

Univerzális programozás

Így neveld a programozód!

Ed. BHAX, DEBRECEN,
2020. március 22, v. 0.0.7

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	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Ács Bátfai, Margaréta	2020. november 3.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2020-03-02	Az Chomsky/ $a^n b^n c^n$ és Caesar/EXOR csokor feladatok kiírásának aktualizálása (a heti előadás és laborgyakorlatok támogatására).	nbatfai

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.6	2020-03-21	A MALMÖ projektes feladatok átvezetése, minden csokor utolsó feladata Minecraft ágensprogramozás ezzel. Mottók aktualizálása. Prog1 feladatok aktualizálása. Javasolt (soft skill) filmek, elméletek, könyvek, előadások be.	nbatfai
0.0.7	2020-03-22	Javítások.	nbatfai

Ajánlás

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

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

„Csak kicsi hatást ért el a videójáték-ellenes kampány. A legtöbb iskolában kétműszakos üzemen 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/csiga folytonos 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-visszaterjesztéssel 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. A BPP algoritmus Java megvalósítása	50
11.2. Java osztályok a Pi-ben	50
11.3. INNEN KEZDŐDNEK A SYLLABUS SZERINTI FELADATOK	50
11.4. Yoda	50
11.5. EPAM: Java Object metódusok	51
11.6. EPAM: Eljárásorientál vs Objektumorientált	52
11.7. EPAM: Objektum példányosítás programozási mintákkal	53
12. Helló, Liskov!	54
12.1. Ciklomatikus komplexitás	54
12.2. EPAM: Interfész evolúció Java-ban	55
12.3. EPAM: Liskov féle helyettesíthetőség elve, öröklődés	55
12.4. EPAM: Interfész, Osztály, Absztrak Osztály	56

13. Helló, Mandelbrot!	58
13.1. Forward engineering UML osztálydiagram	58
13.2. EPAM: Neptun tantárgyfelvétel modellezése UML-ben	60
13.3. EPAM: Neptun tantárgyfelvétel UML diagram implementálás	61
13.4. EPAM: OO modellezés	63
14. Helló, Chomsky!	65
14.1. Encoding	65
14.2. OOCWC lexer	65
14.3. l334d1c4 6	65
14.4. Full screen	65
14.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció	65
14.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció	65
14.7. Perceptron osztály	66
14.8. EPAM: Order of everything	66
14.9. EPAM: Bináris keresés és Buborék rendezés implementálása	67
14.10 EPAM: Saját HashMap implementáció	70
15. Helló, Stroustrup!	79
15.1. Másoló-mozgató szemantika	79
15.2. EPAM: It's gone. Or is it?	80
15.3. EPAM: Kind of equal	82
15.4. EPAM: Java GC	83
16. Helló, Gödel!	85
16.1. Gengszterek	85
16.2. C++11 Custom Allocator	85
16.3. STL map érték szerinti rendezése	85
16.4. Alternatív Tabella rendezése	85
16.5. Prolog családfa	85
16.6. GIMP Scheme hack	85
16.7. EPAM: Mátrix szorzás Stream API-val	86
16.8. EPAM: LinkedList vs ArrayList	86
16.9. EPAM: Refactoring	86

17. Helló, hetedik hét!	87
17.1. FUTURE tevékenység editor	87
17.2. OOCWC Boost ASIO hálózatkézelése	87
17.3. SamuCam	87
17.4. BrainB	87
17.5. OSM térképre rajzolása	87
17.6. EPAM: XML feldolgozás	87
17.7. EPAM: ASCII Art	88
17.8. EPAM: Titkos üzenet, száll a gépben!	88
18. Helló, Lauda!	89
18.1. Port scan	89
18.2. AOP	89
18.3. Android Játék	89
18.4. Junit teszt	89
18.5. OSC1	89
18.6. OSC2	89
18.7. OSC3	90
18.8. EPAM: DI	90
18.9. EPAM: JSON szerializáció	90
18.10 EPAM: Kivételkezelés	90
19. Helló, Calvin!	91
19.1. MNIST	91
19.2. Deep MNIST	91
19.3. CIFAR-10	91
19.4. Android telefonra a TF objektum detektálója	91
19.5. SMNIST for Machines	91
19.6. Minecraft MALMO-s példa	91
20. Helló, Berners-Lee! - Olvasónapló	92
20.1. C++ vs Java	92
20.2. Python	97

IV. Irodalomjegyzék	98
20.3. Általános	99
20.4. C	99
20.5. C++	99
20.6. Python	99
20.7. Lisp	99

DRAFT

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
11.1. A program Yoda conditions nélkül	51
11.2. A program Yoda conditions-al	51
13.1. A program UML terve	58
13.2. A program UML terve	61
14.1. Output	69
15.1. Output	80

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 (hogyan 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.xsl
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](https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf), 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/FktsFcooIG8>, 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 hoztok 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.

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: <https://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 trükközzünk 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 használata 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 használata nélkül írd egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videón.)

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

Megoldás forrása: 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:

$(n+1)!+2, (n+1)!+3, \dots, (n+1)!+n, (n+1)!+(n+1)$ 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 Bاتفai, 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")
```

Soronként 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 prímelek különbségét képi, 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. indexre teljesül.

```
t1primes = 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/t1primes+1/t2primes
```

Az 1/t1primes a t1primes 3,5,11 értékéből az alábbi reciprokokat képi:

```
> 1/t1primes
[1] 0.33333333 0.20000000 0.09090909
```

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

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

Az 1/t1primes + 1/t2primes pedig ezeket a törtet rendre összeadja.

```
> 1/t1primes+1/t2primes
[1] 0.53333333 0.3428571 0.1678322
```

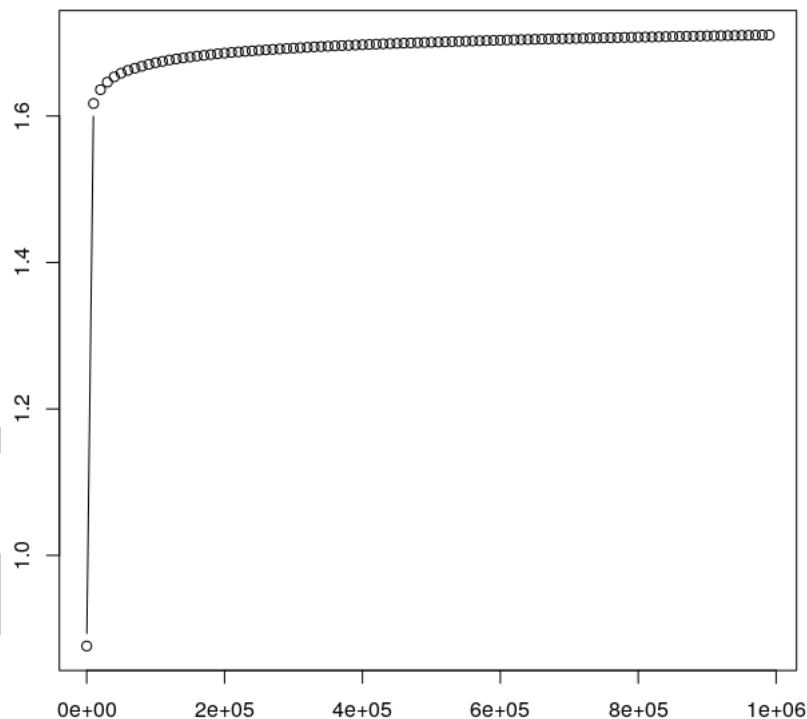
Nincs más dolgunk, mint ezeket a törtet összeadni a `sum` függvénnyel.

```
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!

DRAFT

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

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

**Bugok**

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

i.

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

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?

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

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

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

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

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!

DRAFT

## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\* háromszögmátrix

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

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

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_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");
}

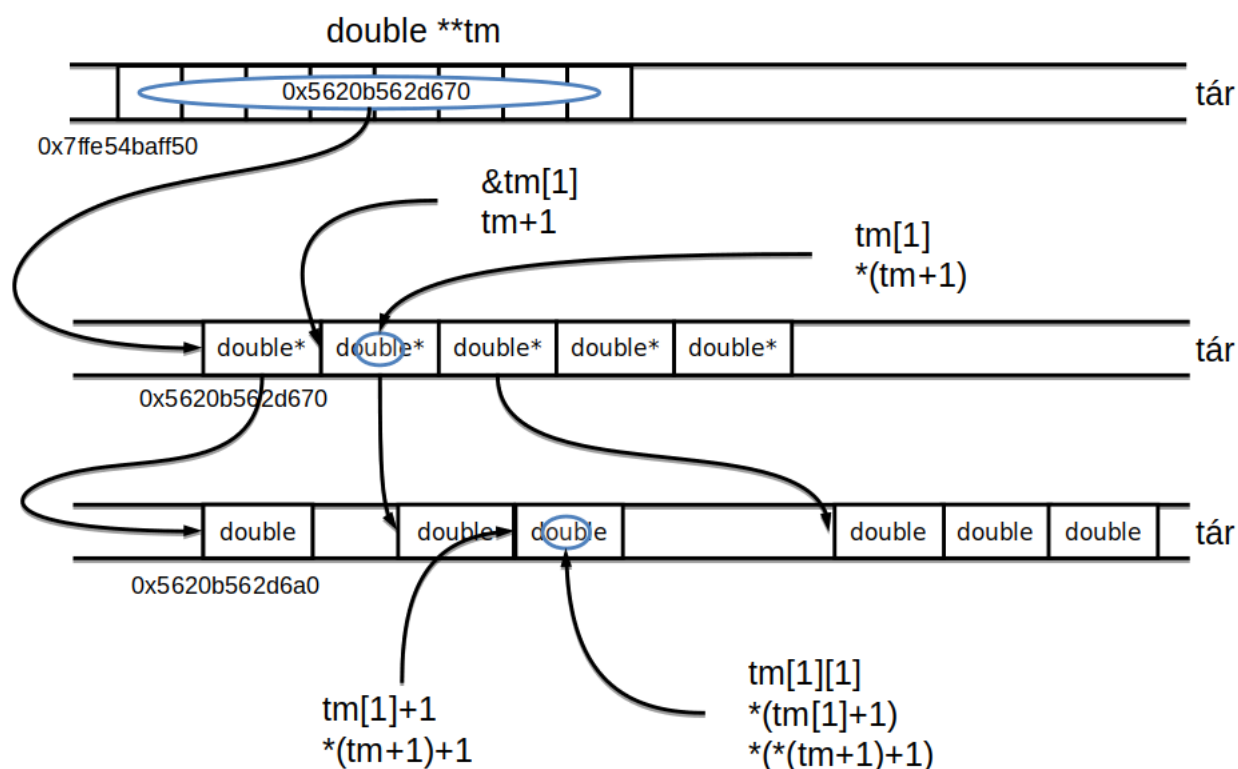
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
((tm + 3) + 3) = 45.0;

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](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](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 többrétegű 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

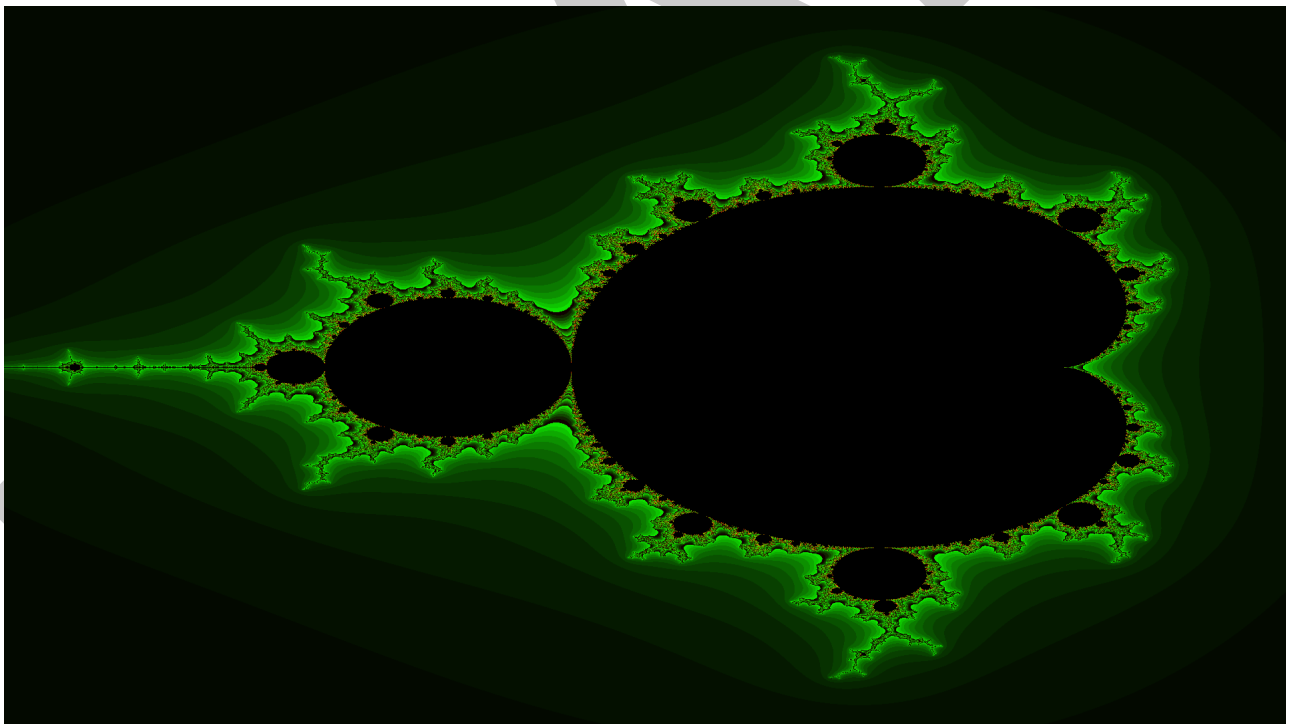
# Helló, Mandelbrot!

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

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

int
main (int argc, char *argv[])
{
 int szelesseg = 1920;
 int magassag = 1080;
 int iteraciosHatar = 255;
 double a = -1.9;
 double b = 0.7;
 double c = -1.3;
 double d = 1.3;

 if (argc == 9)
 {
 szelesseg = atoi (argv[2]);
 magassag = atoi (argv[3]);
 iteraciosHatar = atoi (argv[4]);
 a = atof (argv[5]);
 b = atof (argv[6]);
 c = atof (argv[7]);
 d = atof (argv[8]);
 }
}
```

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

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

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

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

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

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

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

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

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

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

 ++iteracio;
 }

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

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

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

}
```

## 5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbRzY76E>

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/Biomorf](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](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  $c$  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](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: [bhaxor/attention\\_raising/CUDA/mandelpngc\\_60x60\\_100.cu](https://bhaxor.github.io/attention_raising/CUDA/mandelpngc_60x60_100.cu) nevű állománya.

## 5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás videó: Illetve [https://bhaxor.blog.hu/2018/09/02/ismerkedes\\_a\\_mandelbrot\\_halmazzal](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](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 eltéve 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](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](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](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](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](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](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](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!

DRAFT

## 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](https://youtu.be/OKdAkI_c7Sc)

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

Tanulságok, tapasztalatok, magyarázat... 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](https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv)

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

Tanulságok, tapasztalatok, magyarázat... 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!

DRAFT

## 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.

---

DRAFT

# 11. fejezet

## Helló, Arroway!

### 11.1. A BPP algoritmus Java megvalósítása

Ez már a Prog2, de előre dolgozhatsz a nyári szünetben!

Megoldás forrása: <https://regi.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apas03.html>

### 11.2. Java osztályok a Pi-ben

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

### 11.3. INNEN KEZDŐDNEK A SYLLABUS SZERINTI FELADATOK

Ha szükség van feladatléírásra [itt van a syllabus](#)

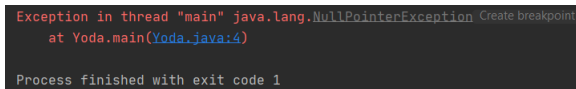
### 11.4. Yoda

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

A Yoda egy olyan programozási stílus, ahol a kifejezés két oldalát felcseréljük. Ez a technika segít elkerülni a NullPointerException-et. Erre mutat példát az alábbi program:

```
public class Yoda {
 public static void main(String[] args) {
 String name = null;
 if (name.equals("Reka")) {}
 }
}
```





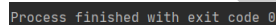
```
Exception in thread "main" java.lang.NullPointerException: Create breakpoint
at Yoda.main(Yoda.java:4)

Process finished with exit code 1
```

11.1. ábra. A program Yoda conditions nélkül

Láthatjuk, hogy itt még nincs használva a Yoda condition, így a program NullPointerException-el hibára fut.

```
public class WorkingYoda {
 public static void main(String[] args) {
 String name = null;
 if ("Reka".equals(name)) {}
 }
}
```



```
Process finished with exit code 0
```

11.2. ábra. A program Yoda conditions-al

Itt már látszik, hogy csak fel kellett cseréljük az equals két oldalát és máris működik a program.

## 11.5. 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 öröklési fa legfelső szintjén az Object(java.lang.Object) osztály található, mely osztály az alábbi metódusokkal rendelkezik: protected Object clone(), boolean equals(Object obj), void finalize(), getClass(), int hashCode(), void notify(), void notifyAll(), String toString(), void wait()

A getClass metódussal az objektum futási osztályát kapjuk vissza.

Az equals metódussal két sztringet hasonlít össze, és megmondja, hogy egyenlőek-e vagy sem.

A clone() egy objektum pontos másolatát készíti el. Létrehoz egy új példányt az aktuális objektum osztályából, és inicializálja az összes mezőjét pontosan az objektum megfelelő mezőinek tartalmával.

A finalizer-t akkor hívják meg, amikor a JVM rájön, hogy ezt a különleges példányt szemétnak kell gyűjteni. Egy ilyen véglegesítő bármilyen műveletet végezhet, beleértve az objektum életre keltését is. A finalize fő célja azonban az objektumok által használt erőforrások felszabadítása, mielőtt eltávolítanák őket a memóriából.

A hashCode() egy egész értéket ad vissza, amelyet egy hash algoritmus generál

Amikor wait() metódust hívunk, arra kényszerítjük az aktuális szálat, hogy megvárja, amíg más szál ugyanarra az objektumra hívja a notify() vagy a notifyAll() metódust.

A `toString()` metódus az objektum karakterlánc-reprezentációját adja vissza. Ha bármilyen objektumot akarunk kiírni, a java fordító belsőleg meghívja a `toString()` metódust az objektumon.

Példaprogram: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraStudent.java](#)

## 11.6. EPAM: Eljárásorientál vs Objektumorientált

Írj egy 1 oldalas értekező esszé szöveget, amiben összehasonlítod az eljárásorientált és az objektumorientált paradigmát, igyekezve kiemelni az objektumorientált paradigma előnyeit!

Eljárás-orientált programozás (POP) és Objektum-orientált programozás (OOP) mindkettő a programozási megközelítés, amely magas szintű nyelvet használ a programozáshoz.

Az OOP programozás arra épül, hogy programunkat különböző modulokra tudjuk felosztani, melyet későbbiekben sokkal egyszerűbben bővíthetünk vagy módosíthatunk, emellett kódunk sokkal átláthatóbb lesz.

A POP programozás során a fő hangsúly az elvégzendő feladat sorrendjére irányul. Ha a program kiterjedt, akkor néhány kisebb függvény segítségével feloszthatjuk. Ezekben a függvényekben megosztják a globális adatokat, ezért felmerül az adatbiztonság kérdése is.

POP a folyamatábrát követi a feladat elvégzéséhez. Az OOP fő hangsúlya adatbiztonság mivel csak egy osztály objektumai férhetnek hozzá az osztály attribútumaihoz vagy funkciójához. A funkciók a nagy programok vagy alprogramok kis egységei, amelyek végrehajtják a fő feladat elvégzését. Ezzel szemben az osztály OOP attribútumai és funkciói meg vannak osztva az osztályok között tárgyak. A POP-ban nincs specifikus hozzáférési mód az attribútumok vagy funkciók elérésére a programban. Ezzel szemben az OOP-ban három `public`, `private`, `protected` hozzáférési mód létezik, amelyeket hozzáférési módszerként használnak attribútumok vagy funkciók eléréséhez. A POP nem támogatja a túlterhelés / polimorfizmus fogalmát. Az OOP igen. A POP-ban nincs fogalom az öröklésről, míg az OOP támogatja azt. A POP kevésbé biztonságos az OOP-hoz képest, mivel az OOP-ban a hozzáférési specifikátor korlátozza a hozzáférést olyan attribútumokhoz vagy funkciókhoz, amelyek növelik a biztonságot. Ha a POP-ban valamilyen adatot meg kell osztani a program összes funkciója között, akkor azt globálisan minden funkción kívül deklarálják. Míg az OOP-ban az osztály adattagjai az osztály tagfunkcióin keresztül érhetők el. A POP-ban nincs a barát funkció fogalma. Ezzel szemben az OOP-ban van egy olyan barát funkció fogalma, amely nem tagja az osztálynak, de mivel barát tag, hozzáférhet az osztály adat tagjának és tag funkcióinak eléréséhez. A POP-ban nincs fogalom a virtuális osztályokról, míg az OOP-ban a virtuális függvények támogatják a polimorfizmust.

A POP előnye, hogy lehetővé teszi, hogy ugyanazt a kódot több helyen is újra felhasználhassuk, megkönnyíti a programáramlás követését, képes modulok építésére. Hátránya azonban, hogy a globális adatok sebezhetők, az adatok a programon belül szabadon mozoghatnak, nehéz az adatok helyzetét ellenőrizni, a funkciók cselekvésorientáltak, a funkciók nem képesek kapcsolódni a probléma elemeihez, a valós problémák nem modellezhetők, a kód részei egymástól függenek, az egyik alkalmazáskód nem használható másik alkalmazásban, az adatok továbbítása a függvények használatával történik.

Az OOP előnye, hogy az objektumok segítenek a feladat particionálásában a projektben, a biztonságos programokat adatrejtés segítségével lehet felépíteni, potenciálisan leképezheti az objektumokat, lehetővé teszi az objektumok különböző osztályokba sorolását. Emellett az objektum-orientált rendszerek könnyedén frissíthetők, a redundáns kódok öröklés útján kiküszöbölhetők, a kódok az újrafelhasználhatósággal kibővíthetők, nagyobb modulitás érhető el, az adatok absztrakciója növeli a megbízhatóságot, rugalmas a

dinamikus kötési koncepció miatt, az információ elrejtésével elválasztja az alapvető specifikációt a megvalósításától. Hátránya azonban, hogy több erőforrást igényel, az objektumok dinamikus viselkedése RAM tárolást igényel, az észlelés és a hibakeresés nehezebb az összetett alkalmazásokban, amikor az átadást végrehajtják, az öröklés osztályait szorosan összekapcsolja, ami befolyásolja a tárgyak újrahasznosíthatóságát.

Az OOP kijavítja a POP hibáit az „objektum” és az „osztályok” fogalmának bevezetésével. Fokozza az adatbiztonságot, valamint az objektumok automatikus inicializálását és tisztítását. Az OOP lehetővé teszi az objektum több példányának létrehozását interferencia nélkül.

## 11.7. EPAM: Objektum példányosítás programozási mintákkal

Hozz példát mindegyik “creational design pattern”-re és mutasd be mikor érdemes használni őket!

A Factory Method egy olyan tervezési minta, amely interfészt biztosít objektumok létrehozásához egy szuperosztályban, de lehetővé teszi az alosztályok számára a létrehozandó objektumok típusának megváltoztatását.

Példaprogram: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraFactory2/src/dolgok/Shop.java](#)

Az Abstract Factory olyan tervezési minta, amely lehetővé teszi kapcsolódó objektumok családjainak előállítását anélkül, hogy megadná a konkrét osztályokat.

Példaprogram: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraAbstractFactory/src/order/Make.java](#)

A Builder egy tervezési minta, amely lehetővé teszi összetett objektumok lépésről lépésre történő elkészítését.

Példaprogram: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraBuilder/src/app/App.java](#)

A Prototype egy olyan tervezési minta, amely lehetővé teszi a meglévő objektumok másolását anélkül, hogy a kódot az osztályuktól függené.

Példaprogram: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraPrototype/src/shapes/Main.java](#)

A Singleton olyan tervezési minta, amely lehetővé teszi, hogy egy osztálynak csak egy példánya legyen, miközben globális hozzáférési pontot biztosít ehhez a példányhoz.

Példaprogram: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraSingleton/singleton/Singleton.java](#)

## 12. fejezet

# Helló, Liskov!

### 12.1. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a [https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_2.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf) (77-79 fóliát)!

A ciklomatikus komplexitás egy szoftvermetrika, melyet Thomas J. McCabe publikált 1976-ban. Alkotója után gyakran McCabe-komplexitásnak is nevezik. A metrika egy adott szoftver forráskódjának alapján határozza meg annak komplexitását, egy konkrét számértékben kifejezve. A komplexitás számítása a gráfelméletre alapul. A forráskódban az elágazásokból felépülő gráf pontjai, és a köztük lévő élek alapján számítható ki.

A ciklomatikus komplexitás értéke:  $M = E - N + 2P$  ahol  $E$  a gráf éleinek száma,  $N$  a gráfban lévő csúcsok száma,  $P$  az összefüggő komponensek száma.

Példaprogram:

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

 int points = 91;

 if (points >= 40) {
 if (points <= 60) {
 System.out.println("2");
 }
 else if (points <= 70) {
 System.out.println("3");
 }
 else if (points <= 80) {
 System.out.println("4");
 }
 else {
 System.out.println("5");
 }
 }
 else {
```

```
 System.out.println("1");
 }
}
} //complexity 5
```

[../../bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraskodok/Exam.java](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/forraskodok/Exam.java)

## 12.2. EPAM: Interfész evolúció Java-ban

Mutasd be milyen változások történtek Java 7 és Java 8 között az interfészekben. Miért volt erre szükség, milyen problémát vezetett ez be?

A Java 8 óta az interfészekben használhatunk statikus metódusokat is. Ezek működése eltér a Java 8 előtti interfészekétől, hiszen eredetileg az interfészek metódusai mind absztrakt metódusok voltak, tehát az implementációt nem kellett megadnunk. Az implementációk megadása az interfészt használó osztály feladata volt.

A statikus metódusoknak az interfészekben való használatakor meg kell adni a body-ját. Ahhoz, hogy egy ilyen metódust létrehozzunk, a `static` kulcsszót kell használnunk. Ezek a metódusok alapból publikusak lesznek.

Az interfészek statikus metódusai nem öröklődnek, tehát ha ilyen metódust akarunk meghívni, akkor azt közvetlenül az interfészben kell megtenni és nem az azt implementáló osztályban.

```
public interface MyInterface {

 //Működik, hiszen statikus metódusnak van body-ja
 static String name(){
 String name = "Reka";
 return name;
 }

 //nem működik, mivel nincs body
 static String anotherName();
}
```

Megoldás linkje: [../../bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraskodok/MyInterface.java](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/forraskodok/MyInterface.java)

## 12.3. EPAM: Liskov féle helyettesíthetőség elve, öröklődés

Adott az alábbi osztály hierarchia.

```
class Vehicle, class Car extends Vehicle, class Supercar extends Car
```

Mindegyik osztály konstruktorában történik egy kiíratás, valamint a `Vehicle` osztályban szereplő `start()` metódus mindegyik alosztályban felül van definiálva.

Mi történik ezen kódok futtatása esetén, és miért?

A Liskov-féle behelyettesítési elv, rövid nevén LSP, kimondja, hogy a program viselkedése nem változhat meg attól, hogy az ős osztály egy példánya helyett a jövőben valamelyik gyermek osztályának példányát használom.

```
Vehicle firstVehicle = new Supercar();
firstVehicle.start();
System.out.println(firstVehicle instanceof Car);
Car secondVehicle = (Car) firstVehicle;
secondVehicle.start();
System.out.println(secondVehicle instanceof Supercar);
Supercar thirdVehicle = new Vehicle();
thirdVehicle.start();
```

A firstVehicle egy Supercar lesz, tehát a Supercar class-on belüli start() metódust fogja meghívni a program. Mivel a Supercar osztály kiegészíti a Car osztályt, így a firstVehicle instanceof Car feltétel igaz lesz.

A secondVehicle egyenlő a firstVehicle-el, így szintén a Supercar classban lévő start() metódus kerül meghívásra.

A thirdVehicle esetén a program hibát jelez.

Forráskód: [../..../bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraskodok/liskov/Vehicle.java](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/forraskodok/liskov/Vehicle.java)

## 12.4. EPAM: Interfész, Osztály, Absztrak Osztály

Mi a különbség Java-ban a Class, Abstract Class és az Interface között? Egy tetszőleges példával / példa kódon keresztül mutasd be őket és hogy mikor melyik koncepciót célszerű használni.

Hasznos linkek: <https://www.javatpoint.com/difference-between-abstract-class-and-interface>, <https://dzone.com/articles/when-to-use-abstract-class-and-intreface>

Abstract class:

Az absztrakt osztálynak lehetnek absztrakt és nem absztrakt metódusai. Nem támogatja a többszörös öröklődést. Lehetnek végleges, nem végleges, statikus és nem statikus változók. Az absztrakt osztály biztosíthatja az interfész megvalósítását. Az abstract kulcsszóval deklaráljuk. Az absztrakt osztály kiterjeszthet egy másik Java osztályt és több Java interfészt is megvalósíthat. Kiterjeszthető a "extends" kulcsszóval. Az absztrakt osztályoknak lehetnek private, protected, stb. tagjai. Akkor használjuk, ha meg szeretnénk osztani a kódot több szorosan kapcsolódó osztály között, az absztrakt osztályt kibővítő osztályoknak sok közös módszere vagy mezője van, vagy a nyilvánosaktól eltérő hozzáférés-módosítókat igényelnek, vagy ha nem statikus, vagy nem final mezőket szeretnénk deklarálni.

Interface:

A Java 8 óta rendelkezhet alapértelmezett és statikus metódusokkal is. Támogatja a többszörös öröklődést. Csak statikus vagy final változók vannak. Az interfész nem tudja biztosítani az absztrakt osztály megvalósítását. Az interface kulcsszóval deklaráljuk. Egy interfész csak egy másik Java interfészt bővíthet. Az „implement” kulcsszóval implementálható. Tagjai alapból publikusak. Akkor használjuk, ha nem kapcsolódó osztályok implementálják az interfészünket, meg szeretnénk adni egy adott adattípus viselkedését, de nem érdekel minket, hogy ki hajtja végre a viselkedését vagy ki szeretnénk használni a többszörös öröklődés lehetőségét.

Class:

Nem lehetnek absztrakt metódusai. A new kulcsszóval példányosítható. Lehet final-el deklarálni.

Példakód: [../../bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/forraskodok/ClassInterfaceAbstractClass/src/Class.java](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/forraskodok/ClassInterfaceAbstractClass/src/Class.java)

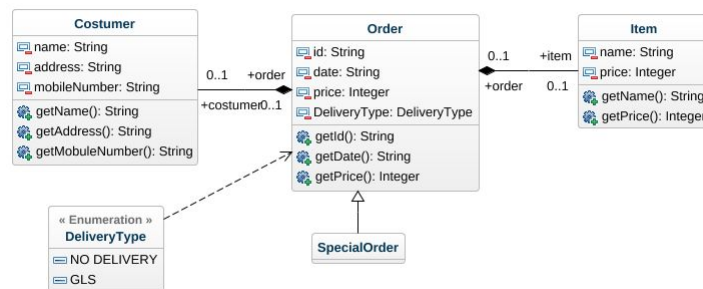
DRAFT

## 13. fejezet

# Helló, Mandelbrot!

### 13.1. Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generáljunk belőle forrást!



13.1. ábra. A program UML terve

```
package order;

public class Costumer {
 private String name;
 private String address;
 private String mobileNumber;

 public String getName() {
 return name;
 }

 public String getAddress() {
 return address;
 }
}
```



```
public String getMobileNumber() {
 return mobileNumber;
}
}
```

A Costumer osztályban az UML terv szerint megadjuk a vásárló nevét, címét és telefonszámát, majd megadjuk az ezekhez tartozó gettereket.

```
package order;

public class Item {
 private String name;
 private int price;

 public String getName() {
 return name;
 }

 public int getPrice() {
 return price;
 }
}
```

Az Item osztályban a termék nevét, árát és a hozzájuk tartozó gettereket láthatjuk.

```
package order;

public enum DeliveryType {
 NO_DELIVERY, GLS
}
```

A DeliveryType enumban megadjuk, hogy milyen szállítási típusok vannak.

```
package order;

public class Order {
 private String id;
 private String date;
 private int price;
 private DeliveryType deliveryType;

 public String getId() {
 return id;
 }

 public String getDate() {
 return date;
 }
}
```

```
public int getPrice() {
 return price;
}

public DeliveryType getDeliveryType() {
 return deliveryType;
}

}
```

Az Order osztályban a rendeléshez szükséges adatokat és azok gettereit adtam meg.

```
package order;

public class SpecialOrder extends Order{

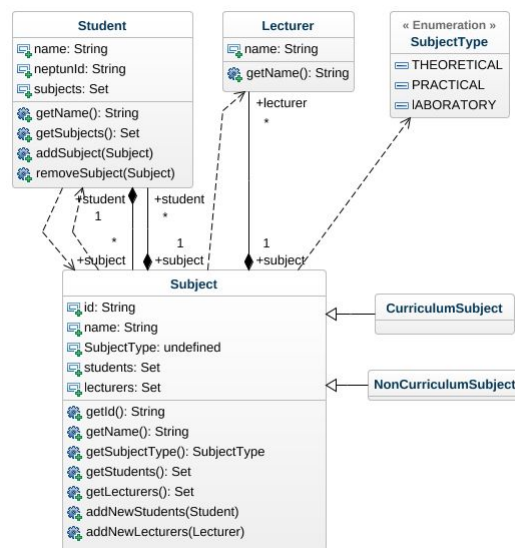
}
```

A SpecialOrder egy kiegészítés az Order osztályhoz.

## 13.2. EPAM: Neptun tantárgyfelvétel modellezése UML-ben

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

Az UML (Unified Modeling Language) szabványos, általános célú modellező nyelv, üzleti elemzők, rendszertervezők, szoftvermérnökök számára. Grady Booch, Ivar Jacobson és James Rumbaugh egyesített munkájának terméke. Az objektum orientált (OO) modellezés módszerét alkalmazni lehet a való világ bonyolultságának leírására. Az UML egy gyakorlati, objektum orientált modellező megoldás, nagy méretű programrendszerek modelljeinek vizuális dokumentálására alkalmas eszköz. Az UML módszer és leíró nyelv segítségével különböző nézőpontú szöveges és grafikus modellek készíthetők többek közt rendszerekről, szervezetekről, szereplőkről, üzleti tevékenységekről és folyamatokról, logikai összetevőkről, szoftverekről, programokról, valamint adatbázisokról.



13.2. ábra. A program UML terve

### 13.3. EPAM: Neptun tantárgyfelvétel UML diagram implementálás

Implementáld le az előző feladatban létrehozott diagrammot egy tetszőleges nyelven.

```

package student;

import java.util.Set;
import subject.Subject;

public class Student {
 private String name;
 private String neptunId;
 private Set<Subject> subjects;

 public String getName() {
 return name;
 }

 public void setName(String name) {
 this.name = name;
 }

 public Set<Subject> getSubjects() {
 return subjects;
 }

 public String getNeptunId() {
 return neptunId;
 }
}

```

```
}

public void addSubject(Subject subject){
 this.subjects.add(subject);
}

public void removeSubject(Subject subject){
 this.subjects.remove(subject);
}

}
```

A Student classban megadjuk a tanuló nevét, Neptun kódját, valamint egy halmazban eltároljuk a felvett tárgyait. Ezekre beállítjuk a gettereket. Létrehozunk egy addSubject metódust, amivel tárgyakat tudunk hozzáadni a halmazunkhoz, valamint egy removeSubject metódust, amivel törölni tudunk.

```
package lecturer;

public class Lecturer {
 private String name;

 public String getName() {
 return name;
 }
}
```

A Lecturer osztályban az oktató adatait adjuk meg.

```
package subject;

import java.util.Set;
import student.Student;
import lecturer.Lecturer;

public class Subject {
 private String id;
 private String name;
 private SubjectType subjectType;
 private Set<Student> students;
 private Set<Lecturer> lecturers;

 public String getId() {
 return id;
 }

 public String getName() {
 return name;
 }

 public SubjectType getSubjectType() {
 return subjectType;
 }
}
```

```
}

public Set<Student> getStudents() {
 return students;
}

public Set<Lecturer> getLecturers() {
 return lecturers;
}

public void addNewStudents(Student student) {
 this.students.add(student);
}

public void addNewLecturers(Lecturer lecturer) {
 this.lecturers.add(lecturer);
}
}
```

A Subject osztályban a tárgyra vonatkozó adatokat adjuk meg. Létrehozunk majd egy enumot a tárgy típusaira, valamint lesz egy Student és egy Lecturer halmazunk. Itt is ugyanúgy megadjuk a gettereket, valamint két metódust, az egyikkel hallgatókat, a másikkal pedig oktatókat adhatunk a halmazunkhoz.

Ezek mellett két alosztályt is létrehozunk, az egyik a CurriculumSubject, a másik pedig a NonCurriculumSubject. Mindkettő a Subject osztályt egészíti ki.

## 13.4. EPAM: OO modellezés

Írj egy 1 oldalas esszét arról, hogy OO modellezés során milyen elveket tudsz követni (pl.: SOLID, KISS, DRY, YAGNI).

Az objektum-orientált számítógépes programozásban a SOLID öt tervezési elv rövidítése, amelynek célja a szoftvertervek érthetőbbé, rugalmasabbá és fenntarthatóbbá tétele. Ez nem kapcsolódik a GRASP szoftvertervezési elvekhez. A SOLID elvek elméletét Robert C. Martin vezette be 2000-ben a Design Principles and Design Patterns című cikkében, bár a SOLID rövidítést később Michael Feathers vezette be. Konceptiója: Single-responsibility principle: Egy osztálynak csak egyetlen felelőssége kell, hogy legyen, vagyis a szoftver specifikációjának csak egy részén végzett változtatások befolyásolhatják az osztály specifikációját. Open-closed principle: "A szoftveralanyoknak ... nyitva kell lenniük a kiterjesztésre, de zárva a módosításra." Liskov substitution principle: A programban lévő objektumoknak az altípusok példányaival cserélhetőnek kell lenniük anélkül, hogy a program helyességét megváltoztatnák. Interface segregation principle: Sok ügyfélspecifikus interfész jobb, mint egy általános célú interfész. Dependency inversion principle: Absztrakcióktól kell függni, nem konkretizációktól.

A KISS, a keep it simple, stupid rövidítése, olyan tervezési elv, amelyet az amerikai haditengerészet 1960-ban jegyzett fel. A KISS elv szerint a legtöbb rendszer akkor működik a legjobban, ha egyszerűbbé teszik, nem pedig bonyolulttá teszik őket, ezért a tervezésnél az egyszerűségnek kell kulcsfontosságúnak lennie, és kerülni kell a felesleges bonyolultságot.

A DRY, azaz Don't repeat yourself a szoftverfejlesztés alapelve, amelynek célja a szoftverminták megismétlődésének csökkentése, absztrakciókkal való helyettesítés vagy adatok normalizálása a redundancia elkerülése érdekében. A DRY elv a következőképpen van megfogalmazva: "Minden tudásnak egyetlen, egyértelmű és hiteles képviselővel kell rendelkeznie a rendszeren belül". Az elvet Andy Hunt és Dave Thomas fogalmazták meg The Pragmatic Programmer című könyvükben. Az elvet alkalmazzák még adatbázis-sémákon, teszterveken, a build rendszeren, sőt dokumentáción is. A DRY elv sikeres alkalmazásakor a rendszer bármely elemének módosítása nem igényel más logikailag nem kapcsolódó elemek megváltoztatását.

A YAGNI a "You aren't gonna need it" rövidítése, amely az extrém programozás (XP) alapelve, amely szerint a programozónak csak addig kell hozzáadnia a funkciókat, amíg szükségesnek nem találják. Az XP társalapítója, Ron Jeffries azt írta: "Mindig akkor hajtsa végre a dolgokat, amikor valóban szüksége van rájuk, soha ne, amikor csak azt látja, hogy szüksége van rájuk." Számos egyéb gyakorlattal kombinálva, például folyamatos refaktorálással, folyamatos automatizált egységteszteléssel és folyamatos integrációval kívánják használni. Folyamatos refaktorálás nélkül használva rendezetlen kódhoz és hatalmas átdolgozáshoz vezethet, amelyet technical debt-nek neveznek.

## 14. fejezet

# Helló, Chomsky!

### 14.1. Encoding

Fealadatleírás

### 14.2. OOCWC lexer

Fealadatleírás

### 14.3. I334d1c4 6

Fealadatleírás, ZÖLD FELADAT

### 14.4. Full screen

Fealadatleírás, ZÖLD FELADAT

### 14.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Fealadatleírás

### 14.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Fealadatleírás

---

## 14.7. Perceptron osztály

Fealadatleírás

## 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 < T > 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 összehasonlíthatóak 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.

```
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.equalTo;

import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.Stream;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

public class OrderOfEverythingTest {

 @ParameterizedTest
 @MethodSource("collectionsToSortDataProvider")
 public void testOrderShouldReturnExpectedListWhenCollectionIsPassed(←
 Collection<Integer> input, List<Integer> expectedOutput) {
 // Given as parameters

 // When
 // createOrderedList(List.of(new OrderOfEverythingTest()));
 // ^ ez piros, az OrderOfEverythingTest nem implementálja a ←
 Comparable<OrderOfEverythingTest> -et
 List<Integer> actualOutput = createOrderedList(input);

 // Then
 assertThat(actualOutput, equalTo(expectedOutput));
 }

 private static Stream<Arguments> collectionsToSortDataProvider() {
```



```
 return Stream.of(
 Arguments.of(Collections.emptySet(), Collections.emptyList()),
 Arguments.of(Set.of(1), List.of(1)),
 Arguments.of(Set.of(2,1), List.of(1,2))
);
 }

 private <T extends Comparable<T>> List<T> createOrderedList(Collection<T> input) {
 return input.stream()
 .sorted()
 .collect(Collectors.toList());
 }
}
```

## 14.9. EPAM: Bináris keresés és Buborék rendezés implementálása

Implementálj egy Java osztályt, amely képes egy előre definiált  $n$  darab Integer tárolására. Ennek az osztálynak az alábbi funkcionálisokkal kell rendelkeznie:

- Elem hozzáadása a tárolt elemekhez
- Egy tetszőleges Integer értékről tudja eldönteni, hogy már tároljuk-e (ehhez egy bináris keresőt implementálj)

A tárolt elemeket az osztályunk be tudja rendezni és a rendezett (pl növekvő sorrend) struktúrával vissza tud térni (ehhez egy buborék rendezőt implementálj)

```
public class IntegerCollection {

 int[] array;
 int index = 0;
 int size;
 boolean sorted = true;

 public IntegerCollection(int size) {
 this.size = size;
 this.array = new int[size];
 }

 public IntegerCollection(int[] array) {
 this.size = array.length;
 this.index = this.size;
 this.array = array;
 this.sorted = false;
 }

 public void add(int value) {
 if (size <= index) {
 throw new IllegalArgumentException("The collection is full");
 }
 }
}
```

```
 }
 sorted = false;
 array[index++] = value;
}

public boolean contains(int value) {
 if (!sorted) {
 sort();
 }

 int left = 0, right = size - 1;
 while (left <= right) {
 int mid = left + (right - left) / 2;

 if (array[mid] == value) {
 return true;
 }

 if (array[mid] < value) {
 left = mid + 1;
 } else {
 right = mid - 1;
 }
 }

 return false;
}

public int[] sort() {
 for (int i = 0; i < size - 1; i++) {
 for (int j = 0; j < size - i - 1; j++) {
 if (array[j] > array[j + 1]) {
 int temp = array[j];
 array[j] = array[j + 1];
 array[j + 1] = temp;
 }
 }
 }
 sorted = true;
 return array;
}

@Override
public int hashCode() {
 final int prime = 31;
 int result = 1;
 result = prime * result + Arrays.hashCode(array);
 return result;
}
```

```
@Override
public boolean equals(Object obj) {
 if (this == obj) {
 return true;
 }
 if (!(obj instanceof IntegerCollection)) {
 return false;
 }
 IntegerCollection other = (IntegerCollection) obj;
 return Arrays.equals(array, other.array);
}

@Override
public String toString() {
 return "IntegerCollection [array=" + Arrays.toString(array) + "]";
}
}
```

```
public class Main {

 public static void main(String[] args) {
 IntegerCollection collection = new IntegerCollection(3);
 collection.add(0);
 collection.add(2);
 collection.add(1);
 System.out.println(collection);
 collection.sort();
 System.out.println(collection);
 System.out.println(collection.contains(0));
 System.out.println(collection.contains(1));
 System.out.println(collection.contains(2));
 System.out.println(collection.contains(3));
 System.out.println(collection.contains(4));
 }
}
```

```
IntegerCollection [array=[0, 2, 1]]
IntegerCollection [array=[0, 1, 2]]
true
true
true
false
false
```

14.1. ábra. Output

Az output első sorában kiíratjuk a még nem rendezett tömbünket, majd meghívjuk rá a `sort()` metódust, ezután újra kiíratjuk és láthatjuk a már rendezett tömböt. Ezt követően megnézzük, hogy bizonyos elemek

tagjai-e a tömbünknek. Az outputon láthatjuk, hogy a 0,1,2 még tagja, azonban a 4,5 már nem.

## 14.10. EPAM: Saját HashMap implementáció

Írj egy saját `java.util.Map` implementációt, mely nem használja a Java Collection API-t. Az implementáció meg kell feleljen az összes megadott unit tesztnek, nem kell tudjon kezelni null értékű kulcsokat és a “keySet”, “values”, “entrySet” metódusok nem kell támogassák az elem törlést.

```
import java.util.*;
import java.util.function.BiPredicate;

public class ArrayMap<K, V> implements Map<K, V> {

 private static final int INITIAL_SIZE = 16;
 private static final String NULL_KEY_NOT_SUPPORTED = "This Map ↔
 implementation does not support null keys!";

 private int size = 0;
 private K[] keys = (K[]) new Object[INITIAL_SIZE];
 private V[] values = (V[]) new Object[INITIAL_SIZE];

 @Override
 public int size() {
 return size;
 }

 @Override
 public boolean isEmpty() {
 return size <= 0;
 }

 @Override
 public boolean containsKey(Object key) {
 Objects.requireNonNull(key, NULL_KEY_NOT_SUPPORTED);

 return searchItemInArray(key, keys, Object::equals) != -1;
 }

 @Override
 public boolean containsValue(Object value) {
 int valueIndex = searchItemInArray(value, values, Object::equals);
 return valueIndex > -1 && keys[valueIndex] != null;
 }

 @Override
 public V get(Object key) {
 Objects.requireNonNull(key, NULL_KEY_NOT_SUPPORTED);
 if (size <= 0) {
 return null;
 }
 }
}
```

```
 }

 int keyIndex = searchItemInArray(key, keys, Object::equals);
 if (keyIndex > -1) {
 return values[keyIndex];
 }

 return null;
}

@Override
public V put(K key, V value) {
 Objects.requireNonNull(key, NULL_KEY_NOT_SUPPORTED);

 int keyIndex = searchItemInArray(key, keys, Objects::equals);
 if (keyIndex < 0) {
 keyIndex = findFirstEmptyPlace();
 if (keyIndex < 0) {
 expandArrays();
 }
 keyIndex = size;
 }

 V prevValue = values[keyIndex];

 keys[keyIndex] = key;
 values[keyIndex] = value;
 size++;

 return prevValue;
}

@Override
public V remove(Object key) {
 Objects.requireNonNull(key, NULL_KEY_NOT_SUPPORTED);

 int keyIndex = searchItemInArray(key, keys, Object::equals);
 if (keyIndex > -1) {
 V prevValue = values[keyIndex];

 keys[keyIndex] = null;
 values[keyIndex] = null;
 size--;

 return prevValue;
 }

 return null;
}
```

```
@Override
public void putAll(Map<? extends K, ? extends V> m) {
 m.forEach(this::put);
}

@Override
public void clear() {
 Arrays.fill(keys, null);
 Arrays.fill(values, null);
 size = 0;
}

@Override
public Set<K> keySet() {
 Set<K> result = new HashSet<>();
 for(K i : keys) {
 if (i != null) {
 result.add(i);
 }
 }

 return result;
}

@Override
public Collection<V> values() {
 Collection<V> result = new ArrayList<>();
 for(V i : values) {
 if (i != null) {
 result.add(i);
 }
 }

 return result;
}

@Override
public Set<Entry<K, V>> entrySet() {
 Set<Entry<K, V>> result = new HashSet<>();
 for(int i = 0; i < keys.length; ++i) {
 K key = keys[i];
 if (key != null) {
 V value = values[i];
 result.add(new AbstractMap.SimpleEntry<>(key, value));
 }
 }

 return result;
}
```

```
private <I> int searchItemInArray(I item, I[] array, BiPredicate<I, I> ←
 equalFunction) {
 for (int index = 0; index < array.length; index++) {
 if (equalFunction.test(item, array[index]))
 return index;
 }

 return -1;
}

private int findFirstEmptyPlace() {
 return searchItemInArray(null, keys, Objects::equals);
}

private void expandArrays() {
 int expandedSize = size * 2;

 keys = Arrays.copyOf(keys, expandedSize);
 values = Arrays.copyOf(values, expandedSize);
}
}
```

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

import java.util.Collection;
import java.util.Map;
import java.util.Set;
import java.util.stream.IntStream;
import java.util.stream.Stream;

import static org.hamcrest.CoreMatchers.equalTo;
import static org.hamcrest.MatcherAssert.assertThat;

public class ArrayMapTest {

 private static final String TEST_KEY = "key";
 private static final String TEST_VALUE = "value";
 private static final String NEW_TEST_VALUE = "newValue";

 private static final ArrayMap<String, String> EMPTY_MAP = new ArrayMap ←
 <>();

 @ParameterizedTest
 @MethodSource("provideMapForSize")
 public void sizeShouldReturnCorrectSizeForGivenMap(Map<String, String> ←
```

```
 arrayMap, int expectedSize) {
// Given

// When
int actualSize = arrayMap.size();

// Then
assertThat(actualSize, equalTo(expectedSize));
}

@ParameterizedTest
@MethodSource("provideMapForSize")
public void isEmptyShouldReturnCorrectValueForGivenMap(Map<String, String> ↵
 > arrayMap, int expectedSize) {
// Given
boolean expectedIsEmpty = expectedSize <= 0;

// When
boolean actualIsEmpty = arrayMap.isEmpty();

// Then
assertThat(actualIsEmpty, equalTo(expectedIsEmpty));
}

@ParameterizedTest
@MethodSource("provideMapForContainsKeyOrValue")
public void containsKeyShouldReturnCorrectContainsKeyForGivenMap(Map< ↵
 String, String> arrayMap, String key, boolean expectedContainsKey) {
// Given

// When
boolean actualContainsKey = arrayMap.containsKey(key);

// Then
assertThat(actualContainsKey, equalTo(expectedContainsKey));
}

@ParameterizedTest
@MethodSource("provideMapForContainsKeyOrValue")
public void containsValueShouldReturnCorrectContainsValueForGivenMap(Map< ↵
 String, String> arrayMap, String key, boolean expectedContainsValue) {
// Given

// When
boolean actualContainsKey = arrayMap.containsValue(key);

// Then
assertThat(actualContainsKey, equalTo(expectedContainsValue));
}
```



```
@ParameterizedTest
@MethodSource("provideMapForGet")
public void getShouldReturnCorrectValueForGivenMap(Map<String, String> ↵
 arrayMap, String key, String expectedValue) {
 // Given

 // When
 String actualValue = arrayMap.get(key);

 // Then
 assertEquals(actualValue, expectedValue);
}

@Test
public void putShouldAddNewValueAndReturnPreviousOne() {
 // Given
 Map<String, String> arrayMap = generateGivenSizeMap(3);
 arrayMap.put(TEST_KEY, TEST_VALUE);

 // When
 String previousValue = arrayMap.put(TEST_KEY, NEW_TEST_VALUE);

 // Then
 assertEquals(previousValue, TEST_VALUE);
 assertEquals(arrayMap.get(TEST_KEY), NEW_TEST_VALUE);
}

@Test
public void putShouldCorrectlyExpandInnerArraysIfNeeded() {
 // Given

 // When
 Map<String, String> arrayMap = generateGivenSizeMap(100);

 // Then
 assertEquals(arrayMap.size(), 100);
 for(int i = 0; i < 100; i++) {
 String keyValue = String.valueOf(i);
 assertEquals(arrayMap.get(keyValue), keyValue);
 }
}

@Test
public void removeShouldDeleteKeyAndValueFromMap() {
 // Given
 Map<String, String> arrayMap = new ArrayMap<>();
 arrayMap.put(TEST_KEY, TEST_VALUE);

 // When
```

```
arrayMap.remove(TEST_KEY);

// Then
assertThat(arrayMap.size(), equalTo(0));
assertThat(arrayMap.containsKey(TEST_KEY), equalTo(false));
assertThat(arrayMap.containsValue(TEST_VALUE), equalTo(false));
}

@Test
public void putAllShouldAddAllEntriesToMap() {
 // Given
 Map<String, String> originalMap = generateGivenSizeMap(10);
 Map<String, String> resultMap = new ArrayMap<>();

 // When
 resultMap.putAll(originalMap);

 // Then
 for(int i = 0; i < 10; i++) {
 String keyValue = String.valueOf(i);
 assertThat(resultMap.get(keyValue), equalTo(keyValue));
 }
}

@Test
public void clearShouldRemoveAllKeysAndValues() {
 // Given
 Map<String, String> arrayMap = generateGivenSizeMap(2);
 arrayMap.put(TEST_KEY, TEST_VALUE);

 // When
 arrayMap.clear();

 // Then
 assertThat(arrayMap.size(), equalTo(0));
 assertThat(arrayMap.get(TEST_KEY), equalTo(null));
}

@Test
public void keySetShouldReturnSetContainingAllKeys() {
 // Given
 Map<String, String> arrayMap = generateGivenSizeMap(3);

 // When
 Set<String> keySet = arrayMap.keySet();

 // Then
 assertThat(keySet.size(), equalTo(3));
 for(int i = 0; i < 3; i++) {
 String k = String.valueOf(i);
```

```
 assertThat(keySet.contains(k), equalTo(true));
 }
}

@Test
public void valueShouldReturnCollectionContainingAllValues() {
 // Given
 Map<String, String> arrayMap = generateGivenSizeMap(3);

 // When
 Collection<String> values = arrayMap.values();

 // Then
 assertThat(values.size(), equalTo(3));
 for(int i = 0; i < 3; i++) {
 String k = String.valueOf(i);
 assertThat(values.contains(k), equalTo(true));
 }
}

@Test
public void entrySetShouldReturnSetContainingAllEntries() {
 // Given
 Map<String, String> arrayMap = generateGivenSizeMap(3);

 // When
 Set<Map.Entry<String, String>> entries = arrayMap.entrySet();

 // Then
 assertThat(entries.size(), equalTo(3));
 for(Map.Entry<String, String> entry : entries) {
 assertThat(entry.getKey(), equalTo(entry.getValue()));
 }
}

private static Stream<Arguments> provideMapForSize() {
 Map<String, String> removedItemMap = generateGivenSizeMap(5);
 removedItemMap.remove(String.valueOf(2));

 Map<String, String> removedEmptyMap = generateGivenSizeMap(1);
 removedEmptyMap.remove(String.valueOf(0));

 return Stream.of(
 Arguments.of(EMPTY_MAP, 0),
 Arguments.of(generateGivenSizeMap(2), 2),
 Arguments.of(generateGivenSizeMap(10), 10),
 Arguments.of(removedItemMap, 4),
 Arguments.of(removedEmptyMap, 0)
);
}
```

```
private static Stream<Arguments> provideMapForContainsKeyOrValue() {
 Map<String, String> arrayMap = generateGivenSizeMap(2);

 return Stream.of(
 Arguments.of(EMPTY_MAP, "0", false),
 Arguments.of(arrayMap, "1", true),
 Arguments.of(arrayMap, "2", false),
 Arguments.of(generateGivenSizeMap(5), "asd", false)
);
}

private static Stream<Arguments> provideMapForGet() {
 Map<String, String> arrayMap = generateGivenSizeMap(2);
 arrayMap.put("key", "value");

 return Stream.of(
 Arguments.of(EMPTY_MAP, "0", null),
 Arguments.of(arrayMap, "0", "0"),
 Arguments.of(arrayMap, "2", null),
 Arguments.of(arrayMap, "key", "value")
);
}

private static Map<String, String> generateGivenSizeMap(int size) {
 ArrayMap<String, String> result = new ArrayMap<>();
 IntStream.range(0, size)
 .forEachOrdered(i -> result.put(String.valueOf(i), String.valueOf(i + 1)));

 return result;
}
}
```

## 15. fejezet

# Helló, Stroustrup!

### 15.1. Másoló-mozgató szemantika

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

```
BinRandTree(const BinRandTree& old) {
 std::cout << "BT copy ctor" << std::endl;

 root = cp(old.root, old.treep);

}

BinRandTree& operator=(const BinRandTree& old) {
 std::cout << "BT copy assign" << std::endl;

 BinRandTree tmp{ old };
 std::swap(*this, tmp);
 return *this;
}

BinRandTree(BinRandTree&& old) {
 std::cout << "BT move ctor" << std::endl;

 root = nullptr;
 *this = std::move(old);
}

BinRandTree& operator=(BinRandTree&& old) {
 std::cout << "BT move assign" << std::endl;

 std::swap(old.root, root);
 std::swap(old.treep, treep);

 return *this;
}
```

```

BT ctor
BT ctor
--0 1 2

--1 1 2
BT copy ctor
BT dtor
BT dtor

--1 1 42
BT dtor
```

15.1. ábra. Output

Ezt a kódcipetet az előző félév binfa forráskódjából vettem ki, hiszen itt jól látszik, hogy a másoló értékadást a másoló konstruktorra, valamint a mozgató konstruktort a mozgató értékadásra alapoztuk. Mivel egy fát akarunk átmásolni egy másikba, így meghívódik a copy assign. Az ebben található `BinRandTree tmp{old}` sor miatt meghívódik a copy ctor. Majd a swap miatt meghívódik a move ctor, amiben található move miatt meghívódik a move assign.

## 15.2. EPAM: It's gone. Or is it?

Adott a következő osztály:

```
public class BugousStuffProducer {
 private final Writer writer;
 public BugousStuffProducer(String outputFileName) throws ←
 IOException {
 writer = new FileWriter(outputFileName);
 }
 public void writeStuff() throws IOException {
 writer.write("Stuff");
 }
 @Override
 public void finalize() throws IOException {
 writer.close();
 }
}
```

Mutass példát arra az esetre, amikor előfordulhat, hogy bár a program futása során meghívtuk a `writeStuff()` metódust, a fájl, amibe írtunk még is üres.

Magyarázd meg, miért. Mutass alternatívát.

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

public class GoneOrIsIt {

 private static class BugousStuffProducer {
 private final Writer writer;
```

```
public BugousStuffProducer(String outputFileName) throws ←
 IOException {
 writer = new FileWriter(outputFileName);
 }
 public void writeStuff() throws IOException {
 writer.write("Stuff");
 }
 @Override
 public void finalize() throws IOException {
 writer.close();
 }
}

public static void main(String[] args) throws IOException {
 BugousStuffProducer bugousStuffProducer = new BugousStuffProducer ←
 ("example.txt");
 bugousStuffProducer.writeStuff();
 System.gc();
}
}
```

Itt azért lesz üres a txt, mivel nem garantált, hogy a main() után lefut a finalize().

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.Writer;

public class GoneOrIsItFixed {
 private static class BugousStuffProducer implements AutoCloseable {
 private final Writer writer;

 public BugousStuffProducer(String outputFileName) throws
 IOException {
 writer = new FileWriter(outputFileName);
 }
 public void writeStuff() throws IOException {
 writer.write("Stuff");
 }

 @Override
 public void close() throws Exception {
 writer.close();
 }
 }

 public static void main(String[] args) throws Exception {
 try (BugousStuffProducer bugousStuffProducer = new ←
 BugousStuffProducer("fixedExample.txt")) {
 bugousStuffProducer.writeStuff();
 }
 }
}
```

```
}
```

## 15.3. EPAM: Kind of equal

Adott az alábbi kódrészlet.

```
// Given
String first = "...";
String second = "...";
String third = "...";
// When
var firstMatchesSecondWithEquals = first.equals(second);
var firstMatchesSecondWithEqualToOperator = first == second;
var firstMatchesThirdWithEquals = first.equals(third);
var firstMatchesThirdWithEqualToOperator = first == third;
```

Változtasd meg a `String third = "...";` sort úgy, hogy a `firstMatchesSecondWithEquals`, `firstMatchesSecondWithEqualToOperator`, `firstMatchesThirdWithEquals` értéke `true`, a `firstMatchesThirdWithEqualToOperator` értéke pedig `false` legyen. Magyarázd meg, mi történik a háttérben.

```
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;

import org.junit.jupiter.api.Test;
public class KindofEqualTest {

 @Test
 public void testKindofEqualExercise() {
 // Given
 String first = "...";
 String second = "...";
 String third = new String("...");

 // When
 var firstMatchesSecondWithEquals = first.equals(second);
 var firstMatchesSecondWithEqualToOperator = first == second;
 var firstMatchesThirdWithEquals = first.equals(third);
 var firstMatchesThirdWithEqualToOperator = first == third;

 // Then
 assertThat(firstMatchesSecondWithEquals, is(true));
 assertThat(firstMatchesSecondWithEqualToOperator, is(true));
 assertThat(firstMatchesThirdWithEquals, is(true));
 assertThat(firstMatchesThirdWithEqualToOperator, is(false));
 }
}
```



A first és a second ugyanarra a objektumra referencia a String interning miatt, a third pedig egy másik, de azonos tartalmú objektum.

## 15.4. EPAM: Java GC

Mutasd be nagy vonalakban hogyan működik Java-ban a GC (Garbage Collector). Lehetséges az OutOfMemoryError kezelése, ha igen milyen esetekben?

Források:

<https://medium.com/@hasithalgamge/seven-types-of-java-garbage-collectors-6297a1418e82>

<https://stackoverflow.com/questions/2679330/catching-java-lang-outofmemoryerror>

A Java olyan objektum-orientált programozási nyelv, amiben van Automatic Garbage Collection. A Java automatikusan lefoglalja és elosztja a memóriát, így a programokat nem terheli ez a feladat. A Java-12-ben 7 garbage collector típust különböztetünk meg. Mint fejlesztő fontos, hogy tudjuk mikor melyik GC-t használjuk, hiszen mindnek megvannak az előnyei és a hátrányai, így a saját projektünkhöz megfelelően kell választanunk.

### 1. Serial Garbage Collector:

Ez a legegyszerűbb GC, amit alapvetően single thread környezethez terveztek. Ez a GC megvalósítás fagyasztja az összes alkalmazásszálat, amikor fut. Egyetlen szálat használ a szemétszállításhoz. Ezért nem jó ötlet több szálon futó alkalmazásokban, például szerver környezetben használni.

### 2. Paralell Garbage Collector:

Ez több szálat használ a szemétszállításhoz. Abban hasonlít a Serial Garbage Collector-hoz, hogy ez is lefagyasztja az összes alkalmazásszálat míg szemetet szállít. Azoknak az alkalmazásoknak a legalkalmasabb, amelyek elviselhetik az alkalmazás szüneteit.

### 3. Concurrent Mark Sweep (CMS) Garbage Collector:

Több szemétszállító szálat használ, felkeresi a kupac memóriában az elszállítani kívánt példányokat, ezeket megjelöli, majd „kisöpri” a memóriából. Olyan alkalmazásokhoz ajánlott használni, melyek meg tudják osztani a processzor erőforrásait a GC-vel az alkalmazás futása közben.

### 4. G1 (Garbage First) Garbage Collector:

Felosztja a kupac memóriát, és párhuzamosan gyűjt bennük. Miután minden területen megjelölte a szemetet megnézi, hogy melyik terület a leginkább üres és elsőnek ott kezd el gyűjtögetni. Olyan alkalmazásokra tervezték, amelyek több processzoros gépeken futnak nagy memóriaterülettel.

### 5. Epsilon Garbage Collector:

Az Epsilon nem működő vagy passzív szemétygyűjtő. Lefoglalja a memóriát az alkalmazás számára, de a nem használt objektumokat nem gyűjti össze. A szemétygyűjtő célja az alkalmazások teljesítményének mérése és kezelése.

### 6. Z Garbage Collector:

A ZGC minden költséges munkát egyidejűleg végez, anélkül, hogy 10 ms-nál hosszabb időre leállítaná az alkalmazásszálak végrehajtását, ami alkalmassá teszi alacsony késleltetést igénylő alkalmazásokhoz és / vagy nagyon nagy heap-et használó alkalmazásokhoz.

## 7. Shenandoah:

A Shenandoah egy rendkívül alacsony szünetidővel rendelkező szemétgyűjtő, amely csökkenti a GC szünetidejét azzal, hogy több szemétgyűjtési munkát végez a futó Java programmal egyidejűleg.

Általában, hogy az `OutOfMemoryError` egy blokkmemória-kiosztás miatt következik be, amely nem elégedett a kupac megmaradt erőforrásaival. Amikor egy hiba eldobásra kerül, a kupac ugyanannyi allokált objektumot tartalmaz, mint a sikertelen kiosztás előtt, és ideje, hogy eldobja a futásidejű objektumok hivatkozásait, hogy még több memória felszabadulhasson, aminek tisztításra lehet szüksége. Ezekben az esetekben kockázatos folytatni, hiszen nem lehetünk benne teljesen biztosak, hogy a JVM javítható állapotban van.

## 16. fejezet

# Helló, Gödel!

### 16.1. Gengszterek

Fealadatleírás

### 16.2. C++11 Custom Allocator

Fealadatleírás

### 16.3. STL map érték szerinti rendezése

Fealadatleírás

### 16.4. Alternatív Tabella rendezése

Fealadatleírás

### 16.5. Prolog családfa

Fealadatleírás

### 16.6. GIMP Scheme hack

Fealadatleírás

---

## 16.7. EPAM: Mátrix szorzás Stream API-val

Fealadatleírás

## 16.8. EPAM: LinkedList vs ArrayList

Fealadatleírás

## 16.9. EPAM: Refactoring

Fealadatleírás

DRAFT

## 17. fejezet

# Helló, hetedik hét!

### 17.1. FUTURE tevékenység editor

Fealadatleírás

### 17.2. OOCWC Boost ASIO hálózatkezelése

Fealadatleírás

### 17.3. SamuCam

Fealadatleírás

### 17.4. BrainB

Fealadatleírás

### 17.5. OSM térképre rajzolása

Fealadatleírás

### 17.6. EPAM: XML feldolgozás

Fealadatleírás

## 17.7. EPAM: ASCII Art

Fealadatléírás

## 17.8. EPAM: Titkos üzenet, száll a gépben!

Fealadatléírás

DRAFT

## 18. fejezet

# Helló, Lauda!

### 18.1. Port scan

Fealadatleírás

### 18.2. AOP

Fealadatleírás

### 18.3. Android Játék

Fealadatleírás, ZÖLD FELADAT

### 18.4. Junit teszt

Fealadatleírás

### 18.5. OSCI

Fealadatleírás

### 18.6. OSCI2

Fealadatleírás, KÉK FELADAT

---

## 18.7. OSCI3

Fealadatleírás, KÉK FELADAT

## 18.8. EPAM: DI

Fealadatleírás

## 18.9. EPAM: JSON szerializáció

Fealadatleírás

## 18.10. EPAM: Kivételkezelés

Fealadatleírás

DRAFT



## 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

C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven

Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-

Ebből a két könyvből pár oldalas esszé jellegű kidolgozást kérek, Java és C++ összehasonlítás mentén, pl. kb.: kifejezés fogalom ua., Javában minden objektum referencia, mindig dinamikus a kötés, minden függvény virtuális, klónozás stb.

A Java egy objektumorientált nyelv, ami azt jelenti, hogy egy Java program objektumok, és ezek mintáinak tekinthető osztályok összessége. Egy osztály változókból, valamint metódusokból épül fel. A változóban adatokat lehet tárolni, a metódusokban pedig megadjuk, hogy milyen műveleteket szeretnénk elvégezni rajtuk.

Mikor változókat használunk meg kell adni azok típusát is. Ilyen például a boolean, ami igaz/hamis értéket vesz fel, a char, ami 16 bites Unicode karakter, a byte, a short, az int és a long, ami 8, 16, 32, és 64 biten tárol előjeles egész számokat, illetve a float és a double, melyek 32 és 64 bites lebegőpontos számok tárolására szolgálnak. Ezen kívül vannak még sztringek, melyekre a String típussal hivatkozunk. A változókhoz = operátorral tudunk értéket rendelni. Arra is figyelniünk kell, hogy ezeket a változókat hogyan nevezzük el. Csak betűvel kezdődhetnek, majd betűvel vagy számmal folytatódhatnak, betűk közé tartozik még a \_ és a \$ karakter.

Java nyelvben [] jelöléssel adjuk meg a tömböket. Ezek indexelése 0-val kezdődik. A tömb értéket a new operátorral adjuk meg. Méretét a length használatával kérdezhetjük le.

Egyszerű típusok és objektumok inicializálásához literálokat használunk: objektumok (az egyetlen literál a null, bárhol használható objektumreferencia helyett), logikai érték (true vagy false értékek), egész számok (lehet oktális, hexadecimális vagy decimális), lebegőpontos számok (szerepelhet benne tizedespont vagy exponens is), karakterek (aposztrófok között adjuk meg), szövegek (idézőjelek között adjuk meg), osztályok.

Felsorolási típusokkal definiálhatunk könnyen megjegyezhető, kifejező nevű értékeket tartalmazó típusokat, például a hét napjait így adnánk meg: enum Napok {HÉTFŐ, KEDD, ..., VASÁRNAP;}. Ezekhez a nyelv több hasznos műveletet biztosít. Az ordinal függvény megadja, hogy egy típusérték hányadik helyen szerepel a felsorolásban. Egy felsorolási típus értékeit tömbbe is tehetjük, erre szolgál a values függvény.

Javában beszélhetünk még elágazásokról és ciklusokról is. Az elágazásoknak két féle szerkezete lehet. Egyszerű elágazásokhoz az if szerkezetet használjuk melynek szintaxisa a következő: if (logikai kifejezés) utasítás1 else utasítás2. Összetett elágazásoknál a switch elágazást használjuk. Itt a fejrészben lévő kifejezés lesz kiértékelve, ezt követően pedig a case ágakban lévő címkeértékek lesznek megvizsgálva. Ha valamelyik címkeérték megegyezik a fejlécben lévő kifejezéssel, akkor az abban lévő utasítások lesznek végrehajtva a switch végéig vagy a break utasításig. Ha nincs megfelelő címke, akkor a default ág kerül végrehajtásra. Ciklusok esetében beszélhetünk előltesztelő, hátultesztelő, léptető, valamint bejáró ciklusokról. Az előltesztelő ciklusban először a ciklus feltétele lesz kiértékelve. Ez egy logikai kifejezés. Ha ez igaz értékű, akkor a ciklus folytatja a futását, ha kezdetben hamis, akkor a ciklusmag egyszer sem fog lefutni. Szintaxisa: while (logikai kifejezés) utasítás. A hátultesztelő ciklus a ciklusmag lefutása után vizsgálja meg a feltételt, ami, ha igaz, a ciklusmag újra lefut. Ha kezdetben hamis, akkor is egyszer le fog futni a ciklusmag. Szintaxisa: do utasítás while (logikai kifejezés). A léptető ciklus nem más, mint a for ciklus, melynek szintaxisa for (kezdet; logikai kifejezés; léptetés). A for ciklus fejrészában deklarált változók csak a cikluson belül lesznek érvényesek. A bejáró ciklussal egy adatszerkezet bejárását végezhetjük, ez lehet tömb, sorozat vagy halmaz. Szintaxisa: for (típus változó: gyűjtemény) utasítás.

Beszélhetünk még címkékről is, ezek bármely utasítás elé írhatóak, az utasítás egyértelmű azonosítását teszik lehetővé. A break utasítást egy blokkból való kilépésre használjuk. Ha nem áll mellette címke, akkor a legbelső blokkból lép ki. A continue utasítással át lehet ugrani egy ciklus magjának hátralévő részét. Egy metódusból a return [kifejezés] utasítással térhetünk vissza.

A kifejezések kiértékelési sorrendjét az operátorok határozzák meg. Először a legbelső zárójel kerül végrehajtásra, ha nincs zárójel, akkor a nagyobb prioritású operátor lesz végrehajtva. Ha több egyenlő prioritású operátor van, akkor balról jobbra haladunk. Beszélhetünk postfix operátorokról (tömb indexelése, minősített név, zárójelezett kifejezés, postfix csökkentő, illetve növelő operátorok) és prefix operátorokról (prefix csökkentő, illetve növelő műveletek, előjelváltás, logikai tagadó művelet, bitsintű nem művelet). Ezekon kívül léteznek még objektum létrehozására (new) és típuskényszerítésre ((típus)kifejezés) használható operátorok, valamint multiplikatív (\*, /, %) és additív (+, -) operátorok, léptető műveletek, összehasonítások, egyenlőségvizsgálatok (==, !=), bitenkénti műveletek (~, |), logikai műveletek (||, &&), feltételes kifejezés (? : operátor), és értékadások.

A Java egy erősen típusos nyelv, szinte minden esetben megvizsgálja, hogy a kifejezésekben összeegyeztethető típusok vannak-e. Az egyszerű típusok között végrehajthatunk értékadásokat, ha az értéket fogadó változó tartománya nagyobb, mint az eredeti. Egy adott osztály objektumának referenciáját bárhol fel lehet használni, ahol valamelyik ősosztályhoz tartozó típus van a kifejezésben. Ezek az automatikus konverziók közé tartoznak. Ha az előbb felsorolt konverziókat nem érjük el, vagy nem egyértelműek, akkor explicit módon is megadhatjuk őket, de ezek nem biztonságosak, vagy információvesztéssel járnak. Szövegek esetében, ha egy kifejezésben String típusra van szükség, de a kifejezésben lévő elem nem az, akkor a fordító meghívja az adott objektum toString metódusát. A szövegeket tudjuk konkatenálni a + operátorral, ebben az esetben is a toString metódus lesz meghívva.

A Javában a legkisebb önálló egységek az osztályok, melyek azonos típusú dolgok modelljét írja le. Ezek létrehozására a class kulcsszót kell használnunk. a class szó előtt szerepelhetnek módosítók, pl public (az adott osztály hozzáférhető más csomagokból is), abstract vagy final. Ha valamelyik osztály nem public, akkor csak a saját csomagján belül használjuk. Az osztály fejlécében kell megadni azt is, hogy az adott osztályt melyik másik osztály kiterjesztéseként definiáljuk, illetve, hogy mely interfészeket valósítja meg. Osztályon belül tetszőlegesen felsorolhatjuk annak változóit és metódusait. Minden elemnek külön tudjuk szabályozni a láthatóságát, ez lehet public, ami mindenki számára látható, private, ami senki más számára nem látható, illetve protected, ami csak a leszármazottak számára látható. Osztályon belül a new operátorokkal hozhatunk létre új objektumokat. Az objektum egy elemére az objektum és az elem nevéből

álló, ponttal elválasztott minősített névvel tudunk hivatkozni. Létezik egy speciális referencia érték, a null, amely nem mutat egyetlen objektumra sem. A static kulcsszóval deklarált elemek nem az objektumhoz fognak tartozni, hanem az osztályhoz, ami azt jelenti, hogy a new operátor használatakor nem lesz számukra memóriaterület foglalva, az objektum adattagjainak inicializálásakor nem kell ezeknek is értéket adni, nemcsak az objektum, hanem osztály nevén keresztül is lehet rájuk hivatkozni. Itt is használunk különböző módosítókat. Vannak félnyilvános, nyilvános, privát és leszármazottban hozzáférhető tagok. Léteznek osztályváltozók, melyek olyan változók, amik magához az osztályhoz kapcsolódnak. Az osztálymetódusok az osztály példányain értelmezett műveleteket jelentik. Minden programban kell lennie egy main metódusnak, ami maga a főprogram. Léteznek konstruktorok, ezek olyan programkódok, melynek végrehajtása a példányosításakor automatikusan megtörténik. Annyiban különböznek a metódusoktól, hogy az azonosító kötött, nem szabad semmilyen visszatérési típust feltüntetni, és csak hozzáférési kategóriát meghatározó módosítókat alkalmazhatunk. Ezek törzsét úgy kell megírunk, mintha void metódus törzse lenne. Konstruktorok mellett beszélhetünk destruktorkról is, amiket akkor használunk, ha értesülni akarunk egy adott objektum megsemmisüléséről. Fontos, hogy a metódusunk protected void és paraméter nélküli legyen. Meghívása a finalize metódussal történik.

Javában fontos az öröklődés fogalma. Legegyszerűbb esete az, amikor egy már létező osztályt szeretnénk kibővíteni új változókkal vagy metódusokkal. Az új osztály a bővítendő osztálytól függetlenül lehet public vagy nem public. Az osztálydefinícióban az extends kulcsszóval jelezzük, hogy melyik osztályt szeretnénk bővíteni. Az eredeti osztályt szülőosztálynak, míg a bővítettet gyerekosztálynak nevezzük. A gyermek csak azokhoz az elemekhez férhet hozzá, amelyeket a szülő engedélyez. Például a szülő private változóit a gyermek közvetlenül nem érheti el, azonban a szülő örökölt metódusain keresztül használhatja őket. A gyermek nem örökli a konstruktorokat. Ez úgy oldható meg, hogy a konstruktor törzsében a this kulcsszó helyett super-t használunk. Az osztályok rokonsági viszonyát osztályhierarchiának nevezzük. Az Object osztály egy kiemelt osztály, mely implicit módon minden olyan osztálynak szülője, amelynek definíciójában nem adtunk meg extends tagot.

Az osztályok mellett a nyelv másik fontos építőkövei az interfészek. Egy interfész egy új referencia típus, absztrakt metódusok deklarációjának és konstans értékeknek az összessége. Itt hiányoznak a valódi változtatható adattagok, és a metódusokat is csak deklarálni lehet bennük. Használata implementáción keresztül történik. Egy osztály akkor implementál egy interfészt, ha az összes, az interfész által specifikált metódushoz implementációt ad. Az interfészek között is létezik öröklődés, annyi különbséggel, hogy itt lehet többszörös öröklődés. Egy interfész a következő módon épül fel: opcionális interfészmódosítók (public vagy abstract), név, az interfész által kiterjesztett interfészek (ha vannak), és az interfésztörzs.

Egy programozónak fontos arra is törekednie, hogy kódja átlátható és könnyen olvasható legyen. A Javában erre a célra úgynevezett csomagokat használunk. Minden csomag egy önálló namespace-t vezet be, amivel egyedivé teszi a benne definiált típusok nevét, ezzel elkerüljük az egyező változónevek keveredését. A csomagoknál is beállíthatjuk azok láthatóságát. A csomagok szerkezete hierarchikus, tehát köztük alárendelő viszony lehet. Egy csomagban tetszőleges számú alcsomag lehet. Az alcsomag annyiban különbözik a csomagtól, hogy megadják melyik csomagnak képezi a részét. Ennek megfelelően egy alcsomagnak is lehetnek további alcsomagjai. Egy csomag tartalmazhat alcsomagokat és típusokat. Ezek használatához kiterjesztett neveket kell majd használni. Új csomagot a package csomagdeklarációval hozhatunk létre. Emellett létezhetnek még névtelen csomagok. Ezek tárolása adatbázisban vagy fájlrendszerben történik. A JDK a fájlrendszerbeli tárolást támogatja. Adatbázisban történő tárolás esetén a csomagok, a fordítási egységek és a hozzájuk tartozó bájtkód kerül tárolásra. A fordítási egység az a hely, ahol a csomag kódja található. A csomagdeklaráció utáni lehetséges elemek az importdeklarációk. Ez lehetővé teszi más csomagokban deklarált publikus típusok, vagy más osztályokban deklarált statikus tagok egyszerű elérését. Az importdeklarációknak három fajtája van: egyszerű, igény szerinti, és statikus. Mindháromnál az import

kulcsszót kell használni. Egyszerű típusimport-deklaráció esetében az import után egy típus kiterjesztett nevét kell megadni, ennek hatására a fordítási egységben a típus egyszerű nevét is lehet használni. Az igény szerinti típusimport-deklaráció arra jó, hogy ne csak egy meghatározott típust lehessen elérni a csomagból, hanem a csomag összes publikus típusát. Statikus importdeklaráció esetén minősítés nélkül lehet hivatkozni osztályváltozók értékeire.

Egy metódus két részből áll, fejből és törzsből. a fej megadja a metódus visszatérési típusát, az azonosítóját, és a paramétereit. A fej előtt használhatunk módosítókat, például public, protected, private, abstract, static, final synchronized és native. A törzs a működést definiáló utasításblokkot tartalmazza. Javában a metódusokat is class kulcsszóval kell bevezetni. A metódus meghívásakor meg kell adnunk definíció szerint a paramétereit. Ezeket zárójelben vesszővel elválasztva tehetjük meg. Void típusú metódust akkor hívhatunk, ha nincs szükség visszatérési értékre. A törzsben a this pszeudováltozóval hivatkozhatunk az aktuális példányra.

Fontos, hogy programunk megbízható legyen, és kivételes helyzetekben is meg tudjuk szabni annak menetét. Amikor valamilyen hiba lép fel a metódus futása során, akkor egy kivételobjektum jön létre, amely információkat tartalmaz a kivétel fajtájáról és a program futási állapotáról. A kivételeknek három fajtája lehet: program futása közben történt rendellenes dolog, a program egy throw utasítása váltott ki kivételt, vagy aszinkron kivétel lépett fel. A kivételek lehetnek ellenőrzöttek vagy nem ellenőrzöttek. Az ellenőrzötteket az kapcsolja össze, hogy mindig el kell kapni vagy specifikálni. ha egy program ennek nem tesz eleget akkor a fordító hibát jelez. Erre használjuk az exception handling-et. A throw kulcsszóval tudunk kivételt kiváltani. A try -catch-finally szerkezettel a kivételkezelőnek egy-egy blokkját lehet létrehozni. A try blokk vezérli a benne fellépő kivételek kezelését, meghatározza a blokkot követő catch ágak és finally ág érvényességi körét. Minden try blokk után következni kell egy ahhoz tartozó catch vagy finally blokknak. Lényeges a catch ágak sorrendje, mivel lehet olyan kivétel, amit több ág is kezel. A teljes kód lefutása után a következők történhetnek: nem lép fel kivétel, kivétel lépett fel és valamelyik catch ágon megy tovább a program, vagy a fellépett kivételt egyik catch ág sem kezeli. A finally után szereplő utasítások mindhárom esetben végrehajtnak. A catch ágak a kivételeket úgy kezelik, hogy hibaüzeneteket írnak ki. A kivételek osztálya a java.lang.Throwable, aminek két gyermek osztálya van, a java.lang.Exception és a java.lang.Error. Az Error utódai a nem ellenőrzött kivételosztályok közé tartoznak és tetszőleges helyen bekövetkezhet. Az Exception-nek számos leszármazottja van, egyik ilyen a RuntimeException, mely futás idejű kivételeket kezel. A megbízhatóbb programok kezelése érdekében állításokat hozunk létre, amely program futása közben kiértékelődő logikai kifejezés, ami helyes működés esetén igaz értéket ad. Az állításoknak is több fajtája van: helyesség ellenőrzése tetszőleges ponton, program haladásának kontrollálása, és elő- és utófeltételek.

A Java 5. verziójában megjelentek a generikusok, melyek segítségével az elvégzendő feladatokat típusokkal is paraméterezhetjük. A generikusokat különböző célokra használják: típusok feletti absztrakció, növekvő biztonság, egyszerű implementálhatóság, hatékonyság, kompatibilitás az eddig megírt Java kódokkal, és kompatibilitás a jövőbeli fejlesztésekkel. Azt, hogy a korábban megírt kódokat és osztálykönyvtárakat megőrizhessük típustördelést kell használnunk. A típustördelés az az eljárás, amikor egy típust leképezünk egy másik, most már nem paraméteres típusra. Törés után az adott típus paraméter nélküli, azaz nyers (raw) verzióját kapjuk. Az Inner típus metódusa az Outer típusparaméterétől függ. Ha az Outer-t típusparaméter nélkül használjuk, akkor az Inner-hez sem köthetünk típust, az is nyers típus lesz. Egy generikus típusnak lehet static tagja. Ezek egy példányban léteznek az őket tartalmazó típusban, függetlenül az objektumok számától és attól, hogy hány féle paraméter típust alkalmazunk a generikusra. A generikusok bevezetése mögött az egyik legnagyobb motiváció a gyűjtemény keretrendszer biztonságossá tétele volt. Javában intenzíven használjuk az interfész fogalmát, minden olyan esetben, amikor az osztályok egy csoportjának valamilyen közös tulajdonságát szeretnénk leírni. Mivel a generikus és nem generikus osztályok egyaránt

részt vehetnek az öröklődésben, számos kombinációt kell figyelembe vennünk: generikus osztály örököl nem generikustól, nem generikus osztály örököl egy konkrét típusparaméterrel ellátott generikustól, generikus osztály örököl egy általános generikustól, generikus osztály örököl egy generikus nyers típusától, nem származtathatunk nem-generikus osztályt generikus bázisból, nem működik a mixin konstrukció, amikor a generikus a típusparaméterétől örököl. Az atípusosság invariáns a típusparaméterekre, ami azt jelenti, hogy a típusparaméterek közötti öröklődés relációk teljesen közömbösek a generikus típusok közötti relációk szempontjából. A következők kivételével az összes referencia-típus lehet generikus: névtelen belső osztályok, kivétel típusok, felsorolási típusok.

Javában megtalálhatóak különböző gyűjtemények is, melyek olyan típuskonstrukciós eszközök, melyeknek célja egy vagy több típusba tartozó objektumok példányainak memóriában történő összefoglaló jellegű tárolása, manipulálása, és lekérdezése. A halmaz interfész nem tartalmaz kifejezéseket a Collection interfészhez képest, csupán annak bizonyos metódusaira tesz megszorításokat. A halmazműveletek a következők: a `containsAll` metódus a tartalmazást segíti eldönteni, az `addAll` az uniót, a `retainAll` a metszet, a `removeAll` pedig a kivonás. A lista szintén a Collection interfész leszármazottja, mint List interfész. A listában az elemek duplikáltan is szerepelhetnek és lényeges a sorrendjük. A lista interfész tartalmaz néhány kiterjesztést, miszerint az elemeket pozíció szerint elérhetjük, tudunk keresni, valamit tartalmaz listaiterációt és részlista-kezelést. A listáknál a `remove` parancs csak a megadott elem legelső előfordulását törli, az `add` és az `addAll` műveletek a lista végére szúrják be az új elemeket. A FIFO adatszerkezeteket nevezzük még soroknak is, aminek az a lényege, hogy a legelőször betett objektumot tudjuk majd elsőnek kivenni. A sorokon a következő műveleteket hajthatjuk végre: az `offer`rel teszünk be új elemet a sorba, a `remove` és a `poll` metódusokkal vehetünk ki elemet, az `element` és a `peek` a legelső elemet adják vissza. A `LinkedList` egy olyan adatszerkezet, amely kulcs-érték párokat tárol, ennek megvalósítása a Map interfész. Minden kulcshoz csak egy érték tartozhat. Leképzések esetén szükség lehet különböző rendezésekre, ilyenkor lehet maga a gyűjtemény rendezett, vagy pedig egy rendező algoritmus segítségével sorba tesszük az elemeket. Ilyen algoritmus lehet például bináris keresés, minimum- és maximumkeresés, rotálás, helyettesítés, csere vagy gyakoriság lekérdezése.

Egy Java program tervezése három szakaszra osztható: analízis, azaz a probléma megfogalmazása és körvonalazása, rendszertervezés, azaz a körvonalazott rendszer részekre bontása, osztálytervezés, azaz a rendszert alkotó osztályok és kapcsolataik kidolgozása.

A C++-ban a memóriát háromféle módon használjuk: van statikus memória, amelyikben egy objektum számára a szerkesztőprogram a program futásának idejére foglal helyet, van automatikus memória, melyben függvényargumentumok és a helyi változók kapnak helyet, valamint a szabad tár, melyet a programozó kezel. A Java programok virtuális gépen futnak, így a memóriát nem tudjuk elérni, csak szimbolikusan. C++-ban az objektumok élettartama pontosan ismert, az automatikus változók a deklaráció blokkba történő belépéskor keletkeznek, és pontosan a blokk végrehajtásáig élnek. A szabad tárban allokkált objektumok élettartamát mi határozzuk meg. Javában egészen addig míg referenciánk van egy adott objektumra, az egészen addig él. A C++ nyelv többparadigmás, míg a Java csak az objektumorientált programozást támogatja. Fordításkor C-ben és C++-ban először az előfordító dolgozza fel a programot. Ilyen a Java-ban nincs. C++-ban bizonyos forrásállományokhoz headerekkel férünk hozzá. Ezt a program fejlécében `#include` paranccsal jelezzük. Erre Java nyelven nincs szükség. Java nyelvben nincsenek külön objektumok és mutatók, míg C++-ban többször is használnunk kell őket. Javában nincsen lehetőség a felhasználói operátorok definiálására. C++-ban egy függvény argumentumainak adhatunk alapértelmezett értéket, míg ilyen lehetőség Javában nincsen. A Java és a C++ utasításkészlete nagyon hasonlít. Java-ban megkülönböztetünk osztályokat és interfészeket. Osztályok körében a Java egyszeres öröklődést támogat, és nincs megkülönböztetés a `private`, `public` vagy `protected` öröklődés között.

## 20.2. Python

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

Itt a kijelölt oldalakból egy 1 oldalas élmény-olvasónaplóra gondoltam.

DRAFT

## **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](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.