

# **Univerzális programozás**

**Írd meg a saját programozás tankönyvedet!**

**Bátfai, Norbert, Debreceni Egyetem <batfai.norbert@inf.unideb.hu>**

---

# Univerzális programozás: Írd meg a saját programozás tankönyvedet!

írta Bátfai, Norbert

kiadás dátuma 2019

Szerzői jog © 2019 Dr. Bátfai Norbert

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

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

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

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

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

---

---

# 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]

---

# Köszönetnyilvánítás

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.

---

# Tartalom

Előszó .....	vi
Hogyan forgasd .....	vi
Milyen nyelven nyomjuk? .....	vi
Hogyan nyomjuk? .....	vi
I. Bevezetés .....	1
1. Vízió .....	3
Mi a programozás? .....	3
Milyen doksikat olvassak el? .....	3
Milyen filmeket nézzek meg? .....	3
II. Tematikus feladatok .....	4
2. Helló, Turing! .....	7
Végtelen ciklus .....	7
Lefagyott, nem fagyott, akkor most mi van? .....	7
Változók értékének felcserélése .....	8
Labdapattogás .....	8
Szóhossz és a Linus Torvalds féle BogoMIPS .....	9
Helló, Google! .....	9
100 éves a Brun tétel .....	9
A Monty Hall probléma .....	9
3. Helló, Chomsky! .....	10
Decimálisból unárisba átváltó Turing gép .....	10
Az $a^n b^n c^n$ nyelv nem környezetfüggetlen .....	10
Hivatkozási nyelv .....	10
Saját lexikális elemző .....	10
l33t.l .....	10
A források olvasása .....	11
Logikus .....	11
Deklaráció .....	11
4. Helló, Caesar! .....	13
int *** háromszögmátrix .....	13
C EXOR titkosító .....	13
Java EXOR titkosító .....	13
C EXOR törő .....	13
Neurális OR, AND és EXOR kapu .....	13
Hiba-visszaterjesztéses perceptron .....	13
5. Helló, Mandelbrot! .....	15
A Mandelbrot halmaz .....	15
A Mandelbrot halmaz a <code>std::complex</code> osztállyal .....	15
Biomorfok .....	15
A Mandelbrot halmaz CUDA megvalósítása .....	15
Mandelbrot nagyító és utazó C++ nyelven .....	15
Mandelbrot nagyító és utazó Java nyelven .....	15
6. Helló, Welch! .....	16
Első osztályom .....	16
LZW .....	16
Fabejárás .....	16
Tag a gyökér .....	16
Mutató a gyökér .....	16
Mozgató szemantika .....	16
7. Helló, Conway! .....	18
Hangyaszimulációk .....	18
Java életjáték .....	18
Qt C++ életjáték .....	18
BrainB Benchmark .....	18
8. Helló, Schwarzenegger! .....	19

Szoftmax Py MNIST .....	19
Szoftmax R MNIST .....	19
Mély MNIST .....	19
Deep dream .....	19
Robotpszichológia .....	19
9. Helló, Chaitin! .....	20
Iteratív és rekurzív faktoriális Lisp-ben .....	20
Weizenbaum Eliza programja .....	20
Gimp Scheme Script-fu: króm effekt .....	20
Gimp Scheme Script-fu: név mandala .....	20
Lambda .....	20
Omega .....	20
III. Második felvonás .....	21
10. Java könyv .....	23
A következő bekezdésekben a Java 2 útikalauz programozóknak 5.0 első kötetének élménybeszámolóját olvashatják: .....	23
11. Helló, Arroway! .....	26
OO szemlélet .....	26
„Gagyí” .....	28
Yoda .....	28
Kódolás from scratch .....	29
12. Helló, Liskov! .....	31
Liskov helyettesítés sértése .....	31
Szülő-gyerek .....	32
Ciklomatikus komplexitás .....	33
EPAM: Interfész evolúció Java-ban .....	34
13. Helló, Mandelbrot! .....	36
Reverse engineering UML osztálydiagram .....	36
Forward engineering UML osztálydiagram .....	36
Egy esettan .....	39
BPMN .....	39
14. Helló, Chomsky! .....	41
Pasziográfia Rapszódia OpenGL full screen vizualizáció .....	41
<i>Full screen</i> .....	41
<i>l334d1c46</i> .....	42
EPAM: Bináris keresés és Buborék rendezés implementálása .....	44
15. Helló, Stroustup! .....	48
JDK osztályok .....	48
Másoló-mozgató szemantika .....	49
Változó argumentumszámú ctor .....	50
Összefoglaló .....	52
16. Helló, Gödel! .....	54
Gengszterek .....	54
STL map érték szerinti rendezése .....	54
Alternatív Tabella rendezése .....	55
GIMP Scheme hack .....	55
Irodalomjegyzék .....	59

---

## Az ábrák listája

12.1. PiBBP.java ciklomatikus komplexitása .....	34
13.1. LZWBinFa UML .....	36
13.2. BPMN .....	40
14.1. Teljes Képernyő .....	42
16.1. beállítás .....	56
16.2. Kész mandala .....	58

---

# Előszó

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

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

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

## Hogyan forgasd

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

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

## Milyen nyelven nyomjuk?

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

## Hogyan nyomjuk?

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

Ha megvannak a könyv DocBook XML forrásai, akkor az alább látható **make** parancs ellenőrzi, hogy „jól formázottak” és „érvényesek-e” ezek az XML források, majd elkészíti a dlatex 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
dlatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
=====
Stripping NS from DocBook 5/NG document.
Processing stripped document.
```



```
Image 'dbratex' 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

---

---

# Tartalom

1. Vízió .....	3
Mi a programozás? .....	3
Milyen doksikat olvassak el? .....	3
Milyen filmeket nézzek meg? .....	3

---

# 1. fejezet - Vízió

## Mi a programozás?

## Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [KERNIGHANRITCHIE]
- [BMECPP]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 [https://web.archive.org/web/20181230041359if\_/http://www.open-std.org/jtc1/sc22/wg14/www/abq/c17\_updated\_proposed\_fdis.pdf] kódcsipeteiből is.

## Milyen filmeket nézzek meg?

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

---

## II. rész - Tematikus feladatok

### **Bátf41 Haxor Stream**

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

---

# Tartalom

2. Helló, Turing! .....	7
Végtelen ciklus .....	7
Lefagyott, nem fagyott, akkor most mi van? .....	7
Változók értékének felcserélése .....	8
Labdapattogás .....	8
Szóhossz és a Linus Torvalds féle BogoMIPS .....	9
Helló, Google! .....	9
100 éves a Brun tétel .....	9
A Monty Hall probléma .....	9
3. Helló, Chomsky! .....	10
Decimálisból unárisba átváltó Turing gép .....	10
Az $a^n b^n c^n$ nyelv nem környezetfüggetlen .....	10
Hivatkozási nyelv .....	10
Saját lexikális elemző .....	10
l33t.l .....	10
A források olvasása .....	11
Logikus .....	11
Deklaráció .....	11
4. Helló, Caesar! .....	13
int *** háromszögmátrix .....	13
C EXOR titkosító .....	13
Java EXOR titkosító .....	13
C EXOR törő .....	13
Neurális OR, AND és EXOR kapu .....	13
Hiba-visszaterjesztéses perceptron .....	13
5. Helló, Mandelbrot! .....	15
A Mandelbrot halmaz .....	15
A Mandelbrot halmaz a <code>std::complex</code> osztállyal .....	15
Biomorfok .....	15
A Mandelbrot halmaz CUDA megvalósítása .....	15
Mandelbrot nagyító és utazó C++ nyelven .....	15
Mandelbrot nagyító és utazó Java nyelven .....	15
6. Helló, Welch! .....	16
Első osztályom .....	16
LZW .....	16
Fabejárás .....	16
Tag a gyökér .....	16
Mutató a gyökér .....	16
Mozgató szemantika .....	16
7. Helló, Conway! .....	18
Hangyaszimulációk .....	18
Java életjáték .....	18
Qt C++ életjáték .....	18
BrainB Benchmark .....	18
8. Helló, Schwarzenegger! .....	19
Szoftmax Py MNIST .....	19
Szoftmax R MNIST .....	19
Mély MNIST .....	19
Deep dream .....	19
Robotpszichológia .....	19
9. Helló, Chaitin! .....	20
Iteratív és rekurzív faktoriális Lisp-ben .....	20
Weizenbaum Eliza programja .....	20
Gimp Scheme Script-fu: króm effekt .....	20
Gimp Scheme Script-fu: név mandala .....	20

Lambda .....	20
Omega .....	20

---

## 2. fejezet - Helló, Turing!

### Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 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 épülő 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? Sehogyz, mert ilyen Lefagy függvényt, azaz a T100 program nem is létezik.

Tanulságok, tapasztalatok, magyarázat...

## Változók értékének felcserélése

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Labdapattogás

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/Primek\\_R](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R)

## A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/03/erdos\\_pal\\_mit\\_keresett\\_a\\_nagykonyvben\\_a\\_monty\\_hall-paradoxon\\_kapcsan](https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan)

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

Tanulságok, tapasztalatok, magyarázat...

---

## 3. fejezet - Helló, Chomsky!

### Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## A források olvasása

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

```
if(signal(SIGINT, jelkezeslo)==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 jelkezeslo 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, jelkezeslo);

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 forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

## Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

Tanulságok, tapasztalatok, magyarázat...

## 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
- függvényt mutató 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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

---

## 4. fejezet - Helló, Caesar!

### int \*\*\* háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 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...

### Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

---

## 5. fejezet - Helló, Mandelbrot!

### A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

### Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

### A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

### Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

### Mandelbrot nagyító és utazó Java nyelven



---

## 6. fejezet - Helló, Welch!

### 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 elspájzold és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltérve kiszámolt szám.

Megoldás videó:

Megoldás forrása:

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!

### LZW

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

Megoldás videó:

Megoldás forrása:

### Fabejárás

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

Megoldás videó:

Megoldás forrása:

### 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:

Megoldás forrása:

### Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

### Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

---

## 7. fejezet - Helló, Conway!

### 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:

Tanulságok, tapasztalatok, magyarázat...

### Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

---

# 8. fejezet - Helló, Schwarzenegger!

## Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

---

## 9. fejezet - Helló, Chaitin!

### Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

### Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

### 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...

### 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...

## Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## Omega

Megoldás videó:

Megoldás forrása:

---

## 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.

---

# Tartalom

10. Java könyv .....	23
A következő bekezdésekben a Java 2 útikalauz programozóknak 5.0 első kötetének élménybeszámolóját olvashatják: .....	23
11. Helló, Arroway! .....	26
OO szemlélet .....	26
„Gagyí” .....	28
Yoda .....	28
Kódolás from scratch .....	29
12. Helló, Liskov! .....	31
Liskov helyettesítés sértése .....	31
Szülő-gyerek .....	32
Ciklomatikus komplexitás .....	33
EPAM: Interfész evolúció Java-ban .....	34
13. Helló, Mandelbrot! .....	36
Reverse engineering UML osztálydiagram .....	36
Forward engineering UML osztálydiagram .....	36
Egy esettan .....	39
BPMN .....	39
14. Helló, Chomsky! .....	41
Pasziográfia Rapszódia OpenGL full screen vizualizáció .....	41
<i>Full screen</i> .....	41
<i>l334d1c46</i> .....	42
EPAM: Bináris keresés és Buborék rendezés implementálása .....	44
15. Helló, Stroustup! .....	48
JDK osztályok .....	48
Másoló-mozgató szemantika .....	49
Változó argumentumszámú ctor .....	50
Összefoglaló .....	52
16. Helló, Gödel! .....	54
Gengszterek .....	54
STL map érték szerinti rendezése .....	54
Alternatív Tabella rendezése .....	55
GIMP Scheme hack .....	55

---

# 10. fejezet - Java könyv

## A következő bekezdésekben a Java 2 útikalauz programozóknak 5.0 első kötetének élménybeszámolóját olvashatják:

A Java programozási nyelv hasonló a C nyelvhez, azonban újabb és több lehetőséggel áll rendelkezésünkre. Nyelvünk objektumorientált, ami annyit tesz, hogy egy program objektumokra, azon belül osztályokra van felbontva, míg a C és C++ nyelvek eljárásorientáltak. Az osztályokon belül használunk változókat, illetve metódusokat. Ez utóbbi felel azért, hogy milyen műveletet hajtunk végre az adatokon. Míg a C-nél mutatókat használunk, a Java-ban már referenciákat. Változókat épp úgy adhatunk meg mint C vagy C++ nyelveknél, először a változó típusa (int, string, double, stb...), majd „=” operátor és az érték amit meg szeretnénk adni. Amikor Java kódot írunk érdemes megjegyzéseket hozzá tenni a programunkhoz (épp úgy mint bármelyik másik nyelvnél), ezt a „/” egyetlen sor esetében, ha több sorban szeretnénk megjegyzést írni akkor a „/\*” nyitó részként, és a „\*/” záró részként szolgáló jelöléssel tehetjük meg. Osztályokat a class kulcsszóval vezetünk be, amiken belül tetszőleges sorrendben vehetjük fel annak metódusait, és adatait. Fontos eldönteni, hogy kinek a számára legyen látható az adott elem. Így pl.: egy public osztály mindenki számára látható, míg private csak az osztályon belülre vonatkozik. Osztályon belül a new operátorral hozhatunk létre újabb objektumokat. Ügyelnünk kell a kivételkezelésre (bizonyos programok esetében pl.: nullával való osztás), ezt a try és catch segítségével oldhatjuk meg. A Java sok kétdimenziós grafikai elemmel rendelkezik, ilyen pl az awt, amivel saját magunk készíthetünk két dimenziós alakzatokat.

A legelterjedtebb karakterkészlet az ASCII kódrendszer, ami 8 biten ábrázolja a karaktereinket. Ebből azonban sok nemzet által gyakran használt karakterek hiányoznak (nálunk pl.: ú és ő betűk). Az ASCII-től kétszer annyi biten ábrázoló Unicode azonban már rendelkezik a számunkra szükséges karakterekkel. Java-ban az azonosítók betűvel kell, hogy kezdődjenek és betűvel vagy számmal kell folytatódniuk. Tetszőleges hosszúak lehetnek, de nem tartalmazhatják a nyelv kulcsszavait (pl.: int, boolean, for, return, stb...). Az egyszerű típusokat és objektumokat literálokkal inicializáljuk, ezek pedig: objektum, logikai érték, egész szám, lebegőpontos szám, karakter, szöveg, és végül osztály. Változódeklarációnál kell hogy megadjunk egyetlen típust és legalább egy változónevet. Tömböket hasonlóan adunk meg mint C-ben, itt viszont a tömb igazi típus lesz, nem pedig mutató. Ha struktúrákat szeretnénk elérni, azokra ponttal hivatkozhatunk.

Az utasítások két fajtája: kifejezés-utasítás és deklaráció utasítás. Elágazásokat az if szerkezettel, összetettebbeket a switch-el oldhatunk meg. A Java-ban a következő ciklusok ismertek: előltesztelő (while), hátultesztelő (do, while), léptető (for, while), és bejáró (for). Egy ciklusból a break parancs segítségével léphetünk ki, ilyenkor a program automatikusan fut tovább a következő kódsorra. Hasonlóan a continue parancs is ezt teszi, azonban ezt már használhatjuk metódusokban és inicializáló blokkokban. Visszatérési értéket C-hez hasonlóan a return parancs fog nekünk adni, viszont a Java-ból kikerült a goto utasítás, a biztonság növelése érdekében.

Java-ban az osztályok a legkisebb önálló egységek. Egy osztály egy adott tulajdonságú halmaz elemeit tartalmazza (pl.: emberek, tárgyak). Itt figyelniük kell, hogy egy adott osztálynak csak egy célja legyen, és ne terheljük túl az osztályokat. Osztályon belüli változódeklarációnál ügyelnünk kell arra, hogy mekkora legyen a változó „hatósugara”, hogy más osztályok ne használhassák egymás változóit. Osztályon belüli metódusokat metódusdefiníciók árják le, amiknek fej és törzs része van. Metódushíváskor nem elég csak a nevet megadni, annak paramétereit is meg kell. Figyelniük kell a metódustúlterhelésre is. Ugyanis egy osztályon belül lehet több azonos nevű metódus is. Ilyenkor a fordítóprogram a paraméterek száma és típusa alapján választja ki a számunkra megfelelő metódust. Objektumokat a new operátorral hozunk létre, a new után pedig meg kell adnunk, hogy melyik osztályt példányosítjuk. Példányosításkor memóriát foglalunk le, ahol az objektum változói lesznek tárolva, és visszaadja a kezdőcímét. Programozáskor problémát szokott jelenteni, hogy az objektumok feleslegesen foglalnak memóriát, mert egyszerűen nem hivatkozik rájuk semmi. Java-ban ez nem



jelent problémát, mert a rendszer automatikus felszabadítja azokat helyeket, amire nincs hivatkozás. Más nyelveknél ezt a programozónak magának kell megtennie. A metódusokra nagyban hasonlítanak a konstruktorok. Ezek végrehajtása a példányosításkor azonnal megtörténik. Egy konstruktornak meg kell egyeznie az osztály nevével. Csak példányosításon keresztül meghívhatóak. Néhány esetben nem árt értesülni egy-egy objektum megsemmisüléséről. Ez osztály szinten a `classFinalize`, metódus szinten `finalize` metódus (ez egy destruktorként fogja nekünk megmondani). Polimorfizmusnak nevezzük amikor egy változó olyan módon van deklarálva, hogy a leszármazottak is hivatkozhatnak rá. Polimorfizmusnál megkülönböztetünk statikus és dinamikus változókat. A statikus változó típusa változatlan, még a dinamikus változó is a változó által hivatkozott tényleges típusra.

Az osztályok mellett másik fontos építőköve a nyelvnek az interfész, ami olyan referenciatípus, amely absztrakt metódusok deklarációjának és konstans értékeknek az összege. Valódi használata az implementációján keresztül történik, így egy absztrakt program konkréttá válik. Interfészek között is van öröklődés. Deklarálni az interface kulcsszóval lehet, hasonlóan mint az osztályoknál a `class`. Deklarációval egy vázat hozunk létre, és ebben implementációkat helyezünk el, amik osztályokat helyettesítenek. Fontos a fordítási hiba elkerülése végett, hogy publikus implementációkat adjunk az interfész összes metódusához. Az interfész egy újabb referenciatípus. Úgy használjuk mint egy osztályt. Az interfészek öröklődését kiterjesztésnek nevezzük, és az `extend` paranccsal tudjuk megvalósítani. Ez szintén úgy működik mint az osztályoknál. Konstansokat a következő képpen vezetünk be: módosítók, konstans típusa, azonosító, inicializáló kifejezés és `;` zárjuk le, mint a legtöbb sorunkat. Deklarálásukhoz a módosítókat kell használnunk. Az interfészek használata az őket használó osztályok hívják meg. Ez csak akkor tud megvalósulni, ha az összes implementáló metódus szignatúrája és visszatérési értéke megegyezik az interfészével, különben `error`.

Amikor programozunk törekedjünk a letisztult és átlátható formára. Révén, hogy objektumorientált nyelvről van szó, a programrészeket megfelelően tagoljuk. Erre lesznek segítségünkre a csomagok, amik tartalmazzák a fejlesztői környezetet és a kódunkat. Úgy is mondhatjuk, hogy a csomag a hozzáférési kategóriák használatának az eszköze. A csomagoknál az öröklődési szint hierarchikusan van jelen. Itt a gyermeket alsomagnak hívjuk. Tartalmuk lehetnek típusok vagy további alsomagok, melyeket ponttal választunk el. Itt egy fájlstruktúrát kell elképzeljünk, ahol nincs szorosabb kapcsolat az ős felé, mint bármelyik másik csomag iránt. Célja a programkód átláthatóságának növelése, vagyis a programozó munkájának könnyítése. Csomagokat fájlrendszerben (JDK) vagy adatbázisban tárolunk. A csomagok kódja a fordítási egységekben található. Deklarálásuk `package` kulcsszóval történik, amik a kód legelején kell hogy legyenek, már deklaráció után nem állhatnak. Továbbá lehet `import`deklarációt alkalmazni aminek három fajtája a következő: egyszerű `import`-deklaráció, igény szerinti `import`-deklaráció, statikus `import`-deklaráció (ez utóbbi Java 5-től kezdve van jelen). Ezek segítik a típusok használatát. Csomagot létrehozni úgy tudunk, hogy a elkészítjük a hozzá tartozó fordítási egységeket, ezeket deklaráljuk aszerint, hogy melyik csomaghoz tartoznak. Majd megírjuk a `import`- és `import`deklarációkat a fordítási egységekben. Ügyeljünk a megfelelő névválasztásra, ugyanis csoportos projekt esetén egy rosszul megválasztott csomagnév sok fejfájást okozhat. Erre egy jó módszer ha az egész elérési útvárat adjuk meg névként.

Sokszor esünk bele abba a hibába, hogy azt hisszük mindent jól kódoltunk le, mégis hibás a program. Azt tudjuk, hogy bizonyos részek lefutnak, na de hol keressük a bugot? Erre használjuk a kivételkezelést (Java exception), ami Java-ban a hasonlóan történik mint C-ben. Ez egy olyan különleges helyzet, hogy valahol hibát dob a program, de mégis lefut, a hibás részt pedig egy kivételobjektum fogja kezelni, pontosabb információt kapva a hibáról. Ezt nevezzük kivétel kiváltásnak (throwing exception), és szintén három fajtája fordul elő: rendellenes dolog történt (pl.: 0-val való osztás), `throw` utasítás kivált egy kivételt (csomagokban vagy a kódban), aszinkron hiba (párhuzamos futásnál egyik szál megszakad). A kivétel kezelése oly módon történik, hogy a kivételkezelő megkeresi azt a helyet, ahol a kiváltott kivétel kezelése megtörténhet. A kiváltott típusnak meg kell egyeznie a kivétel típusával, vagy őse az osztályhierarchiában. A kivételkezelő dönti el, hogy mekkora az a blokk amit kivált. Ezeket az utasításokat kapcsolószerű jelek közé teszi. A kivételt el is kell kapnunk valahogy, erre a célra a `try` lesz segítségünkre. Ehhez azonban hozzá kell tennünk még közvetlenül utána egy `catch` vagy `finally` blokkot, különben `error`ot kapunk.

A generikusok a Java 5. újdonságai, feladatuk az osztályok vagy eljárások típusokkal való paraméterezése, ez által sokkal egyszerűbb megoldani a feladatokat. Céljai az egyszerűség, biztonság növelése és a hatékonyság. Típushelyettesítő forma a wildcard ami az alábbi formák egyike lehet: `?`, `?`

extends T, ? super T, itt T egy típust jelöl. Ügyelnünk kell, hogy a következők kivételével minden referencia-típus lehet generikus: névtelen belső osztály, kivétel típusok (oka, hogy futás közben a generikusok nincsenek tárolva, a throwable osztály márpedig futási időben él), felsorolási típusok (mivel statikus jellegűek ugyan az a probléma mint a kivételnél). A generikusok típustöreléssel foglalkoznak, és objektumok terén végrehajtott ezért object típus lesz. Figyelnünk kell az instanceof kifejezésre, ugyanis nem mindig működik generikusokra, a típustörlés következményeként.

Gyűjteményeknek nevezzük azokat a típuskonstrukciós eszközöket, melyek feladata az egy vagy több típusba tartozó objektumok példányainak memóriában történő összefoglaló jellegű tárolása, lekérdezése és manipulálása. A legtöbb programozási nyelvben megtalálhatók. Általában objektumok és tömbök mutatóinak vagy referenciáinak összekapcsolása. A Java 2-ben jelent meg, inkább a praktikus jellegű, mintsem a teljesség. Alapvető gyűjtemények és leképezések osztályai szorosan összefüggnek, külön-külön nem használjuk őket. Az interfészeikhez több reprezentációs osztály is tartozhat, ezek közötti váltás egyszerű. A gyűjtemény (collection) egyik leszármazottja a halmaz (set), feladata a halmaz adattípus megvalósítása. Megszorításokat tesz a, nem tesz kiterjesztést a gyűjteményhez képest. Örökli a műveleteket, amik kifejezhetők halmazműveletekként. Másik leszármazottja a lista (list), és a lista adattípust valósítja meg. Itt az elemek duplikáltan is szerepelhetnek, és számítja a sorrendjüket. A lista ad kiterjesztést néhány gyűjtemény interfészhez. Néhány örökölt műveletet is megváltoztat, ez duplikálás és sorrend miatt fontos (pl.: a remove művelete csak az első elemet távolítja el a listából). A sor adatszerkezet lényege, hogy az első objektumot ami bekerül, azt vegyük ki elsőnek. A leképezés adatszerkezet kulcs-érték párokat tárol, a Map interfészen keresztül. Egy kulcshoz egy érték tartozik.

Egy szoftverrendszer felépítését az alapján kell megtervezni, hogy az adatokon milyen tevékenységek lesznek elvégezve. A legáltalánosabb a felülről lefelé haladó eljárás, hiszen gyors, könnyen áttekinthető. Azonban nem minden esetben ez a legmegfelelőbb eljárás. Ezt mindig a programozónak kell eldöntenie, hogy mi lesz a legmegfelelőbb a programhoz. Objektumorientált nyelvként először mindig objektumokra bontjuk a feladatot. Ez reprezentálja a modellezendő világ egységeit. Az objektumokat osztályokba soroljuk azok tulajdonságai alapján (pl.: élőlény osztálynak van ember, állat és növény objektuma). Ezen belül lesznek az alosztályok (állatoknál pl.: emlős, madár, stb...). Amikor egy objektumot nem tudunk egyértelműen besorolni egyetlen osztályba sem, olyankor kapcsolatokkal kötjük őket osztályokhoz. Minden osztály meghatározható egy másik osztály leszüktetésével vagy kiterjesztésével. Így jön létre az örökös és gyermek kapcsolat az osztályok között. Minden gyermek osztály örökli az ősoosztály tulajdonságait, fordítva nem igaz. A programtervezés legfőbb célja, hogy jól átlátható kódot készítsünk, erre az objektum orientált nyelv az egyik legjobb módszer a fentebb említett példák miatt. Szintén három lépésre oszthatjuk: analízis (a probléma körvonalazása), rendszertervezés (a körünket részekre osztjuk), osztálytervezés (a rendszert további osztályokra bontjuk). Ha mindezzel megvagyunk akkor megkezdődhet az implementálás.

A C++ jelölésrendszeréből sok mindent átvett a Java. Egy C++-ban jártas programozónak nem okoz majd nagy nehézséget megtanulni a Java szintaktikai szabályait, mert nagy átfedés van a két nyelv között, mivel a Java szintaxisa a C és C++-ból fejlődött ki. Felépítésükben viszont eltérést látunk, mivel a Java egy teljesen "objektumorientált nyelv. Ez annyit takar, hogy minden változó és metódus egy osztálynak a része. Amikor megírunk egy programot, aztán fordítani és futtatni szeretnénk, akkor a C++ fordítója a saját kódunkat gépi kóddá konvertálja. A gépi kódot már értelmezni tudja az eszközünk és így a programunk már futtatható. Ennek a módszernek az a hátránya, hogy egy másik számítógépen csak azonos platform mellett lesz garantált ugyanaz az eredmény. Ezt másnéven platform-függőségnek is nevezzük. Java-ban ha a típusokról van szó, akkor primitív és nem primitív típusokra oszthatjuk fel őket. A primitív típusok egy konkrét értéket tárolnak. Ezeket a primitív típusokat helyettesíthetjük csomagoló osztályokkal is. Minden primitív adattípus rendelkezik egy hasonló (vagy vele megegyező) névű, de nagy betűs csomagoló osztállyal. Például az int-et helyettesíthetjük az Integer-rel. Osztályokat úgy mint C++-ban, Java-ban is a class teremti meg. A C++-ban különféle beépített vagy akár saját osztályok eléréséhez header fájlokat include-olása szükséges. Ez Java-ban viszont a header fájlok hiányában nem lehetséges. Helyette csomagokat használunk, amelyeket az import-tal érhetünk el, vagy simán megadhatjuk a csomag elérési útját.

---

# 11. fejezet - Helló, Arroway!

## OO szemlélet

Feladatunk egy polártranszformációs normális generátor megvalósítása Java nyelven. Tekintsünk el a matematikai háttér megértéséről. Számunkra inkább maga a program működése a lényeges.

A polártranszformációs generátor Java-ban:

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

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
        if(nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do{
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2* u1 -1;
                v2 = 2* u2 -1;
                w = v1*v1 + v2*v2;
            } while (w>1);

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

Az elején láthatunk egy `nincsTarolt` nevű boolean típusú változót. Ez fogja meghatározni a későbbi sorokban, hogy a program generál-e számot, vagy pedig az eltárolt értéket adja vissza. Ez a változó alapértéke igaz, mivel az eddig nincs semmink, amit eltárolhatnánk. Amennyiben nincs tárolt számunk, a `kovetkezo()` függvény generál nekünk két random számot, az egyiket visszaadja, a másikat pedig eltárolja egy `tarolt` nevű double-ben. Ekkor a `nincsTarolt` hamis értéket kap. Ha már rendelkezünk tárolt értékkel, akkor a függvényünk egyszerűen kiírja a tárolt számot és a `nincsTarolt` igaz értéket kap.

Ha a fenti kódot összevetjük az `OpenJDK Random.java` állományában találhatóval, láthatjuk, hogy lényegében megegyezik a két forráskód.

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

Fellelhető ugyan egy apró eltérés a felcserélt if és else ágakban, de mivel haveNextNextGaussian alapértéke hamis, így az eredmény ugyanaz lesz.

A C++ megvalósítás is nagyban hasonlít a Java verziójához, de mint azt már tudjuk, a Java szintaktikája a C++-tól vett át sok mindent

```
#include "polargen.h"
double
PolarGen::kovetkezo ()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1 = std::rand () / (RAND_MAX + 1.0);
            u2 = std::rand () / (RAND_MAX + 1.0);
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);
        double r = std::sqrt ((-2 * std::log (w)) / w);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
        return r * v1;
    }
    else
    {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
}

#include <iostream>
#include "polargen.h"
```

```

int
main (int argc, char **argv)
{
    PolarGen pg;
    for (int i = 0; i < 10; ++i)
        std::cout << pg.kovetkezo () << std::endl;
    return 0;
}

```

## „Gagyí”

Az ismert formális tesztkérdéstípusra adj a szokásosnál (miszerint *x*, *t* az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írd Java példaprogramot mely egyszer végtelen ciklus, más *x*, *t* értékekkel meg nem! A példát építsd a JDK `Integer.java` forrására<sup>3</sup>, hogy a 128-nál inkluzív objektum példányokat poolozza!

Vizsgáljuk meg a két alábbi kódcsipet eredményét.

```

public static void main (String[] args)
{
    Integer x = 127;
    Integer t = 127;
    System.out.println (x);
    System.out.println (t);
    while (x <= t && x >= t && t != x);
}

```

```

public static void main (String[] args)
{
    Integer x = 128;
    Integer t = 128;
    System.out.println (x);
    System.out.println (t);
    while (x <= t && x >= t && t != x);
}

```

Egy olyan ciklust szeretnénk létrehozni, ahol két számnak egyszerre kell kisebbnek (és egyenlőnek), nagyobbak (és egyenlőnek), és nem egyenlőnek lenni. Ezt egy kis trükkkel megtudjuk valósítani, ha ismerjük az `Integer` memóriafoglalási elvét. Az első példában nem következik be végtelen ciklus, mivel nem teljesültek a feltételek. Ellemben a második példában végtelen ciklusba torkollik a program, mert teljesül minden feltétel.

De hogyan teljesülhet egyszerre mind a három feltétel? A válasz abban rejlik, hogy Java-ban az `Integer` osztályban 127-ig pool-olva vannak a pozitív egészek értékei, tehát ha 127 vagy attól kisebb értéket adunk az objektumunk, akkor ugyanarra a memóriarészre hivatkozik. Lényeges, hogy a `!=` operátor nem az objektumok értékeit fogja összehasonlítani, hanem referenciákat. Mivel a 128 már nem pool-olt érték, ezért különböző memóriacímmel fog rendelkezni a két objektum.

## Yoda

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

A Yoda condition tulajdonképpen annyi, hogy feltétel sorrendje megfordul, és a bal oldalra fog elhelyezkedni a konstans, jobb oldalra pedig a változó kerül. Használtaával elkerülhetőek olyan hibák, mint például az értékadó operátor(=) használata az összehasonlító helyet(==).

```
String myString = null;
if (myString.equals("foobar")) { /* ... */ }
```

Ezzel a kóddal NullPointerException-be ütközünk, mivel null értéket nem hasonlíthatunk konstashoz.

Nézzük meg a Yoda conditions alkalmazásával:

```
String myString = null;
if ("foobar".equals(myString)) { /* ... */ }
```

Így viszont lefut a kódunk, mivel string konstans hasonlíthatunk null értékhez. Ebben a példában hamis értéket ad a kifejezés, ahogy az várható.

## Kódolás from scratch

A BBP, azaz Bailey-Borwein-Plouffe algoritmus segítségével a Pi hexadecimális számjegyeit tudjuk meghatározni.

Vessünk egy pillantást a kódra.

```
public class PiBBP {
    public PiBBP(int d) {

        double d16Pi = 0.0d;

        double d16S1t = d16Sj(d, 1);
        double d16S4t = d16Sj(d, 4);
        double d16S5t = d16Sj(d, 5);
        double d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

        Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};

        while(d16Pi != 0.0d) {

            int jegy = (int)StrictMath.floor(16.0d*d16Pi);

            if(jegy<10)
                sb.append(jegy);
            else
                sb.append(hexaJegyek[jegy-10]);

            d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
        }

        d16PiHexaJegyek = sb.toString();
    }
}
```

Az algoritmus alapján a  $\{16^d \text{ Pi}\} = \{4 \cdot \{16^d \text{ S1}\} - 2 \cdot \{16^d \text{ S4}\} - \{16^d \text{ S5}\} - \{16^d \text{ S6}\}$  kiszámolásával kezdi a program. A  $\{\}$  jelölés a törtész jelölésére szolgál. Mivel 16-os számrendszerben dolgozunk, és a Java nem tartalmaz hexa számjegyeket, így meg kell adnunk őket egy `hexaJegyek[]` tömbben.

```
public double d16Sj(int d, int j) {
    double d16Sj = 0.0d;

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

Itt a  $\{16^d \text{ Sj}\}$  részt számoljuk ki. A hexajegyek kiszámolása  $(d+1)$ -edik elemtől kezdődik.

```
public long n16modk(int n, int k) {
    int t = 1;
    while(t <= n)
        t *= 2;

    long r = 1;

    while(true) {
        if(n >= t) {
            r = (16*r) % k;
            n = n - t;
        }

        t = t/2;

        if(t < 1)
            break;

        r = (r*r) % k;
    }

    return r;
}
```

A  $16^n \bmod k$  kiszámítása bináris hatványozással történik, ahol  $n$  a kitevő,  $k$  a modulus.

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

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

A `toString()` függvény a kiszámolt hexajegyek visszaadására szolgál. A `main` függvényben példányosítunk egy `PiBBP` objektmot. Ebben a példában  $d=1000000$ , ezért 1000001. hexajegytől írjuk ki a  $\text{Pi}$ -t.

---

# 12. fejezet - Helló, Liskov!

## Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés.

A Liskov elv szerint minden osztály legyen helyettesíthető leszármozzataival anélkül, hogy ez befolyásolná a program helyes működését. Tehát ha T leszármazottja S, akkor behelyettesíthetjük T helyére S-t, és ezután is helyes eredmény kell kapnunk.

Nézzünk egy példát az elvet sértő programra:

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

A Madar lett a T osztályunk. Az S osztályok pedig a Sas és Pingvin lettek. Mivel a Madar osztály tartalmazza a repülést, így az abból származtatott Sas és Pingvin osztályok is képesek leszenk rá a program szerint, de tudjuk, hogy a pingvin röpképtelen, ezért ez a valóságban nem következhet be.

Nézzünk egy megoldást a madaras példára a Liskov elv betartása mellett:

```
class Madar {};
class Program {
public:
    void fgv ( Madar &madar ) {
    }
};
class RepuloMadar : public Madar {
public:
    virtual void repul() {};
};
```



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

Láthatjuk, hogy a Madar osztálynak egy új leszármazottja is van, a RepuloMadar, és ebben található a repülés. A Pingvin ugyanúgy a Madar-ból származik, azonban a Sas-t a RepuloMadar-ból származtatjuk, így csak az lesz képes a repülésre.

Végül az elv teljesülése Java-ban. A gondoltamenet ugyanaz, a szintaktika kicsit eltér.

```
class Madar {}
class Program {
    public void fgv ( Madar madar ) {}
}
class RepuloMadar extends Madar {
    public void repul() {}
}
class Sas extends RepuloMadar {}
class Pingvin extends Madar {}
public class figyel{

    public static void main ( String[] args )
    {
        Program program = new Program();
        Madar madar = new Madar();
        program.fgv(madar);

        Sas sas = new Sas();
        program.fgv(sas);
        sas.repul();
        Pingvin pingvin = new Pingvin();
        program.fgv(pingvin);
    }
}
```

## Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetők!

Ebben a feladatban be kell mutatnunk azt, hogy az ős mutatóin vagy referenciáin keresztül csak az ős metódusait érhetjük el.

Nézzünk egy rövid példakódot:

```

#include <iostream>
using namespace std;
class Szulo
{
};
class Gyerek: public Szulo{
public: void kiir(){
cout<<"gyerek"<<endl;
}
};
int main()
{
Szulo* sz= new Gyerek();
cout << sz->kiir() <<endl;
}

```

A Szulo gyermekosztályaként létrehozunk egy Gyerek osztályt. Ebben a leszármazottban definiálunk egy kiir nevű eljárást. Ez a kód nem fog lefordulni helyesen, hiszen a kiir() metódus az őszosztályban nem található meg.

Ha ezt a kódot Java-ban próbáljuk megvalósítani szintén problémába ütközünk, ugyanis nem hívható meg a leszármazott metódusa az őson keresztül. Errort kapunk fordításnál, jogosan. Íme a kód Java-ban:

```

public class Szulo{
    public class Gyerek extends Szulo{
        public void kiir(){System.out.println("Gyerek");}
    }
    public void main(String[] args)
    {
        Szulo szulo = new Gyerek();
        System.out.println(szulo.kiir());
    }
}

```

## Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalomtekintetében 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 főlát)!

A ciklomatikus komplexitás vagy másnéven McCabe-komplexitás (alkotója után kapta ezt a nevet) a program forrásának komplexitását határozza meg egy konkrét számértékkal. A számítása gráfelméletre alapul. A képlet így néz ki:  $M = E - N + 2P$ , amelyben E a gráf éleinek száma, N a csúcsok száma és P az összefüggő komponensek száma.

Akkor nézzük is meg egy tetszőleges forráskód ciklomatikus komplexitását. Ez most az előzőekben már többször használt PiBBP algoritmus Java verziója lesz. Én a forrás ciklomatikus komplexitását a: Lizard [<http://www.lizard.ws/>] nevű open source elemzővel számoltattam ki. Tetszőleges kódot másolhatunk be a bal oldalra, majd kiválasztjuk a programnyelvet, és futtatjuk rajta az elemzőt. Az eredményt a jobb oldalon kapjuk meg:

## 12.1. ábra - PiBBP.java ciklomatikus komplexitása

<div>File Type <b>.java</b> Token Count <b>397</b> NLOC <b>52</b></div>				
Function Name	NLOC	Complexity	Token #	Parameter #
PiBBP::PiBBP	20	3	192	
PiBBP::d16Sj	6	2	70	
PiBBP::n16modk	17	5	87	
PiBBP::toString	3	1	8	
PiBBP::main	3	1	22	

## 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?

Vizsgáljuk meg az alábbi két kódot:

```
package com.epam.training;

public interface InterfaceA {

    default public void method() {
        System.out.println("InterfaceA.method()");
    }

}
```

A Java 7, vagy korábbi verziókban nem lehetett implementálni az interfészekben default metódust. Ehhez képest a Java 8-ban újításnak hatott default metódusra azért volt szükség, hogy könnyebb legyen biztosítani a visszafele kompatibilitást az ebben a verzióban megjelent újításokkal kapcsolatban. Lássuk milyen problémát vetett is ez fel:

```
package com.epam.training;

public interface InterfaceB {

    default public void method() {
        System.out.println("InterfaceB.method()");
    }

}
```

A példákban jól látható, hogy ha több interfész is rendelkezik ugyan azzal a default metódussal (és adnak hozzá default implementációt), akkor fellép egy többszörös öröklődési probléma. Erre kénytelenek vagyunk megadni egy explicit módon felüldefiniáló osztályt, ahol implementáljuk a két interfészt, és eldöntjük, hogy még is melyiket szeretnénk használni.

```
public class Implementation implements InterfaceA, InterfaceB {  
  
    @Override  
    public void method() {  
        InterfaceA.super.method();  
    }  
  
}
```

# 13. fejezet - Helló, Mandelbrot!

## Reverse engineering UML osztálydiagram

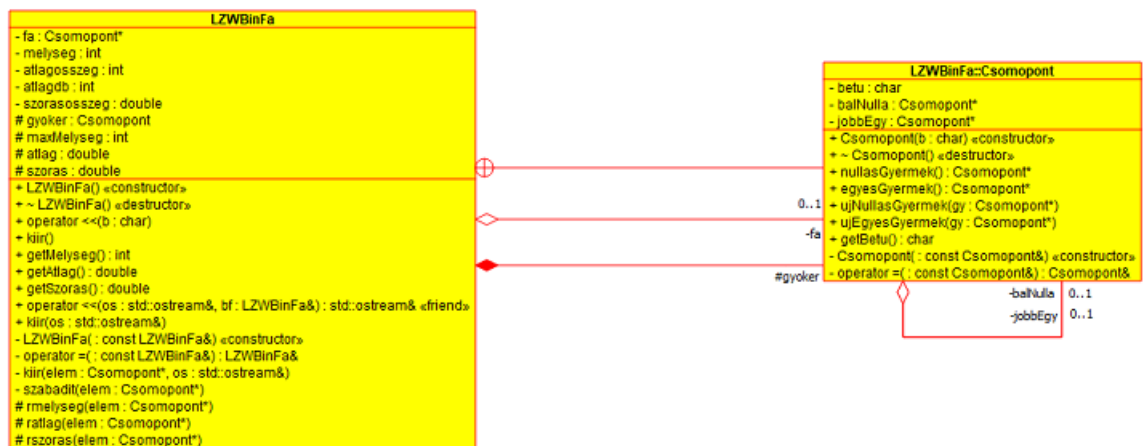
UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: [https://youtu.be/Td\\_nIERIEOs](https://youtu.be/Td_nIERIEOs)

[https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1\\_6.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_6.pdf) (28-32. oldal) [[https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1\\_6.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_6.pdf)]

Az UML (Unified Modeling Language) egy általános célokra szánt modellező nyelv. Számunkra abban lesz hasznos, hogy ábrázoljuk az objektum orientált szemléletben írt kódunkat. Először is szükségünk lesz egy erre alkalmas programra. A választás az Umbrello-ra esett, mivel ez egy ingyenes UML diagram készítő.

A kódimportáláshoz a varázslóra lesz szükségünk. A `z3a7.cpp` ábrázolása a feladatunk, ezért a C++ nyelvet kell kiválasztanunk. Miután kiválasztottuk a nyelvet, a kiterjesztések részt bővítjük ki a `.cpp`-vel, hogy importálhassuk a kódunkat. Ezután a fástruktúra nézetből behúzzuk az osztályokat. Majd a fenti diasor alapján módosítjuk egy kicsit.

### 13.1. ábra - LZWBinFa UML



Végül egy pár szót az aggregáció és kompozíció kapcsolatáról. Ezek az egész-rész viszonyoknak alapvető formái. Az aggregációt üres rombuszal ábrázoljuk, a kompozíciót pedig teli rombuszal. Az előbbi hozzátartozik valamihez, de önmagában is létezhet, amíg az utóbbi csak valami részeként létezhet.

## Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generálunk belőle forrást!

A feladathoz ismét az LZWBinfa C++ verzióját fogom felhasználni. Az Umbrello nem csak kódból készült diagram generálásra alkalmas, hanem fordítva is felhasználható. Ezúttal egy diagramból készítünk kódot. A kapott eredmény nem lesz futtatásra kész, ez csak egy váza lesz a programnak, az osztályok és függvények deklarációit tartalmazza. A Code Generation Wizard használatával ez könnyen kivitelezhető. Lássuk a kapott eredményt:

```
#ifndef LZWBINF_H
```

```

#define LZWBINF_H
#include <string>
class LZWBinFa
{
public:
    LZWBinFa ();
    virtual ~LZWBinFa ();
    void operator_ (char b)
    {
    }
    void kiir ()
    {
    }
    int getMelyseg ()
    {
    }
    double getAtlag ()
    {
    }
    double getSzoras ()
    {
    }
    std::ostream& operator_ (std::ostream& os, LZWBinFa& bf)
    {
    }
    void kiir (std::ostream& os)
    {
    }
protected:
    Csomopont gyoker;
    int maxMelyseg;
    double atlag;
    double szoras;
public:
protected:
public:
    void setGyoker (Csomopont new_var)    {
        gyoker = new_var;
    }
    Csomopont getGyoker ()    {
        return gyoker;
    }
    void setMaxMelyseg (int new_var)    {
        maxMelyseg = new_var;
    }
    int getMaxMelyseg ()    {
        return maxMelyseg;
    }
    void setAtlag (double new_var)    {
        atlag = new_var;
    }
    double getAtlag ()    {
        return atlag;
    }
    void setSzoras (double new_var)    {
        szoras = new_var;
    }
    double getSzoras ()    {

```

```

        return szoras;
    }
protected:
    void rmelyseg (Csomopont* elem)
    {
    }
    void ratlag (Csomopont* elem)
    {
    }
    void rszoras (Csomopont* elem)
    {
    }
private:
    Csomopont* fa;
    int melyseg;
    int atlagosszeg;
    int atlagdb;
    double szorasosszeg;
public:
private:
public:
    void setFa (Csomopont* new_var)    {
        fa = new_var;
    }
    Csomopont* getFa ()    {
        return fa;
    }
    void setMelyseg (int new_var)    {
        melyseg = new_var;
    }
    int getMelyseg ()    {
        return melyseg;
    }
    void setAtlagosszeg (int new_var)    {
        atlagosszeg = new_var;
    }
    int getAtlagosszeg ()    {
        return atlagosszeg;
    }
    void setAtlagdb (int new_var)    {
        atlagdb = new_var;
    }
    int getAtlagdb ()    {
        return atlagdb;
    }
    void setSzorasosszeg (double new_var)    {
        szorasosszeg = new_var;
    }
    double getSzasosszeg ()    {
        return szorasosszeg;
    }
private:
    LZWBinFa (const LZWBinFa& )
    {
    }
    LZWBinFa& operator_ (const LZWBinFa& )
    {
    }

```

```
void kiir (Csomopont* elem, std::ostream& os)
{
}
void szabadit (Csomopont* elem)
{
}
void initAttributes () ;
};
#endif
```

Ahogy azt láthatjuk, az UML diagramoknak nagy hasznát vehetjük már tervezési fázisban is. A kapott kód jól szemlélteti a program szerkezeti felépítését.

## Egy esettan

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

Ennek a feladatnak is szerves részét képezik az UML diagramok, ezért ismerkedjünk meg az alapvető dolgokkal az előzőekben látottak alapján. UML-ben téglalpokkal ábrázoljuk az osztályokat. Ezeket három részre bonthatjuk tovább. Fentről lefelé az első rész az osztály neve, majd ezt követik annak attribútumai, és végül a változókat láthatjuk. Függvények és változók előtt található a láthatóságot jelölő '+'(public), '-'(privát) és '#'(védtett).

Az esettanulmány a következő. Adott egy számítógép-alkatrész és számítógép-konfiguráció értékesítésével foglalkozó kereskedés. El kell készíteni egy olyan programot, amellyel megvalósítható a kereskedés alkatrészeinek és konfigurációinak a nyilvántartása. A szofvernek támogatnia kell a termékek állományból való betöltést, képernyőre kiírását, az állományba kiírását és a rugalmas árképzést.

Először is egy keretrendszer kialakítására van szükség osztálykönyvtárak által, ami alapszinten támogatja a termékcsaládokat. Majd erre a keretre építve egy alkatrészszelektáló alkalmazás elkészítése vár ránk. A végső cél az, hogy a meglévő keretrendszerre változtatása nélkül más termékcsaládok támogatása is megvalósuljon.

A keretrendszer kialakításához kelleni fog egy `Product` osztály, amelyben termékek általános kezelését implementáljuk. Ez az osztály tárolja az általános jellemzőket, pl.: terméknév, beszerzési ár, beszerzési dátum. A különböző termékek a `Product` leszármazottai lesznek. A típusnév és típuskód a `Product` osztályban virtuális függvényként definiáljuk, mivel ezeknek a leszármazott osztályokban egyediek.

Az összetett termékek (például a számítógép-konfigurációk) kezelése a `CompositeProduct` osztályban valósul meg, amit szintén a `Product`-ból származtatunk. Vektorban fogjuk tárolni az alkatrészeket, amit az `AddPart` metódus segítségével bővíthetünk.

A termékek nyilvántartására létrehozunk a `ProductInventory` osztályt. Arra szolgál, hogy betöltse a termékek listáját, a betöltött termékeklistákat tárolja a memóriában, a memóriából adatfolyamba írás és a formázások megjelenítése. A lista bővítésére az `AddProduct` szolgál.

Mivel a keretrendszer osztálykönyvtár, ezért az általunk bevezetett termékosztályokat nem ismeri, így szükségünk van egy osztályra, amivel kiküszöbölhető ez a probléma. A megoldást számunkra egy `ProductFactory` osztály lesz. Ebben lesz egy `ReadAndCreateProduct` függvénye, amit a keretben definiálunk, de az általa meghívott `CreateProduct` függvény már virtuális.

## BPMN

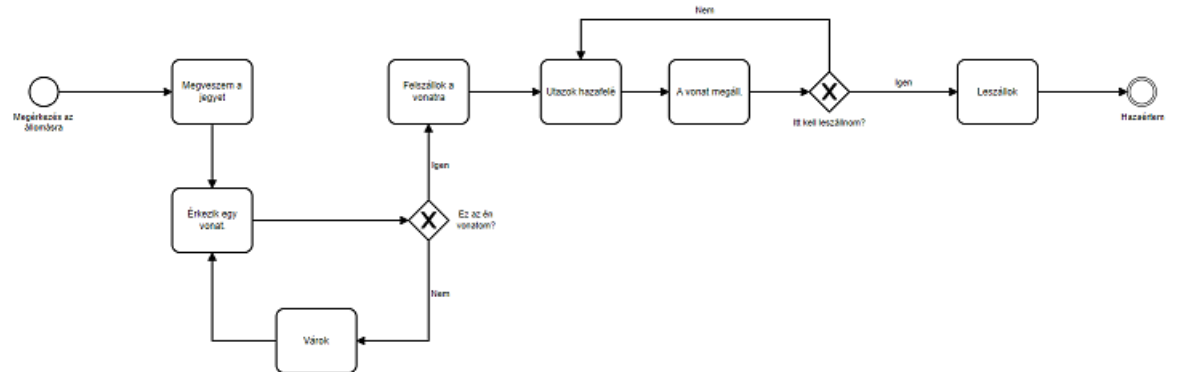
Rajzoljunk le egy tevékenységet BPMN-ben!

A BPMN (Business Process Model and Notification) üzleti folyamatok modellezéséhez használatos folyamatábra jelölés. A célja az üzleti folyamatok egységes grafikus ábrázolása. Ilyen folyamatábrákat mi is tudunk készíteni. Én böngészőből csináltam a <https://demo.bpmn.io/> oldalon.



Ahhoz, hogy mi is ábrákat készítsünk először ismerjük meg az alapvető jelöléseket. Az eseményeket(event) körrel jelöljük: kezdeti esemény simar körrel, és a végső esemény pedig vastag körrel. A tevékenységeket(activity) lekerekített sarkú téglalap jelöli. Az átjáróknak(gateway) trapéz a jele. Az elemek közti összekötést(association) nyilakkal jelölik.

### 13.2. ábra - BPMN



---

# 14. fejezet - Helló, Chomsky!

## Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis\_prel\_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

A Paszigráfia Rapszódia célja, hogy lehetővé tegye homunkulusz és mesterséges homunkulusz közti kommunikációt. OpenGL segítségével valósul meg a vizualizáció. Szükségünk lesz lib-boost és freeglut3 csomagokra. Ezeket könnyedén letölthetjük az alábbi két paranccsal

```
sudo apt-get install libboost-all-dev
sudo apt-get install freeglut3-dev
```

Az a feladat, hogy miután felélesztettük a programot, apróbb változtatásokat is hajtsunk végre benne. A színek változtatásához a glColor3f függvényt kell módosítanunk.

```
glColor3f ( 0.0, 1.0f, 0.0f );
```

Ez például zöld színt eredményez. Ha más színt szeretnénk megadni, akkor néhány alapvető színt megtalálhatunk például itt [<https://pemavirtualhub.wordpress.com/2016/06/20/opengl-color-codes/>].

```
void keyboard ( int key, int x, int y )
{
    if ( key == GLUT_KEY_UP ) {
        cubeLetters[index].rotx -= 5.0;
    } else if ( key == GLUT_KEY_DOWN ) {
        cubeLetters[index].rotx += 5.0;
    }
}
```

Ez pedig felcseréli a fel és le gombok funkcióját, tehát másik irányba fordulnak, mint eddig.

A program fordítása és futtatása:

```
g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut
./para 3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:0:1:0:1:0:1:0:2:2:0:1:1:1:3:
```

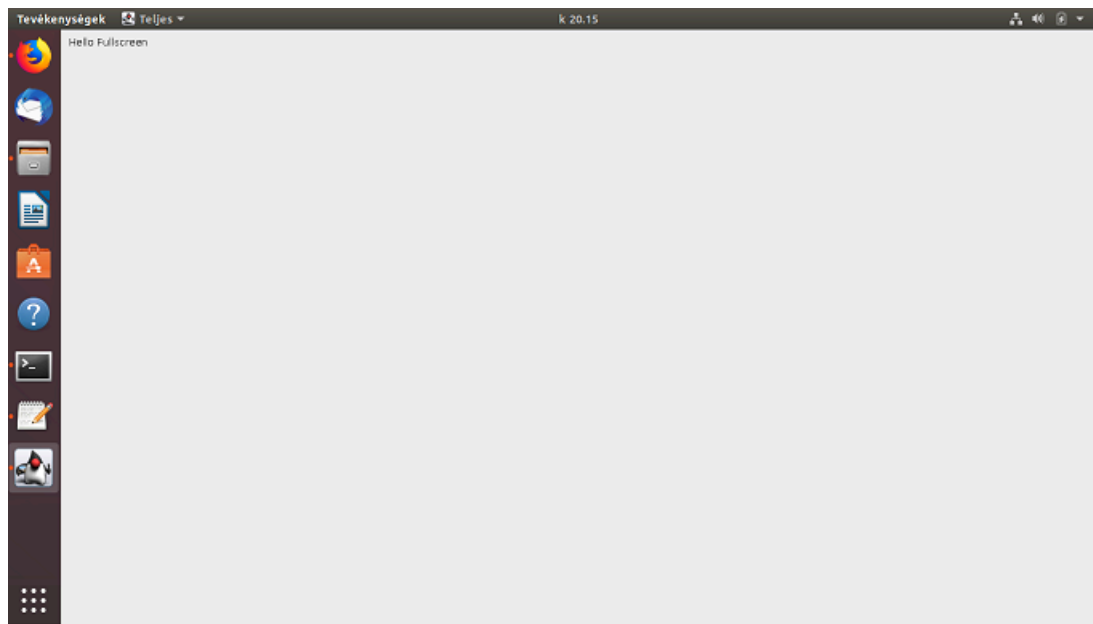
## Full screen

Készítsünk egy teljes képernyős Java programot! Tipp: [https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanito-javat/ch03.html#labirintus\\_jatek](https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanito-javat/ch03.html#labirintus_jatek) [[https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanito-javat/ch03.html#labirintus\\_jatek](https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanito-javat/ch03.html#labirintus_jatek)]

Ebben a feladatban egy egyszerű kiírást fogunk kitenni teljes képernyőre. Néhány YouTube videó és kis böngészés után ez [<http://www.java2s.com/Code/JavaAPI/java.awt/GraphicsdrawStringStringstrintxinty.htm>] alapján készítettem el egy nagyon egyszerű változatot.

A végeredmény nem lett túl látványos, de a célnak megfelel:

## 14.1. ábra - Teljes Képernyő



A programot a szükséges osztályok improtálásával kezdjük.

```
import java.awt.*;  
import javax.swing.JFrame;  
import javax.swing.JPanel;
```

Létrehozuk a `Teljes` osztályunkat a `Jpanel` leszármazottjaként.

```
public class Teljes extends JPanel {
```

A `paint` metódusban található `drawString` függvény lesz felelős kiírásért.

```
public void paint(Graphics g) {  
    g.drawString("Hello Fullscreen", 200, 200);  
}
```

Elértünk a `main`-hez. Az elején példányosítunk egy `JFrame`-t. A többi sorba pedig magyarázó kommenteket tettem.

```
public static void main(String[] args) {  
    JFrame frame = new JFrame("Teljes"); //Zárójelek között a programnév  
    frame.getContentPane().add(new Teljes());  
    frame.setExtendedState(JFrame.MAXIMIZED_BOTH); //Itt lesz teljes képernyős  
    frame.setUndecorated(true); //etávolítja a címsort  
    frame.setVisible(true);  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Kilépéskor bezárja a  
}  
}
```

# I334d1c46

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

A leet nyelvvel már volt dolgunk korábban a Progl-es könyv során. Lényege, hogy a betűket (vagy akár számokat is) kicseréljük hozzájuk hasonló karakterre vagy karakterláncra. A feladatban látható Java forrás alapját ez a kód [<https://codehackersblog.blogspot.com/2015/06/leet-speak-converter-with-java-code.html>] adta. Ezen egy kicsit változtattam, mivel tartalmazott számunkra most lényegtelen részeket.

Itt is a szükséges könyvtárak improtálásával kezdünk.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
```

Deklaráljuk a toLeetCode függvényt. Létrehozunk egy Patter típust, aminek az elfogadott szövegmintát adjuk meg értékül, ami jelen esetben kis- és nagybetűkből állhat. A StringBuilder a végeredmény tárolására szolgál.

```
class L33tConvertor {

    private String toLeetCode(String str) {
        Pattern pattern = Pattern.compile("[^a-zA-Z]");
        StringBuilder result = new StringBuilder();
```

Ezt követően egy HashMap-ot csinálunk, ami kulcs érték párok tárolására alkalmas. Ez fogja tárolni a helyettesítéseket, amiket megadunk.

```
HashMap<Character, String> map = new HashMap<Character, String>();
    map.put('A', "@");
    map.put('B', "8");
    map.put('C', "©");
    map.put('D', "d");
    map.put('E', "€");
    map.put('F', "f");
    map.put('G', "6");
    map.put('H', "#");
    map.put('I', "!");
    map.put('J', "¿");
    map.put('K', "X");
    map.put('L', "£");
    map.put('M', "M");
    map.put('N', "r");
    map.put('O', "0");
    map.put('P', "p");
    map.put('Q', "0");
    map.put('R', "®");
    map.put('S', "$");
    map.put('T', "7");
    map.put('U', "µ");
    map.put('V', "v");
    map.put('W', "w");
    map.put('X', "%");
    map.put('Y', "Ÿ");
    map.put('Z', "z");
```

Maga az átalakítás egy for ciklusban valósul meg. Ez a ciklus bejárja a szöveget egyesével és nagybetűre alakítja őket, és megnézi, hogy van-e helyettesítése a vizsgált karakternek a mapunkban. Ha nem talál helyettesítést, akkor visszaadja az adott karaktert, egyébként pedig a map.get visszaadja a leet változatot. Az végeredményt stringre konvertálva kapjuk vissza.

```
for (int i = 0; i < str.length(); i++) {
    char key = Character.toUpperCase(str.charAt(i));
    Matcher matcher = pattern.matcher(Character.toString(key));
    if (matcher.find()) {
        result.append(key);
        result.append(' ');
    } else {
        result.append(map.get(key));
        result.append(' ');
    }
}
return result.toString();
```

A main függvényben példányosítunk egy L33tConvertor osztályt. Egy BufferedReader fog beolvasni a standard bementről. A leetWord tárolja az általunk beírt szöveget, és erre hívjuk meg a toLeetCode függvényt.

```
public static void main(String[] args) throws IOException {
    L33tConvertor obj = new L33tConvertor();
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String leetWord;

    System.out.println("\nEnter the English Words :-");
    leetWord = br.readLine();
    String leet = obj.toLeetCode(leetWord);
    System.out.println("The 1337 Code is :- " + leet);
}
```

## 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)

Mivel halmazokkal fogunk foglalkozni, fontos, hogy importáljuk is azt: `java.util.Arrays`. Majd vegyük fel a "főosztályt" `IntegerCollection`, ahol lesz maga a halmazunk, egy index (mely utal majd a halmaz i-edik elemeire), egy logikai változós `sorted` (hogy tudjuk rendezve vannak-e már a számok), és kell egy `size` is, ami a halmaz számosságát adja vissza.

```
package com.epam.training;
```

```
import java.util.Arrays;

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;
    }
}
```

A következő rész felel az új elem felvételéért, ahol először megnézzük, hogy nem-e haladtuk meg a halmaz méretét. Amennyiben igen úgy error-t dob a prog-i `The collection is full`, de ha nem ütközünk semmilyen problémába, úgy jöhet az új elem, de ekkor már a rendezést hamisra kell állítsuk.

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

A második kritérium szerint el kell tudni döntenie, hogy a felvett érték szerepel-e már a kis halmazunkban. Tehát a `contains` osztály pontosan ezt fogja tenni. Végig megy az elemeken és egyesével összehasonlít. A végén pedig hamis értéket ad vissza.

```
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;  
}
```

Végül pedig a rendezés. Itt a két egymásba ágyazott for ciklus az egymás mellett álló elemeket nézi meg, és rendezi őket növekvő sorrendbe.

```
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) + "];"  
}  
}
```

A kiírás és értéfelvétel a mian.java-ban történik, ahol először felvesszük az IntegerCollection elemeit és tároljuk őket. Majd a felvehető elemeket. Végül kiíratjuk őket tárolt és növekvő sorrendben, illetve a felvett elemeket:

```
package com.epam.training;
```

```
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));  
    }  
}
```



---

# 15. fejezet - Helló, Stroustup!

## JDK osztályok

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

A feladatunk az lesz, hogy kilistázzuk a JDK osztályokat. Ehhez kicsomagolt src állományra lesz szükségünk. Eerre az src-re ráengedünk egy programot, ami bejárja az állományt és kiírja a .java kiterjesztésű fájlokat. A feladat alapjául egy fénykard nevezetű program fog szolgálni. Ennek a programnak a `read_acts` metódusa megkereste a .props fájlokat. Mi is ebből indulunk ki.

Szükségünk lesz a boost könyvtárra, így mindenekelőtt telepítsük.

```
sudo apt-get install libboost-all-dev
```

A programot a szükséges könyvtárak inkludálásával kezdjük, mint mindig.

```
#include <stdio.h>
#include <string>
#include <vector>
#include <iostream>
#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>

using namespace std;
void read_classes ( boost::filesystem::path path, vector<string> & classes )
{
    if ( is_regular_file ( path ) )
    {
        string ext ( ".java" );
        if ( !ext.compare ( boost::filesystem::extension ( path ) ) )
        {
            classes.push_back(path.string());
        }
    }
    else if ( is_directory ( path ) )
        for ( boost::filesystem::directory_entry & entry : boost::filesystem::directory_iterator( path ) )
            read_classes ( entry.path(), classes );
}
```

Az eljárás ami most a `read_classes` nevet kapta, a fénykardban található `read_acts` módosított változata. Az if ágat módosítottuk, mint az látható. A függvény egyébként annyi csinál, hogy megnézi a paraméterül kapott útvonal egy fájl-e. Ha egy egyszerű fájl és .java a kiterjesztése, akkor pusholjuk az osztályokat tartalmazó vektorunkba. Ha a kapott útvonal egy mappa, akkor ismét meghívjuk rá a `read_classes`-t. Ezzel a módszerrel az egész könyvtárt bejárhatjuk rekurzívan, akárhány alkönyvtár is legyen benne.

```
vector<string> getClass (string path)
{
    vector <string> classes;
    boost::filesystem::path root (path);
```

```

        read_classes(root, classes);

    return classes;
}
int main(int argc, char const * argv[])
{
    boost::filesystem::path full_path(boost::filesystem::current_path());
    string root = full_path.string() + "/" + "src";

    vector<string> classes = getClass ( root );

    for(const auto & i : classes)
    {
        cout << i << endl;
    }

    cout << classes.size();

    return 0;
}

```

## 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!

Konstruktorról van szó, ezért a neve ugyanaz, mint az osztály neve.

```

LZWBInFa::LZWBInFa (const LZWBInFa & forras){
    std::cout << "Copy ctor" << std::endl;
    gyoker = new Csomopont('/');
    gyoker->ujEgyesGyermeke(masol(forras.gyoker->egyesGyermeke(), forras.fa));
    gyoker->ujNullasGyermeke(masol(forras.gyoker->nullasGyermeke(), forras.fa));
    if (forras.fa == forras.gyoker){
        fa = gyoker;
    }
}

```

Látható, hogy a konstruktor az LZWBInFa konstans referenciáját várja paraméterül. Ilyenkor nem hívódik meg az alap konstruktor, ezért a gyoker mutatónak értéket kell adni. Ezután létrehozuk a gyermekeket a ujNullasGyermeke és a ujEgyesGyermeke függvények felhasználásával a gyökkértől indulva. A masol függvénynek két paramétert kell megadni: a forrás gyökerének gyermeke és a famutatót.

```

Csomopont* LZWBInFa::masol (Csomopont* elem, Csomopont* regi_fa){
    Csomopont* ujelem = nullptr;

    if (elem != nullptr){
        ujelem = new Csomopont (elem->getBetu());

        ujelem -> ujEgyesGyermeke(masol(elem->egyesGyermeke(), regi_fa));
        ujelem -> ujNullasGyermeke(masol(elem->nullasGyermeke(), regi_fa));

        if (regi_fa == elem){
            fa = ujelem;
        }
    }
}

```

```

    }

    return ujelem;
}

```

A masol egy Csomópont pointert ad vissza. A későbbiekben átadni kívánt csomópont létrehozása az első. Utána megnézzük, hogy az eredeti fa csomópontja nullpointer-e. Ha nem, akkor az ujelem az eredeti csomópontához tartozó memóriacímet. Ezután újra meghívjuk a függvényt, hogy az ujelem gyermekeit is létrehozzuk. Ez folytatódik, amíg a fe végére nem érünk. Ha a régi fához tartozó mutató a masol-nak átadott elemre mutat, akkor az új fának a mutatóját ez elem alapján létrehozott csomópontra állítjuk be.

Ez volt a másoló konstruktor. Nézzük meg a másoló értékadást aminek alapja szintén a masol.

```

LZWBinFa & LZWBinFa::operator= (const LZWBinFa & forras){
    gyoker->ujEgyesGyermek(masol(forras.gyoker->egyGyermek(), forras.fa));
    gyoker->ujNullasGyermek(masol(forras.gyoker->nullasGyermek(), forras.fa));

    if (forras.fa == forras.gyoker){
        fa = gyoker;
    }
    return *this;
}

```

Ami elsőnek szemet szúrhat az az, hogy ennek van visszatérési értéke a konstruktorral szemben. A visszatérési érték egy LZWBinFa referencia. Visszaadunk egy mutatót az egyenlőségjel bal oldalán szereplő objektumról.

A mozgató konstruktor paraméterül jobbtértek referenciát kap, ezt jelzi a dupla and jel.

```

LZWBinFa::LZWBinFa (LZWBinFa&& forras)
{
    gyoker = nullptr;
    *this = std::move(forras); //ezzel kényszerítjük ki, hogy a mozgató értékadást
}

```

Lényegében felcseréljük a két fa gyökérmutatójának értékét, ezért az új fának gyökeréből nullpointert csinálunk és meghívjuk a mozgató értékadást. Ezt az std::move függvénnyel tesszük. A this egy inicializált objektumot jelöl, így nem a mozgató konstruktor, hanem azz értékadás hívódik meg.

```

LZWBinFa& LZWBinFa::operator= (LZWBinFa&& forras)
{
    std::cout<<"Move assignment ctor\n";
    std::swap(gyoker, forras.gyoker);
    return *this;
}

```

Az értékek cseréjét az std::swap végzi el. Ezzel teljes a mozgató. Mivel az új fa mutatója nullpointer volt, az eredeti fáé is az lett.

## Változó argumentumszámú ctor

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

A feladat során megismerjük, hogy készíthető változó argumentumszámú konstruktor. C++-ban a ...-tal jelöljük, hogy bármennyi paramétere lehet a konstruktornak. Az összes paraméter eléréséhez a `cstdarg` headerre van szükségünk. Nézzük a konstruktort

```
Perceptron ( int nof, ... )
{
    n_layers = nof;

    units = new double*[n_layers];
    n_units = new int[n_layers];

    va_list vap;

    va_start ( vap, nof );

    for ( int i {0}; i < n_layers; ++i )
    {
        n_units[i] = va_arg ( vap, int );

        if ( i )
            units[i] = new double [n_units[i]];
    }

    va_end ( vap );

    ...
}
```

A `Perceptron` osztály osztály konstruktorában található a `nof` nevű paraméter. A maradék paraméter eléréséhez pedig a `va_list vap`-ot deklaráljuk, ebben a változó paramétereket tároljuk. Értékét a `va_start` inicializálja. A `nof` azt fogja számon tartani, hogy hány paramétert adtunk meg. Ennek paraméterül meg kell adni egy `va_list`-et, amelybe betölti az argumentumokat és egy argumentumot, ahonnan a betöltés kezdődjön. A `va_arg` azt aktuális paramétert adja vissza nekünk. Minden meghívásra módosítja a `vap`-ot úgy, hogy a már visszaadott értéket követő értéket adja vissza. Ez hasznos lesz a paraméter lista bejárásánál. A `va_end` után nem tudjuk használni többet az argumentum listát.

Kisebbségi módosításokat hajtunk végre a `Perceptron` osztályon. A feladat szerint nem értéket fog visszaadni, hanem egy vektort, aminek a mérete a bemeneti képtől függ. Majd manipuláljuk az eredeti képet és egy `output.png`-be elmentjük.

Példányosításnál az utolsó paraméter `size` lesz 1 helyett.

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

Ezután meg kell oldani az osztályon belül az előbbi változás kezelését.

```
std::vector< double > operator() ( double image [] )
{
    std::cout << "std" << std::endl;
    units[0] = image;
    for ( int i {1}; i < n_layers; ++i )
    {
#ifdef CUDA_PRCPS
        cuda_layer ( i, n_units, units, weights );
    }
    #else
        #pragma omp parallel for
    #endif
    }
```

```

for ( int j = 0; j < n_units[i]; ++j )
{
units[i][j] = 0.0;
for ( int k = 0; k < n_units[i-1]; ++k )
{
units[i][j] += weights[i-1][j][k] * units[i-1][k];
}
units[i][j] = sigmoid ( units[i][j] );
}
#endif
}
std::vector< double > result_vector;

for (int i = 0; i < units[n_layers - 1]; ++i){
result_vector.push_back(sigmoid( units[n_layers - 1][i]));
}
return result_vector;
}

```

Az `operator()` túlterhelés során egy vektort hozunk létre, később ezt adjuk vissza. A `result_vector`-ba pusholjuk a `sigmoid` függvény által kiszámolt értékeket. A vektor értékei alapján módosítjuk az eredeti kép pixelehez rendelt zöld értéket. Végül létrehozuk a képünket a `write` segítségével.

...

```

std::vector< double > value = (*p) (image);
for (int i = 0; i<png_image.get_width(); ++i)
for (int j = 0; j<png_image.get_height(); ++j){
png_image[i][j].green = value[i*png_image.get_width() + j];
}

png_image.write("output.png");

```

...

## Összefoglaló

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutató „esszé” szöveget!

A másoló és a mozgató konstruktor.

Amikor egy osztályt példányosítunk C++-ban a konstruktor hívódik meg. Ha nincs definiálva konstruktor, akkor a fordító az alapértelmezettet fogja használni. A konstruktorok feladata a az objektum inicializálása, azaz itt adunk az egyes tagváltozóknak értéket, tagfüggvényeket hívni. Már megfigyelhettük, hogy a konstruktorok nevei megegyeznek az osztályok neveivel. Ennek párja a destruktork, ami az objektum törlésénél hívódik meg, és felszabadítja a már nem használt memóriaterületet. Azonban ezen a kettőn kívül szükségünk van másoló és mozgató konstruktorokra is.

Az utólag említett két konstruktor segítségével ugyancsak az objektum inicializálását valósíthatjuk meg, de egy másik azonos osztályú objektum alapján. Ha nem írunk másoló/mozgató konstruktort, akkor a fordító az alapértelmezettet fogja meghívni, ez viszont váratlan problémákhoz vezethetnek.

A másoló konstruktor, ahogy a neve is mutatja, objektumot másol le, és az alapján készít egy másikat. Paraméterként egy őt tartalmazó osztállyal azonos osztályú objektumreferenciát vár. Két fajta másolásról beszélhetünk: sekélyről és mélyről. A sekély másolás annyit takar, hogy a paraméterként megadott objektumra csak egy másik mutatót hozunk létre. Ebből adódóan nagy hátránya, hogy ugyanaz a memóriaterület, tehát ha az egyiket módosítjuk, akkor az hatással van a másikra is. A mély másolás viszont már egy különálló objektumot hoz létre, amely ugyanúgy megegyezik a paraméterül kapott elmmel, de különbözik a memóriacímük.

A mozgató konstruktor a többi konstruktorhoz hasonlóan inicializálást hajt végre, és itt is egy másik objektum az alapja. Amíg másolásnál az érintett objektumok megmaradnak, addig mozgításnál az eredeti objektum eltűnik. A mozgató konstruktor ismertető jegye, hogy jobbérték referenciát vár paraméterül(dupla and jel) ahogy már írtam korábban). Hogy ez hívódjon meg,a `std::move` függvényt kell meghívunk. ez valójában a kapott objektum jobbérték referenciájával tér vissza.

Az LZWBinFa programban a mozgató konstruktor a mozgató értékadásra van alapozva. Az alap elgondolás az, hogy létrehozni kívánt fa gyökerét nullpointer-ré tesszük, majd megcseréljük a forrás- és célfa mutatóját, ezzel valósítva meg a mozgatót. Fontos, hogy mikor hívódik meg konstruktor és mikor az értékadás. Ha még nem inicializált objektumot szeretnénk másolással vagy mozgatóval létrehozni, akkor a konstruktor hívódik meg. Azonban ha már inicializált objektumba szeretnénk értéket mozgatóval/másolni, abban az esetben már értékadás megy végbe. Ennek tudatában szerepel a mozgató konstruktorban az alábbi sor:

```
*this = std::move(forras);
```

Meghívódik a mozgató értékadás, ahol a mutatók értékének cseréjét hajtjuk végre.

---

# 16. fejezet - Helló, Gödel!

## Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban

[https://youtu.be/DL6iQwPx1Yw\(8:05-től\)](https://youtu.be/DL6iQwPx1Yw(8:05-től)) [<https://youtu.be/DL6iQwPx1Yw>]

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

Ahogy látható, a `sort` függvény három paraméterrel rendelkezik. Az első két paraméter azt adja meg, hogy mettől meddig menjen a függvény. Ebben az esetben a `gangsters` nevű vektor elejétől fog menni a végéig, amit a `begin()` és `end()` függvények segítségével adunk meg. A harmadik paraméter egy függvény ami a rendezés alapjául fog szolgálni. Ez egy név nélküli, úgynevezett lambda függvény.

A []-ból tudjuk, hogy egy lambda kifejezésről van szó, a kódcsipetben ez tartalmaz egy mutatót az aktuális objektumra és a `cop` nevű objektumot. A kerek zárójelek között két `Gangster` osztályú objektumot találunk paraméterként. Ez a két paraméter kerül majd összehasonlításra. A függvény törzsében pedig egy logikai típusú változóval térünk vissza, az alapján, hogy melyik objektum van közelebb a rendőrhöz.

A `sort` tehát az eredményezi végső soron, hogy a gengszterek rendezve lesznek rendőrtől való távolságuk szerint növekvő sorrendben.

## STL map érték szerinti rendezése

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

A `map` kulcs és érték párokat tárolnak és alapból a kulcs szerint növekvően van rendezve.

```
std::vector<std::pair<std::string, int>> sort_map ( std::map<std::string, int> m ) {
    {
        std::vector<std::pair<std::string, int>> ordered;

        for ( auto & i : m ) {
            if ( i.second ) {
                std::pair<std::string, int> p {i.first, i.second};
                ordered.push_back ( p );
            }
        }

        std::sort (
            std::begin ( ordered ), std::end ( ordered ),
            [=] ( auto && p1, auto && p2 ) {
                return p1.second > p2.second;
            }
        );
    }
}
```

```
        return ordered;
    }
}
```

A tárolt párokat egy `pair` vektorhoz rendeljük, majd rendezzük őket. A `pair` lehetővé teszi, az értékek párban való tárolását. A korábban már megismert lambda kifejezés lesz segítségünkre a rendezés során. Az értékek alapján csökkenő sorrendbe rendezzük át a vektort. Amivel eddig még nem találkoztunk, az az `auto` típus, ami arra szolgál, hogy a fordító automatikusan megtalálja a megfelelő típust egy adott változóhoz.

## Alternatív Tabella rendezése

Mutassuk be a [https://progpater.blog.hu/2011/03/11/alternativ\\_tabella](https://progpater.blog.hu/2011/03/11/alternativ_tabella) a programban a `java.lang` `Interface Comparable<T>` szerepét!

Az Alternatív Tabella a magyar labdarúgó bajnokság alternatív rangsorát tárolja. Nem csak azt vizsgálja, hogy nyert, vesztett vagy döntetlent játszott-e egy csapat, hanem azt is, hogy ki ellen. Ez segít egy hitelesebb kép kialakításában.

A csapatok kezeléséhez a `Csapat` osztály feladata, amely a `Comparable` interfészt implementálja. A kód legvégén található a kis kódcsipetünk.

```
class Csapat implements Comparable<Csapat> {
    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }

    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

Látunk benne egy `Csapat` nevű objektumot, aminek van egy neve és egy értéke. A `Comparable` interfészből adódóan definiáljuk a `compareTo` függvényt, aminek segítségével meg tudjuk határozni, hogyan hasonlítsa össze a két objektumot a program. A paraméterként megadott objektumokat hasonlítja össze a függvény. Visszatérési értéke lehet -1, 1 vagy 0.

## GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témát (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom! Forrás: Link [[https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/GIMP\\_Lisp/Mandala/bhax\\_mandala9.scm](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/GIMP_Lisp/Mandala/bhax_mandala9.scm)]

Ezt a feladatot GIMP-ben fogjuk elkészíteni a Scheme nyelv használatával.

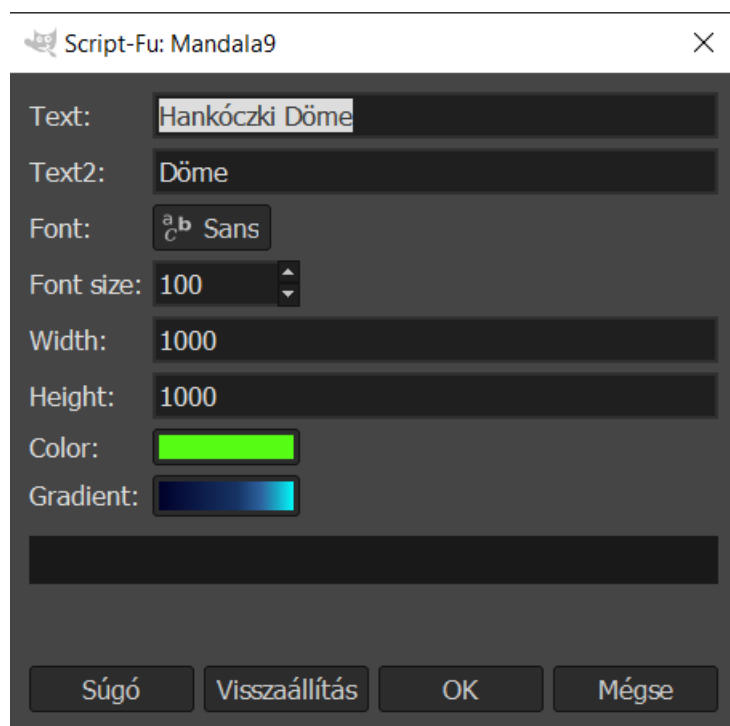
Először is helyezzük el a fájlt a `GIMP 2\share\gimp\2.0\scripts` mappában, így a továbbiakban annyi a dolgunk, hogy a Fájlt>Létrehozás menüpontban kiválasztjuk a scriptünket. Az alakzatot adó



szöveg "Hankóczy Döme" lesz, és középen pedig "Döme" fog elhelyezkedni. A szélesség, magasság, betűtípus, szín és színátmenet állítható akár kódon belül akár grafikus felületen.

```
(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"      "Hankóczy Döme"
  SF-STRING      "Text2"     "Döme"
  SF-FONT         "Font"      "Sans"
  SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
  SF-VALUE        "Width"     "1000"
  SF-VALUE        "Height"    "1000"
  SF-COLOR        "Color"     '(86 252 20)
  SF-GRADIENT     "Gradient"  "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

### 16.1. ábra - beállítás



A kép több rétegből épül fel, amit a gimp-image-insert eljárás alakít ki. A legalsó réteg, ami a bg, sima RGB-ben megadott zöld színt kapott.

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

A `gimp-layer-set-offsets` használatával módosítjuk a szöveg magasságát és szélességet, majd a `gimp-layer-resize-to-image-size` segítségével atméretezzük, hogy illeszkedjen a készülő képre.

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

A szöveg forgatását a `gimp-item-transform-rotate-simple` eljárással valósítjuk meg. Ennek először a szövegréteget, majd a forgatás mérétkét adjuk paraméterül. Végül a `gimp-image-merge-down` összevonja az képet a szövegréteggel.

```
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM-LAYER)))
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM-LAYER)))
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM-LAYER)))

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

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

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

A `gimp-image-select-ellipse` azt mondja a kijelölőnek, hogy cserélje az előző kijelölőmaszk tartalmát.

```
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) (/ textfs
  (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ textfs-h
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

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

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

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) (/ textfs
  (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ textfs-h
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)
```

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

A gradient layert aktívra állítjuk a `gimp-context-set-gradient` alkalmazásával. A `gimp-edit-blend` segítségével a kezdő- és végkoordináták közötti átmenetet alakítjuk. Végül pedig használjuk a `gimp-display-new` és `gimp-image-clean-all` eljárásokat a mandalánk betöltéséhez.

```
(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE "gradient")))
(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT-REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ (/ width 2) (/ height 2))))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)
(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS)))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ height 2) (/ text2-height 2)))
(gimp-display-new image)
(gimp-image-clean-all image)
```

A kód a következő képet generálja le nekünk, lehet nem volt szerencsés a színválasztás, de ezt kaptuk:

**16.2. ábra - Kész mandala**



---

# Irodalomjegyzék

## Általános

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

## C

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

## C++

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

## 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.