni és hozzá fogja fűzni a preorder és postorder kiíratást a binfához. A preOrder metódus egy új txt file-ba fogja kiírni a binfa preorder bejárását. Esetleges hiba esetén egy Exception-t fogunk kapni. A postOrder metódus lényegében ugyan ezt a módszert fogja elvégezni csak postorder módon.

```

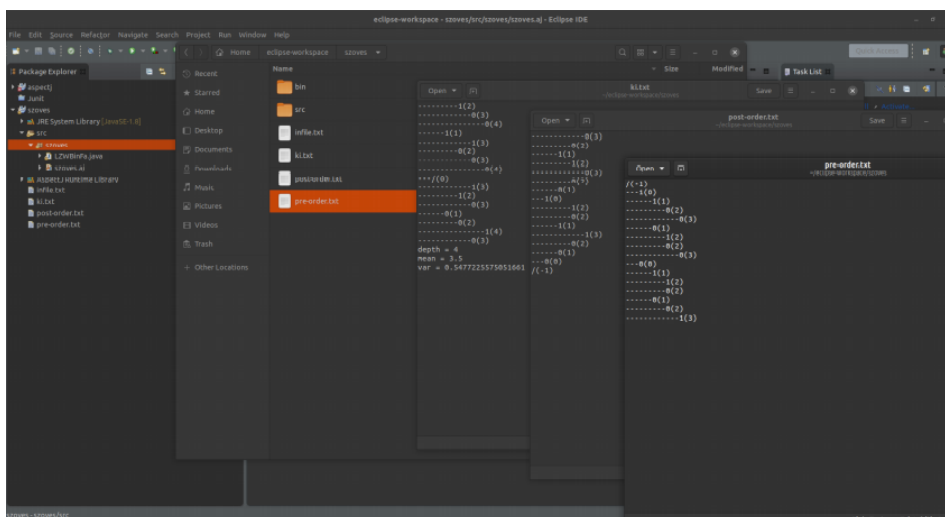
public void preOrder(LZWBInFa.Csomopont elem, java.io. ←
    PrintWriter os){
if (elem != null) {
for (int i = 0; i < melyseg; ++i) {
os.write("---");
}
os.print(elem.getBetu());
os.print("(");
os.print(melyseg - 1);
os.println(")");
++melyseg;
preOrder(elem.egyGyermek(), os);
preOrder(elem.nullasGyermek(), os);
--melyseg;
os.flush();
}
}

public void postOrder(LZWBInFa.Csomopont elem, java.io. ←
    PrintWriter os){

```

```
if (elem != null) {
    ++melyseg;
    postOrder(elem.egyGyermek(), os);
    postOrder(elem.nullasGyermek(), os);
    --melyseg;
    for (int i = 0; i < melyseg; ++i) {
        os.print("---");
    }
    os.print(elem.getBetu());
    os.print("(");
    os.print(melyseg - 1);
    os.println(")");
    os.flush();
}
}
```

Itt látható az a két fabejárás, melyek hozzá lettek fűzve az eredeti binfa kódhunkhoz, ezek segítségével elvégezzük a két féle kiírást.



18.3. Android Játék

Írjunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Helló, Android!” feladatára!

A feladatunk megoldásához egy egyszerűbb akasztófa játékot írtunk. A játék során a klasszikus akasztófa játékot játszhatjuk, annyi különbséggel, hogy a programunk nem rajzolja ki magát az akasztófát hanem számolja a fennmaradó életeket. A feladat megoldásához az Android Studio fejlesztői környezetet használtam, melyben java nyelven lett megírva a kód. Kezdjük is el a kód elemzését. Alapvetően mivel ezt a feladatot még anno amikor a barátaim csinálták a prog2-t velük közösen dolgoztuk ki, így a program jelenleg nem fut le nekem mivel azóta másik gépem van, így a következő órára be tudom fejezni az éppen készülöben lévő Amőbás játékomat.

```
ArrayList<Character> vonalak = new ArrayList<>();
```

```

int Lives = 5;
String amostaniszo;
List<String> Words = new ArrayList<>();
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final Button Game_start = (Button)findViewById(R.id.Game_Start) ←
        ←
;
    final Button Restart = (Button)findViewById(R.id.restart);
    final Button Quit = (Button)findViewById(R.id.quit);
    final EditText char_imput = findViewById(R.id.Char_imput);
    final TextView Word = findViewById(R.id.word);
    final TextView Result = findViewById(R.id.result);
    final TextView Elet = findViewById(R.id.elet);
    Game_start.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Game_start.setVisibility(View.INVISIBLE);
            char_imput.setVisibility(View.VISIBLE);
            Elet.setVisibility(View.VISIBLE);
            Elet.setText("Elet: " + Lives);
            ReadWords(Words);
            amostaniszo = getWord(Words);
            Set_LinesFirs(amostaniszo, Word, vonalak);
        }
    });
}

```

Először fel kell vennünk a játékhoz a szükséges változókat. A vonalak ArrayList-ben kerülnek eltárolásra a játék aktuális menete alapján. Ezt látjuk kiírva a képernyőn, amelyben alpból _ karakterek vannak, de ahogyan a betűk kitalálásra kerülnek, ezeket felülírjuk. A Lives változó értéke határozza meg, hogy mennyi életünk van még, míg az amostaniszo változóban tároljuk az éppen aktuális szót melyet ki kell találnunk. Az utolsó létrehozott változó pedig a Words. Ezen listában töltődik be a szavak listája melyből random, véletlenszerűen kiválasztásra kerül a kitalálendő szó. Ezt követően a képernyőn megjelenő elemek deklarálása történik hogy a későbbiekben lehessen velük dolgozni.

Most pedig lássuk a start gomb funkcióját. A játék indítása gombra kattintva megjelenik az élet számláló, illetve egy mező ahová a tippünket tudjuk beírni, továbbá generálásra kerül a szó melyet ki kell találni. A szó generálásához a getWord metódust alkalmazzuk, amely véletlenszerűen választ egy szót a beolvasott listából.

```

Restart.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Lives = 5;
        Restart.setVisibility(View.INVISIBLE);
        char_imput.setVisibility(View.VISIBLE);
        Elet.setVisibility(View.VISIBLE);
    }
}

```

```
Elet.setText("Elet: " + Lives);
Word.setText(" ");
Result.setVisibility(View.INVISIBLE);
ReadWords(Words);
amostaniszo = getWord(Words);
Set_LinesFirs(amostaniszo, Word, vonalak);
}
});
Quit.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
finish();
System.exit(0);
}
});
```



18.4. Junit teszt

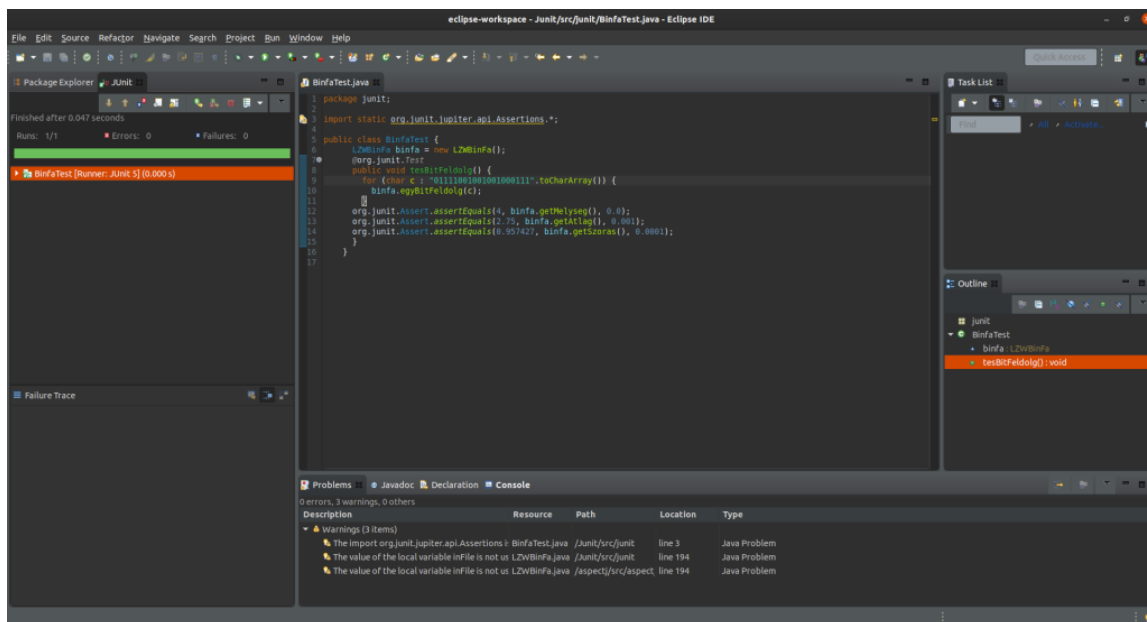
A https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat poszt kézzel számított mélységét és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

A feladatunk szerint, az LZWBInfa java verzióján kell Junit tesztet végrehajtani. A Junit teszt lényege, hogy ellenőrzést hajt végre a megírt kódon, hogy az az elvárásainknak megfelelő módon működik-e, vagy

sem. Ha jól működik a megírt kódunk, akkor a teszt sikeresen lefut, ellenben ha a teszt sikertelen akkor error-t fogunk kapni eredményül. Most pedig nézzük meg a kódunkat.

```
import static org.junit.jupiter.api.Assertions.*;
public class BinfaTest {
    LZWBinFa binfa = new LZWBinFa();
    @org.junit.Test
    public void tesBitFeldolg() {
        for (char c : "01111001001001000111".toCharArray()) {
            binfa.egyBitFeldolg(c);
        }
        org.junit.Assert.assertEquals(4, binfa.getMelyseg(), 0.0);
        org.junit.Assert.assertEquals(2.75, binfa.getAtlag(), 0.001);
        org.junit.Assert.assertEquals(0.957427, binfa.getSzas(), 0.0001);
    }
}
```

A kód létrehoz egy binfa példányt, majd egy for ciklus segítségével végigmegyünk a megadott stringen és feldolgozzuk azt. Ezután az assertEquals metódus elvégzi az ellenőrzést. Az első paraméterként megadjuk hogy milyen eredményt várunk a tesztelni kívánt metódustól, a második paraméter maga a metódus amit tesztelni akarunk, a harmadik pedig egy olyan érték lesz ami azt adja meg hogy mekkora eltérést engedünk meg a teszt során.



18.5. EPAM: Kivételkezelés

Adott az alábbi kódrészlet. Mi történik, ha az input változó 1F, “string” vagy pedig null? Meghívódik-e minden esetben a finally ág? Válaszod indokold!

```
public void test(Object input) {
    try {
        System.out.println("Try!");
        if (input instanceof Float) {
            throw new ChildException();
        } else if (input instanceof String) {
            throw new ParentException();
        } else {
            throw new RuntimeException();
        }
    } catch (ChildException e) {
        System.out.println("Child Exception is caught!");
        if (e instanceof ParentException) {
            throw new ParentException();
        }
    } catch (ParentException e) {
        System.out.println("Parent Exception is caught!");
        System.exit(1);
    } catch (Exception e) {
        System.out.println("Exception is caught!");
    } finally {
        System.out.println("Finally!");
    }
}
```

Amennyiben az input változó értéke null, abban az esetben a egy Runtime Exception-t (futásidejű kivételt) fogunk kapni, mivel ez a parent, azaz szülő osztálya a Child és Parent kivételeknek, így az **instanceof** hamis értéket fog visszaadni. Az előzőekből adódóan a trycatch blokkokon belül csak az utolsó blokk fogja "elkapni."

Második esetként nézzük meg azt az eshetőséget, amennyiben az input változó 1F értékű azaz "string" típusú. Ebben az esetben egy ParentExceptionot fogunk visszakapni, mivel a szülő osztály nem instance-ja a a gyerek osztálynak. Ez ugye fordított esetben igaz lenne, azonban a trycatch-en belül a catch ágak sorrendje miatt jelen esetben nincs létjogosultsága. A trycatch blokkok közül ő a második blokkban kerül "elkapásra."

Végül, de nem utolsó sorban, amikor az input "float" típusú, akkor egy ChildExceptionot fogunk visszakapni, mely már az első catch blokkban elkapásra kerül, mivel a ChildException saját magának egy instance-e lesz.

19. fejezet

Helló, Calvin!

19.1. MNIST

Fealadatleírás

19.2. Deep MNIST

Fealadatleírás

19.3. CIFAR-10

Fealadatleírás, ZÖLD FELADAT

19.4. Android telefonra a TF objektum detektálója

Fealadatleírás

19.5. SMNIST for Machines

Fealadatleírás

19.6. Minecraft MALMO-s példa

Fealadatleírás

20. fejezet

Helló, Berners-Lee! - Olvasónapló

20.1. C++ vs Java összehasonlítása

Mint ahogy a feladatunk címe is sugallja, ebben a fejezetben a C++ és a Java nyelveket fogjuk összehasonlítani Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven, valamint Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II. c. könyvek alapján.

Az összehasonlítási alapul szolgáló két programozási nyelv, egyaránt az objektumorientált programozási nyelvek csoportjába tartozik. Az objektumorientáltságra jellemző, hogy szintaxis szerint a programok osztályokból, valamint az ezekből létrehozott objektumokból állnak. A nyelvekre jellemző kurriózumok közé tartozik továbbá, hogy lehetőségünk van lényegesebben összetettebb kódok programozására, valamint bonyolultabb problémák lemodellezésére is.

A tervezési célokat vizsgálva, a C++ nyelvet rendszerek, valamint alkalmazások programozására fejlesztették ki, a C programozási nyelv kiterjesztésével. A C nyelvhez a C++ támogatja az objektumorientált programozást, a kivételkezelést, az élethosszig tartó erőforrás-kezelést (RAII), az általános programozást, a Template metaprogramming -et (TMP), valamint a C++ szabvány könyvtárát, amely általános tárolókat és algoritmusokat tartalmaz. A Java egy általános célú, statikusan tipizált (gévelt), egyidejű, osztály alapú, objektumorientált programozási nyelv, melyet a végrehajtási függőségek minimalizálására terveztek. Maga a nyelv virtuális gépre támaszkodik, mely megfelelő biztonságot és könnyed "hordozhatóságot" rejt magában. A nyelvi csomag egy kiterjedt könyvtárrendszerrel van ellátva, melynek célja az alapjául szolgáló platform teljes absztrakciója. Ugyan a szintaxisa kísértetiesen hasonló a C++ nyelvéhez, azonban azzal nem kompatibilis. (pl: míg a C++ nyelv esetében lehetséges olyan futtatható programot írunk melyben nem kell osztályt definiálnunk, addig ez a java nyelv esetén nem lehetséges (mivel a Java -ban a legkisebb önálló egységek az osztályok)). Itt egy egyszerű példa erre a hasonlóságra:

```
//C++
class osztaly
{
    public:
        static doStuff();
};
osztaly::doStuff();
```

```
//Java
class osztaly
{
    public static doStuff()
    {}
}
osztaly.doStuff();
```

Az osztályainknak létre tudunk hozni különböző példányokat, melyek minden esetben az adott osztály tulajdonságával rendelkeznek. Ilyen eset lehet pl: amennyiben létrehozunk egy autó osztályt, melyben képesek leszünk különböző tulajdonságok tárolására (autó márkája, színe, stb.) Ezeket az adatokat általában az osztályban létrehozott változóknak tároljuk. A változókat, osztályokon kívülről legtöbbször függvények segítségével tudjuk definiálni, (illetve módosítani).

A programozók többsége az osztályok változóit többnyire védetté szokták tenni, erre azért van szükség, hogy ne tudjunk helytelen paramétert (értéket) megadni. A fent említett megoldást a `private` kulcsszó használatával lehet kivitelezni, mely a tag hozzáférését privátként deklarálja (vagyis a tag csak az adott osztályon belül látható, semmilyen más osztályból (beleértve az alosztályokat is) viszont nem.) A privát tagok láthatósága ez esetben kiterjed a beágyazott osztályokra is. Létezik azonban egy másik kulcsszó is, melyet `public`-nak nevezünk. Itt a tag az adott osztályon kívülről is elérhető, így általában itt szokás deklarálni a függvényeinket.

Amennyiben alapesetben nem állítjuk be az osztálybeli változóink elérhetőségét, abban az esetben a default változó érték `private` lesz, azonban ezt a későbbiek folyamán bármikor szabadon módosíthatjuk. C++ nyelv esetében magában az osztály `public` részében hozzuk létre a változót, míg Java nyelv esetében az osztály deklarációjakor a `public` kulcsszó segítségével tudjuk ezt megtenni.

Az osztályokban szereplő változóinkat lehetőségünk van egy tetszőleges kezdőértékkel ellátni, viszont! amennyiben ezt nem tesszük meg, abban az esetben C++ program esetén a kezdőértékünk egy véletlenszerű érték lesz, míg Java esetében 0. Különböző változók létrehozása esetén azonban könnyedén hibüzenetekbe ütközhetünk, ezért, hogy ezt az eshetőséget elkerüljük, mind a két programnyelvünk konstruktorokat használ.

Na de mik is azok a konstruktorok? Konstruktornak hívjuk az objektumorientált programozási nyelvekben egy osztály azon metódusát, amely az objektum példányosításakor hívódik meg. A konstruktor felelősségi területe általában az, hogy az objektum adatait egy osztály szempontjából értelmezhető, érvényes állapotba hozza (egyes programozási nyelvekben az objektum által használt memóriaterület lefoglalásáért is felelős.) Amennyiben programozáskor nem hozunk létre olyan konstruktort, mely a kezdeti értékeket kezeli, abban az esetben a programnyelvbe alapértelmezettként beépített konstruktor fog lefutni.

A két nyelv közös alkotóelemei közé tartoznak továbbá a metódusok is De hogy mik is azok a metódusok? A metódus, (vagy más néven tagfüggvény), egy olyan eljárás, vagy függvény, amelyet az objektumon belül deklarálunk, mivel szorosan kapcsolódik magához az objektumunkhoz. A metódus az objektumorientált programozás egyik legfontosabb alkotóeleme. Tulajdonképpen arról van szó, hogy az adatmezőkhöz hasonlóan függvény vagy eljárás típusú mezőket is megadhatunk egy objektumtípus definiálásakor. Az ilyen jellegű objektum mezőket összefoglaló néven metódusoknak nevezzük.

A metódusoknak két fajtája létezik: az egyik az eljárás, a másik pedig a függvény. Az eljárásokat lehetőségünk van speciális függvényként értelmezni, mivel nagyon hasonlítanak egymásra. A különbség mindössze-

sen abban irányul meg, hogy az eljárásoknak nincsen visszatérítési értékük, míg a függvényeknek van. Az eljárások bevezetésére a programozásban ismert void kulcsszót fogjuk használni. Függvények használata esetében a megadott kulcsszó az a típus lesz amivel a végén visszatér a függvényünk.

A metódusok létrehozásakor lehetőségünk van különböző paraméterek megadására, melyeket meghíváskor kapnak meg, ezekkel az értékekkel fognak metódusaink dolgozni. Léteznek azonban olyan eshetőségek is, amikor a metódusunk létrehozása esetén egy alapértelmezett (default) értékkel definiáljuk a paramétereket. Ezen megoldásra a C++ egyszerű lehetőséget nyújt, azonban Java esetén egy picit eltérő megvalósítást kell alkalmaznunk, ezt az alábbi példakódok segítségével tudjuk a legjobban szemléltetni.

```
//C++ változat
int darab (int a = 1, int b = 1)
{ }
```

```
//Java verzió
int darab (int a, int b)
{
    a = 1;
    b = 1;
}
```

Térjünk át a memória kezelésben rejlő különbségek bemutatására, mivel ezen terület nagyban eltér a két nyelv vonatkozásában. Míg a Java nyelv alapértelmezetten tartalmaz beépített memória kezelést, azonban C++ nyelv esetében, ez lényegeseb más módon valósul meg. Amennyiben a memória terhelését/kihasználtságát vizsgáljuk, fontos megjegyezni, hogy mekkora szabad kapacitása van még? illetve amire már nincs szükségünk benne, azt lehetőség szerint töröljük. Ez azért fontos, hiszen amennyiben a felesleges adatokat nem töröljük ki, abban az esetben előbb vagy utóbb el fog fogyni memóriánk kapacitása, ennek következtében a programunk már nem lesz képes a jól megszokott, megfelelő módon tovább működni. Java nyelvben a memória kezelésre léteznek u.n: beépített metódusok, melyek automatikusan meghívásra kerülnek, amennyiben már szükségtelen adatok találhatóak benne. Azonban, lehetőségünk van arra is, hogy kérést küldjünk a JVM számára, hogy a Garbage Collectort futtassuk. C++ esetén ez egészen más-képp zajlik, itt nincs lehetőségünk beépített Garbage Collector-t használni. C++ nyelven, amennyiben a memóriából törölni szeretnénk a, már használaton kívüli adatokat, akkor ezt saját magunknak kell megírnunk. Ebből az következik, hogy mindig az aktuális helyzethez legmegfelelőbb módon lehet megírni magát a memóriakezelés folyamatát, így optimálisabb megoldás lehet mint a java beépített verziója.

```
public class Test
{
    public static void main(String[] args) throws ←←
        InterruptedException
    {
        Test t1 = new Test();
        Test t2 = new Test();
        // Nullifying the reference variable
```

```
        t1 = null;
        // requesting JVM for running Garbage Collector
        System.gc();
        // Nullifying the reference variable
        t2 = null;
        // requesting JVM for running Garbage Collector
        Runtime.getRuntime().gc();
    }
}
```

Nézzük meg mi is az a Garbage Collector? A számítógép-programozásban a szemétygyűjtés (garbage collection) a memóriakezelés egyik biztonságos formája. A Garbage Collector (GC) megkísérli eltávolítani a memóriából azokat az objektumokat, amelyeket a programunk már nem használ. A módszert gyakran állítják szembe a hagyományos, manuális memóriakezeléssel, ahol a programozó szabja meg az egyes objektumok élettartamát. Alapelve: meghatározni mely objektumok nincsenek már használatban, illetve felszabadítani az általuk elfoglalt memóriát. Garbage collection előnyei: Megszabadítja a programozót a memóriamanageléstől, így ki tudunk küszöbölni néhány gyakori hibát pl: Dangling pointerek: a mutatott memóriahely már felszabadult, de még van rá hivatkozás Többszörös felszabadítás: a már felszabadított memóriát ismét felszabadítja a program. Memóriaszivárgás (memory leak): a már nem használt memória lefoglalva marad, anélkül, hogy felszabadulna. Hátrányai: Lassú, folyamatosan figyelni kell az objektumokat, ez számításigényes feladat. Nem determinisztikus. Minden objektum törlődni fog, de nem tudni, hogy mikor.

A memória kezelés egyik másik fontos eleme: a pointerek? De mik is azok a pointerek? A programozásban mutatónak (pointer) hívjuk azokat a változókat, melyek tartalma egy memória terület címe. Általában memóriaterületek, rekordok, objektumok, más változók tartalmának elérésére és módosítására használjuk őket. Megkülönböztetünk típussal rendelkező és típus nélküli mutatókat. Előbbiek csak a típusuknak megfelelő típusú változók adatterületére mutathatnak.

A C++ -ban igencsak gyakran használunk pointereket, Javában viszont nem léteznek. Javában a pointer helyett hivatkozásokat találhatunk, melyek segítségével dolgozni tudunk. Nincsenek mutatók, melyek változókra vagy függvényekre mutatnának, helyette visszatérési értékeket, interfészeket használunk.

A dinamikus adatszerkezetek is más megvalósításban szerepelnek mint Javában mind pedig C++ -ban. Míg C++ esetében vektorok szerepelnek, addig Java esetében ezen cél megvalósítására az ArrayList-et használhatjuk. Magában a Java nyelvben is találkozhatunk ugyan vektorokkal, azonban gyakorlati szempontból ez a módszer jelentősen elavulttá vált. Míg az ArrayList párhuzamosítható tulajdonsággal rendelkezik, addig a vektorhoz egyszerre csak egy szál képes hozzáférni, így lassabb működésre lesz képes. C++ nyelv esetében nem találkozunk ezzel az ArrayList sorral.

```
// Java megvalósítás
ArrayList<String> animals = new ArrayList<String>();
animals.add("oroszlán");
animals.add("medve");
animals.add("tigris");
animals.add("papagáj");
System.out.println(animals);
```

```
// C++ megvalósítás
vector<string> v;
    v.push_back("Kutya");
    v.push_back("Macska");
    v.push_back("Veréb");
```

A különböző kifejezések kiértékelése is más-más módon zajlik a két nyelvben. C++ esetén az egyes kifejezések részeit változó sorrendben értékeljük ki, ez esetben nincs megszabott sorrend. Java nyelvben ez a sorrend előre meghatározott érték alapján értékelődik ki. (a kiértékelés mindig balról jobbra történik).

A programozás egy másik igencsak fontos része a kivételkezelés. De mi is az a kivételkezelés? A kivételkezelés egy programozási mechanizmus, melynek célja a program futását szándékosan vagy nem szándékos módon megszakító esemény (hiba) vagy utasítás kezelése. Az eseményt magát kivételnek hívjuk. A hagyományos, szekvenciális és struktúrált programozási kereteken túlmutató hibakezelésre, valamint magasabb szintű hibadetektációra, esetleg korrigálásra használható

Előfordulhatnak olyan események, amikor a megírt kódunk valamilyen oknál fogva nem tud megfelelő módon működni. Ilyen eset lehet pl: amikor egy fileból szeretnénk beolvasni valamit, azonban a megadott file nem létezik. Ezen esetben bátran használhatjuk magát a kivételkezelést. Nagy előnye, hogy C++ és Java nyelvben egyaránt használhatjuk, azonban amíg C++ esetén a fordító "lazábban" kezeli a kivételkezelést, addig Java nyelvben nem. Ez azért van, mivel Javában a fileból nincs lehetőségünk közvetlenül olvasni, amennyiben nincs lekezelve az IOException.

```
// Kivétel kezelés Java nyelven
class Division {
    public static void main(String[] args)
    {
        int a = 10, b = 5, c = 5, result;
        try {
            result = a / (b - c);
            System.out.println("result" + result);
        }
        catch (ArithmeticException e) {
            System.out.println("Exception caught:Division by zero") ←←
                ←←
            ;
        }
        finally {
            System.out.println("I am in final block");
        }
    }
}
```

```
// Kivétel kezelés C++ nyelven
int main()
{
    int x = -1;
    // Some code
    cout << "Before try \n";
    try {
        cout << "Inside try \n";
        if (x < 0)
        {
            throw x;
            cout << "After throw (Never executed) \n";
        }
    }
    catch (int x ) {
        cout << "Exception Caught \n";
    }
    cout << "After catch (Will be executed) \n";
    return 0;
}
```

20.2. Python - Élménybeszámoló

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

A Python egy általános célú, magas szintű programozási nyelv, melyet Guido van Rossum holland programozó kezdett el fejleszteni 1989 végén. A nyelv fő tervezési filozófiája szerint az olvashatóságot, valamint a programozói munka megkönnyítését helyezték előtérbe a futási sebességgel szemben (tehát inkább fusson le lassabban, azonban lényegesen egyszerű legyen a programozó számára.) Többek között a funkcionális, az objektumorientált, az imperatív és a procedurális programozási paradigmákat támogatja. A Python úgynevezett interpreteres nyelv, ami pontosan azt jelenti, hogy nincs különválasztva a forrás és a tárgykód, a megírt kódunk azonnal futtatható, amennyiben rendelkezünk előre telepített Python értelmezővel. A nyelv számos előnye között szerepel, hogy sok beépített eljárást tartalmaz, valamint képes összetettebb adatszerkezetekkel is dolgozni. (ilyenek lehetnek pl: a listák valamint a szótárak is.)

Nyelv főbb jellemzői: Számos programozási nyelvvel szemben, hogy Python esetén nincs szükségünk fordításra a program futtatása előtt, mivel ezt a folyamatot elvégzi helyettünk a Python interpretere. A nyelv rendkívül egyszerű szintaktikával rendelkezik, hiszen nincs szükség a kód tagolásánál zárójelezés alkalmazására (mint számos más nyelv esetében), itt a behúzások jelölik a tagolást. Tagoláskor egyetlen kritériumnak kell megfelelnünk, a megírt program során végig egységes módon tagoljunk, azaz, amennyiben táblázattal kezdtük el programunk tagolását, abban az esetben végig azt használjuk. A nyelv további előnyei közé tartozik az is, hogy az utasításainkat nem szükséges lezárni, amennyiben egy új utasítást szeretnénk megadni, egyszerűen csak ütünk egy enter majd a következő utasítást már új sorba írjuk.

A Python nyelv többféle adatszerkezettel valamint különböző változókkal rendelkezik, ezen változók lehetnek akár számok, vagy stringek, valamint itt is létezik a Null értéknek megfelelő változó típus is. A változó

értékei a számokon belül is lehetnek egészek és lebegőpontosak is. A nyelv egyszerű szintaktikáját illetve felépítését jelzi az is, hogy itt nincs szükség megadni külön az egyes változók típusait, mivel ezt maga a nyelv automatikusan megállapítja számunkra, az alapján, hogy mit tárolunk a változóban. Találkozhatunk a már korábban említett összetett adatszerkezetekkel is, ilyenek a szótárak vagy a listák. Előnyeik, pl: hogy egy listán belül többféle adatot is képesek vagyunk tárolni, még akkor is ha azoknak a típusai különbözőek lennének. (lényegében lehetőségünk van egyszerre számok valamint stringek alkalmazására is.) Méretük dinamikus, ami azt jelenti, hogy folyamatosan bővíthetőek. A szótár hasonló a listához, azonban annyi köztük a különbség, hogy a szótáraknak más szintaktika szerint zajlik az indexelésük. A nyelv előnyeik közé tartozik továbbá az is, hogy számos előre beépített függvényt tartalmaz, melyek nagyban megkönnyíthetik a listák használatát, csak arra várnak hogy használjuk őket.

A program nyelvi eszközei lényegében hasonlítanak a többi programozási nyelvben alkalmazottakhoz. Itt is megtalálhatóak az elágazások, a ciklusok (számos fajtája), valamint a címkék is. Az elágazások ugyan olyan elv szerint működnek mint más programozási nyelvekben, a ciklusok pedig merően hasonlóan, azonban nem teljesen azonosan működnek más programozási nyelvekhez hasonlítva. Ami lényeges különbség a Python esetében, hogy itt a többi nyelvtől eltérően itt a for ciklus megadása során megadhatjuk más módon is a for ciklust abban az esetben, ha azt listákra szeretnénk alkalmazni. A címkék alkalmazása a nyelv esetében, a többi nyelvhez hasonlóan működik, tehát meghatározunk egy címkét és goto utasítással, a megadott címkéhez tudunk ugrani, ahonnan szeretnénk hogy a program futása folytatódjon.

A Python nyelvben is használhatunk különböző függvényeket és osztályokat, ezek használata nagyban hasonlít a C++ nyelvben alkalmazottakhoz. Különbségek nyilván akadnak, azonban a működési elvük lényegében ugyanaz. Különbségek közé tartozik az, hogy pl a függvények esetén, akárcsak a változóknál nincs szükség arra, hogy megadjuk milyen típusú lesz az az elem, mely visszatérésre fog kerülni. A függvényeknél lehetőségünk van arra is, hogy meghívásuk esetén a beadott változónak meghívás közben határozzuk meg az értékét. Osztályok: Az osztályok az objektumorientált programozás részét képezik, sajnos a könyv nem tér ki olyan sok dologra ilyen téren, azonban fontos lehet kiemelni, hogy az osztályok képesek más osztályoktól örökölni.

IV. rész

Irodalomjegyzék

20.3. Általános

- [MARX] Marx, György, *Gyorsuló idő*, Typotex , 2005.
- [PICI] Juhász, István, *Magas szintű programozási nyelvek I.*
- [SMNIST] Norbert Bátfai, Dávid Papp, Gergő Bogacsovics, Máté Szabó, Viktor Szilárd Simkó, Márió Bersenszki, Gergely Szabó, Lajos Kovács, Ferencz Kovács, and Erik Szilveszter Varga, *Object file system software experiments about the notion of number in humans and machines*, Cognition, Brain, Behavior. An Interdisciplinary Journal , DOI 10.24193/cbb.2019.23.15 , 2019.

20.4. C

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

20.5. C++

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

20.6. Python

- [BMEPY] Ekler, Péter, Forstner, Bertalan, És Kelényi, Imre, *Bevezetés a mobilprogramozásba - Gyors prototípusfejlesztés Python és Java nyelven*, Bp., Szak Kiadó, 2008.

20.7. Lisp

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

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPROG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

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

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.