

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. szeptember 12, v.
0.0.4

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert	2019. október 16.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	5
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	7
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	7
2.6. Helló, Google!	7
2.7. 100 éves a Brun tétel	8
2.8. A Monty Hall probléma	8
3. Helló, Chomsky!	9
3.1. Decimálisból unárisba átváltó Turing gép	9
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	9
3.3. Hivatkozási nyelv	9
3.4. Saját lexikális elemző	10
3.5. l33t.1	10
3.6. A források olvasása	10
3.7. Logikus	11
3.8. Deklaráció	11

4. Helló, Caesar!	13
4.1. int *** háromszögmátrix	13
4.2. C EXOR titkosító	13
4.3. Java EXOR titkosító	13
4.4. C EXOR törő	13
4.5. Neurális OR, AND és EXOR kapu	14
4.6. Hiba-visszaterjesztéses perceptron	14
5. Helló, Mandelbrot!	15
5.1. A Mandelbrot halmaz	15
5.2. A Mandelbrot halmaz a std::complex osztállyal	15
5.3. Biomorfok	15
5.4. A Mandelbrot halmaz CUDA megvalósítása	15
5.5. Mandelbrot nagyító és utazó C++ nyelven	15
5.6. Mandelbrot nagyító és utazó Java nyelven	16
6. Helló, Welch!	17
6.1. Első osztályom	17
6.2. LZW	17
6.3. Fabejárás	17
6.4. Tag a gyökér	17
6.5. Mutató a gyökér	18
6.6. Mozgató szemantika	18
7. Helló, Conway!	19
7.1. Hangyaszimulációk	19
7.2. Java életjáték	19
7.3. Qt C++ életjáték	19
7.4. BrainB Benchmark	20
8. Helló, Schwarzenegger!	21
8.1. Szoftmax Py MNIST	21
8.2. Szoftmax R MNIST	21
8.3. Mély MNIST	21
8.4. Deep dream	21
8.5. Robotpszichológia	22

9. Helló, Chaitin!	23
9.1. Iteratív és rekurzív faktoriális Lisp-ben	23
9.2. Weizenbaum Eliza programja	23
9.3. Gimp Scheme Script-fu: króm effekt	23
9.4. Gimp Scheme Script-fu: név mandala	23
9.5. Lambda	24
9.6. Omega	24
 III. Második felvonás	 25
10. Helló, Arroway!	27
10.1. OO szemlélet	27
10.2. Gagyí	30
10.3. Yoda	31
11. Hello, Liskov!	32
11.1. Liskow helyettesítés sértése	32
11.2. Szülő-gyerek	33
11.3. Ciklomatikus komplexitás	34
12. Olvasó napló!	36
12.1. Python	36
12.2. Java és C++ összehasonlítása	37
13. Helló, Mandelbrot!	42
13.1. Reverse engineering UML osztálydiagram	42
13.2. Forward engineering UML osztálydiagram	43
13.3. Esettan	43
14. Helló, Chomsky!	45
14.1. Encoding	45
14.2. l334d1c4	46
14.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	48
15. Helló, Stroustrup!	50
15.1.	50
15.2.	50
15.3.	50

16. Helló, Gödel!	51
16.1.	51
16.2.	51
16.3.	51
17. Helló, Valami!	52
17.1.	52
17.2.	52
17.3.	52
18. Helló, Lauda!	53
18.1.	53
18.2.	53
18.3.	53
19. Helló, Calvin!	54
19.1.	54
19.2.	54
19.3.	54
IV. Irodalomjegyzék	55
19.4. Általános	56
19.5. C	56
19.6. C++	56
19.7. Lisp	56

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

Megoldás videó:

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

```
Program T100
{

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

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

```

    }
}

```

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

```

T100 (t.c.pseudo)
true

```

akár önmagára

```

T100 (T100)
false

```

ezt a kimenetet adja.

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

```

Program T1000
{

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

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

    main (Input Q)
    {
        Lefagy2 (Q)
    }

}

```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.4. Labdapattogás

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.5. Szóhossz és a Linus Torvalds féle `BogoMIPS`

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.7. 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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

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

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

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



Bugok

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

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

$$\$(\text{exists } y \text{ forall } x (y < x) \supset \text{neg } (x \text{ text\{prím\}}))\$$$
Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeXMegoldás videó: <https://youtu.be/ZexiPy3ZxsA>, https://youtu.be/AJSXOQFF_wk

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás 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!

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. 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!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. 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.

10. fejezet

Helló, Arroway!

10.1. OO szemlélet

A már tavaly taglalt polárgenerátor implementálásával kezdjük a félévi munkát. Először javában, majd c++ nyelven.

Java implementáció

```
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;
        }
    }
}
```

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

A program egyszerűen működik. Ha nincs tárolt érték, akkor legenerálunk kettő randomszámot, majd általunk még nem ismert (de majd alk.staton megtanuljuk) képletek segítségével létrehozunk két számot. Az egyik számot eltárolja, míg a másikat visszatéríti, mindeközben hamisra állítja a **nincs_Tarolt** logikai változót.

```
-1.3884437079560341  
-0.22173362114739772  
0.7732909445791712  
-0.11453491398610038  
-0.22499864176528497  
-1.4753347117243099  
0.27153920864281333  
-0.11348591603080997  
0.029387032747922006  
-1.4591237094195701
```

A lenti screenshot az openJDK Random.java forrásából származik. Lényegében ez a metódus ugyanúgy működik, mint a mi kódunkban. Azonban mégis van különbség. Például a lenti kódban a **nextGaussian()** metódus kapott egy **synchronized** kulcsszót, amíg egy szál ezt a metódust veszi igénybe, addig a többinek várnia kell.

```

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;
    }
}
}

```

C++ implementáció

```

#include <iostream>
#include <math.h>
using namespace std;

class polar
{
public:
    bool nincstarolt = true;
    double kovi();
    double tarolt;
};

double polar::kovi()
{
    if(nincstarolt)
    {
        double u1,u2,v1,v2,w;
        do
        {
            u1 = ((double) rand() / (RAND_MAX));
            u2 = ((double) rand() / (RAND_MAX));
            v1 = 2*u1 -1;
            v2 = 2*u2-1;
            w = v1*v1+v2*v2;
        }
        while(w>1);

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

```

```

tarolt = r* v2;
    nincstarolt = !nincstarolt;
    return r* v1;
}
else
{
    nincstarolt = !nincstarolt;
    return tarolt;
}
}

int main()
{
    polar g ;
    for (int i=0; i<10;i++)
        cout << g.kovi() << endl;
}

```

10.2. Gagyí

A Gagyiban a feladatunk egy bizonyítás volt. Mégpedig az, hogy alátámasszuk minta programokkal a "while (x <= t && x >= t && t != x);" tesztkérdéstípust, hogy mikor lehet végtelen ciklus. Akkor lehet, ha x, t az egyik esetben az objektum által hordozott érték, míg a másikban meg az objektum referenciája.

```

nohar > ... > prog2 > 1. csokor > gagyí > java gagyí2
-128
-128
nohar > ... > prog2 > 1. csokor > gagyí > □

nohar > ... > prog2 > 1. csokor > gagyí > java gagyí
-129
-129
□

```

Ahogy azt az ábra is mutatja ha x és t értéke -128, akkor a program futása befejeződik, mivel a while ciklusban lévő feltétel nem teljesül. Viszont a második ábrán azt láthatjuk, hogy végtelen ciklust kaptunk. Felmerül a kérdés, hogy lehetséges ez, hisz a kódunk ugyanúgy néz ki csak egy-egyszámmal tér el. A választ az openJDK **Integer.java** fájlban találjuk, amit ezen a linken meg is tekinthetünk.

<https://hg.openjdk.java.net/jdk/jdk11/file/1dddf9a99e4ad/src/java.base/share/classes/java/lang/Integer.java#l1997>

```

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

```


A megoldást a 997. sortól kezdődő **IntegerCache** osztályban találhatjuk. A fenti kódcsipetben generáljuk le az Integer objektumokat, amíg -128 és 127 között választunk számot, akkor egy referenciát kapunk vissza a már legenerált IntegerCache-ben

lévő objektumra, ami mind a két esetben, ugyan oda mutat, így a két objektum meg fog egyezni és a ciklus feltétele nem fog teljesülni.

```
public static Integer valueOf(int i) {  
    if (i >= IntegerCache.low && i <= IntegerCache.high)  
        return IntegerCache.cache[i + (-IntegerCache.low)];  
    return new Integer(i);  
}
```

Feljebb látható, hogy a range-en(-128-127) kívül megadott érték esetén a **return new Integer(i)** fog lefutni, így 2 azonos értékű, de különböző memóriacímű objektum jön létre, emiatt igazzá válik a $x!=t$ feltétel és így egy végtelen ciklust kapunk.

10.3. Yoda

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

```
public class yoda  
{  
    public static void main (String[] args)  
    {  
        String yodaString = null;  
  
        if(("valami").equals(yodaString)){  
            System.out.println ("apu");  
        }  
  
        if(yodaString.equals("anyu"))  
        {  
            System.out.println ("apu");  
        }  
    }  
}
```

Létrehozunk egy string objektumot, amit nullra állítunk. Az első if-nél egy stringet hasonlít egy null értékű string objektumhoz. Ez minden lehetséges. Viszont a második if-nél egy inicializálatlan string objektumra tagfüggvényt hívunk meg.

```
Exception in thread "main" java.lang.NullPointerException  
    at com.company.Main.main(Main.java:7)  
  
Process finished with exit code 1
```

11. fejezet

Hello, Liskov!

11.1. Liskow helyettesítés sértése

Feladatunk a Liskov-elv megsértése volt, ami a következőt foglalja magába: Minden osztály legyen helyettesíthető a leszármazottjával anélkül, hogy a program helyes működése megváltozna.

Ebben a feladatban lehetőségünk van megismerkedni a **getterek** és **setterek** használatával, valamint az **öröklődés** és **polimorfizmus** fogalmak gyakorlati használatával.

```
public class Liskov {  
  
    public static void main(String[] args) {  
  
        Rectangle r = new Rectangle();  
        r.setWidth(10);  
        r.setHeight(20);  
  
        System.out.println(r.getArea());  
  
        Square s = new Square();  
        s.setHeigth(10);  
        s.setWidth(20);  
        System.out.println(s.getArea());  
    }  
}
```

Azt várjuk, hogy a szülő **Rectangle** és gyermek **Square** ugyanúgy viselkedjenek, hisz Square megörökölte Rectangle metódusait, jelen esetben a `getArea()`-t. Viszont futtatás után azt látjuk, hogy a **getArea()** metódus az egyik esetben 200, míg a másik esetben 400-zal tér vissza, mert a Square típusú objektum máshogy viselkedik.

```
<terminated> Liskov [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (2019. szept. 28. 22:36:02)  
200  
400
```

11.2. Szülő-gyerek

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

Ebben a feladatban szintén lehetőségünk van ismerkedni a **getterek** és **setterek** használatával, valamint az **öröklődés** és **polimorfizmus** fogalmak gyakorlati használatával.

```
public class Macska {  
  
    protected String m_name;  
    protected int   m_age;  
  
    public void setName(String name) {  
        m_name = name;  
    }  
  
    public void setAge(int age) {  
        m_age = age;  
    }  
  
    public int getAge() {  
        return m_age;  
    }  
}
```

```
public class Kiscica extends Macska {  
  
    public String getName() {  
        return m_name;  
    }  
}
```

A bal oldalon látható a szülő osztály, míg a jobb oldalon a gyerek osztály, amit kiterjesztettünk a szülőosztályra. Innentől kezdve a Kiscica használhat szinte minden metódust, változót, (nyilván, ha private, akkor nem), ami a Macskában implementálva van. Fordítva viszont nem igaz.

```
public class MacskaKiscica {  
  
    public static void main(String[] args) {  
  
        Macska m = new Macska();  
        m.setName("Hókuszpók");  
        m.setAge(49);  
  
        Kiscica k = new Kiscica();  
        k.setName("Sziamiaú");  
        k.setAge(13);  
  
        System.out.println(k.getAge() + " " + m.getName());  
    }  
}
```

A main metódusunkban példányosítottuk a Macskát és a Kiscicát is, majd beállítottuk az értékeket. A kiíratásnál azonban problémába ütközünk. A korábban már említett ok miatt, azt írja a fordító, hogy nincs definiálva a getName() a Macska osztályunkban, ami persze igaz, hisz csak a Kiscicában lett létrehozva. Tehát láthatjuk, hogy oda-vissza nem működik az öröklődés.

```
<terminated> MacskaKiscica [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (2019. szept. 29. 15:25:06)  
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
    The method getName() is undefined for the type Macska  
  
    at macskaKiscica.MacskaKiscica.main(MacskaKiscica.java:15)
```

11.3. Ciklomatikus komplexitás

forrás: https://www.tankonyvtar.hu/en/tartalom/tamop425/0046_szoftverteszteles/ch04s04.html

Az összetett jobb, a komplikáltnál. Az rendben van, hogy egy nagyon összetett szoftvert készítesz, de ne komplikált módon tedd ezt!

Ebben a feladatban, az volt a dolgunk, hogy egy általunk választott program függvényeinek a ciklomatikus komplexitását számoljuk ki. Több programot is használhatunk a számításhoz. Sajnos nekem többszöri próbálkozás után, több program használatával sem sikerült. De végül csak megcsináltam. Pacsi :) Interneten megtalálható egy olyan weboldal, ami az általad írt program forrás kódját kéri, hogy másold be oda és ő majd kiszámolja neked a komplexitását a programnak.

Itt található: http://www.lizard.ws/?fbclid=IwAR3IoZ8H3MOSFOt7weO_rqKQ8ekKkOfofBhI

Na de nézzük meg miről is van szó. Ciklomatikus komplexitás = vezérlési gráfban megtalálható független utak száma. Akkor mondjuk, hogy két út független, ha mindkettőben van olyan pont vagy él, ami nem eleme a másik útnak.

És végül egy Screen, hogy lássuk valóban bemásoltam a forráskódot.

<div><div>File Type</div><div>.java</div><div>Token Count</div><div>467</div><div>NLOC</div><div>59</div></div>			
Function Name	NLOC	Complexity	Token #
BBP::BBP	20	3	196
BBP::n16MODk	20	6	103
BBP::d16Sj	10	3	120
BBP::toString	3	1	8
BBP::main	3	1	22

12. fejezet

Olvasó napló!

12.1. Python

Sokszor előfordul alkalmazás fejlesztése közben, hogy egy adott problémát C, C++ vagy éppen java nyelven macerás megírni. Ilyenkor nagy szolgálatot tud tenni nekünk a python nyelv, ahol nincs szükség fordításra, mivel az értelmezőnek elég megadni a forrást és automatikusan futtatja az alkalmazást.

Adott alkalmazás elkészítése a már említett nyelveken sokkal nehezebb, több időt vesz igénybe és jóval hosszabb kódot von maga után.

Sokszor előfordul alkalmazás fejlesztése közben, hogy egy adott problémát C, C++ vagy éppen java nyelven macerás megírni. Ilyenkor nagy szolgálatot tud tenni nekünk a python nyelv, ahol nincs szükség fordításra, mivel az értelmezőnek elég megadni a forrást és automatikusan futtatja az alkalmazást.

Adott alkalmazás elkészítése a már említett nyelveken sokkal nehezebb, több időt vesz igénybe és jóval hosszabb kódot von maga után.

Van egy úgynevezett python kódkönyvtár, ami rengeteg modult tartalmaz, amik megkönnyítik a kódírást. Pl.: fájlkezelés, hálózatkezelés, rendszerhívások felhasználói felület készítése.

Köztes nyelv, nincs szükség linkelésre, fordításra. Tömör, de jól értelmezhető kódokat tudunk benne írni.

Kód szerkesztése

behúzás alapú: nincsenek kapcsos zárójelek, egy adott blokk végét egy kisebb behúzású sor jelzi. Az első utasítás nem lehet behúzott.

sor végéig tart egy utasítás, így nem kell a ; jelet használni. Hanem fér ki egy sorba, akkor \ jellel jelezzük.

Tokenek:

azonosító, kulcsszó, operátor, delimiter, literál

Kis- nagy betű érzékeny.

megjegyzések a # jel után használható.

sor végéig tart egy utasítás, így nem kell a ; jelet használni. Hanem fér ki egy sorba, akkor \ jellel jelezzük.

Tokenek:

azonosító, kulcsszó, operátor, delimiter, literál

Kis- nagy betű érzékeny.

megjegyzések a # jel után használható.

Típusok

Minden adatot objektumok reprezentálnak. Az objektum típusa határozza meg, hogy milyen művelet használható rajta. Nem kell a változókat explicit megadni, a rendszer futtási időben kitalálja a azt, a hozzárendelt érték alapján.

Adattípusok: számok, sztringek, ennesek, listák, szótárak.

számok: egész, lebegőpontos, komplex

sztring: idézőjel vagy aposztróf is használható.

ennesek: objektumok gyűjteményei vesszővel elválasztva, Általában zárójelekbe írjuk őket.

lista: rendezett szekvencia, a típusok lehetnek különbözőek is. Szögletes zárójelek közé írjuk őket.

szótár: kulcsokkal azonosított elemek rendezetlen halmaza. AZ elemek lehetnek különböző típusúak is, Használata: { } között vesszővel elválasztva.

Minden adatot objektumok reprezentálnak. Az objektum típusa határozza meg, hogy milyen művelet használható rajta. Nem kell a változókat explicit megadni, a rendszer futtási időben kitalálja a azt, a hozzárendelt érték alapján.

12.2. Java és C++ összehasonlítása

A java nyelv tervezői a C és C++ nyelv szintaxisát vették alapul. Sok minden hasonlóságot fedezhetünk felakét nyelv között, de fontos megjegyezni, hogy a hasonlóság nem azonosság.

Objektummodellek különbözősége

A C++ nyelvben az objektumok a memória egy összefüggő területén elhelyezkedő bájt sorozatok. A pointerek révén közvetlenül manipulálhatjuk a memóriát. Viszont a java nyelvben a memóriát nem tudjuk közvetlenül elérni, csak hivatkozásokon keresztül. A javában nincs linker, így nincs, ami címezzé oldaná fel a hivatkozásokat.

A java ismeri egy osztály kompatibilis megváltoztatásának fogalmát. Ha a hivatkozott osztályban megváltoznak a tagok vagy függvények deklarációs sorrendje, vagy újjal bővülnek, akkor a hivatkozó osztály újrafordítás nélkül is érvényes marad.

A java virtuális gép az objektumokat egy szemétygyűjtő mechanizmus által ellenőrzött tárterületen tárolja (garbage collector). Ezért egy metódus paramétere és visszatérítési értéke is csak primitív típusú vagy objektumhivatkozás lehet, de maga az objektum nem. A java nyelvben nincs auto destruktorkívó mechanizmus.

Legyen T egy típus. Adott az alábbi kódrészlet C++-ban:

```
T x;
```

A fenti csipet egy T típusú objektumot hoz létre annak alapértelmezett konstruktorával. Viszont javában ez egy inicializálatlan T típusú változót hoz létre, szóval jelentésben inkább erre hasonlít.

```
T *x;
```

Vagy például:

```
T w[10];
```

Egy T-kből álló tömböt kezdőértékkadás nélkül csak úgy tudunk felvenni, ha a T osztálynak van alap konstruktora, hiszen ez végzi az inicializálást. Viszont java-ban

```
T W[] = new T[10];
```

a fenti kód hatására egy hivatkozásokból álló 10 elemű tömb fog létrejönni, amiben mindegyik elem null értékű lesz.

A javaban ugye nincs destruktork mechanizmus, ezt a **finalize()** metódus tudja úgy-ahogy pótolni.

A C++ nyelvben pontosan tudjuk, hogy meddig "él" egy objektum, a deklaráló blokkba való belépéskor kezdődik és a blokk végrehajtásának végén törölődnek. A memóriában foglalt objektumok élettartamát mi szabályozzuk a **new** valamint **delete** operátorokkal.

Javában nincsenek globális változók és függvények, csak osztályokhoz és interfészekhez tartozó metódusok, változók, konstansok.

Kommentek

A C++-ban megszokott `/* */` és `//` jeleken kívül lehetőségünk van használni a `/**` dokumentációs kommentet. Így a javadoc segítségével HTML- formátumú dokumentációt készíthetünk.

Előfordító

C++-ban fordításkor preprocesszor dolgozza fel: kifejti a makrókat, behelyettesíti a definiált szimbólumokat, beemeli a header fájlokat. Viszont a java nem rendelkezik előfordítóval, így nem támogatja a header fájlok beemelését, valamint a makrókat sem. Viszont valamilyen szinten támogatja a szimbólumbehelyettesítést, mégpedig úgy, hogy a szimbólumként való használatra szánt azonosítókat valamilyen osztályban vagy interfészben **final static** adattagként definiálhatja.

Header fájlok

Ahhoz, hogy a több különböző forrásból is hivatkozott objektumokat következetesen használjuk, az átlományokban azonosan kell deklarálnunk őket. Ezeket a közös deklarációkat tudjuk egy header fájlban elhelyezni. Így tudjuk garantálni a deklarációk megegyezőségét.

Viszont a java-nak nincs ilyen mechanizmusa és nincs is szüksége rá. Hisz a java fordítóprogram a hivatkozott osztályok tulajdonságait alapvetően a class-fájlokból veszi.

A C++-ban egy osztály deklarációja és definíciója eékülönül, és a deklaráció végső zárójele után pontosvesszőt kell raknunk. Viszont javában lehet, de nem szükséges a pontos vesszőt kirakni.

Ha egy osztály definíciót egy másik osztályból felszeretnénk használni, az **import** paranccsal tehetjük láthatóvá a szükséges neveket.

Primitív típusok

C++	Java	Java burkoló osztály	méret
bool	boolean	Boolean	logikai típus
char	-	-	8 bites karakter
wchar_t	char	Character	16 bites Unicode karakter
signed char	byte	Byte	8 bites előjeles szám
short	short	Short	16 bites előjeles egész
long	int	Integer	32 bites előjeles egész
long long	long	Long	64 bites előjeles egész
float	float	Float	32 bites lebegőpontos szám
double	double	Double	64 bites lebegőpontos szám
long double	-	-	>64 bites lebegőpontos szám

Javában a boolean típus a C++-beli bool-lal ellentétben nem tartozik az egész típusok családjához. Ezért az **if**, **for**, **do**, **while** utasításokban feltételként csak logikai típusú kifejezést használhatunk, egészet vagy objektumhivatkozást nem.

Mutatók, referenciák

Javában nincsenek külön objektumok és mutatók. A objektumok dinamikus tárterületen jönnek létre és csakis hivatkozásokon keresztül érhetőek el, referencia vagy mutató használata nélkül.

Nincsenek se függvénymutatók, se tagfüggvényre vagy adattagra mutató mutatók sem. Ezek helyett objektum-referenciákat, visszatérési értékeket, tömböket, interfészeket használhatunk.

Operátorok, kifejezések

A javában nem tudunk felhasználói operátorokat definiálni, így be kell érünk a gyáriakkal. Megtalálható aritmetikai, logikai, bitmanipuláló, összehasonlító, kiválasztó operátor is. Továbbá használhatunk egy úgynevezett előjel-kiterjesztéses **>>**, valamint zéró-kiterjesztéses jobbróléptetést **>>>** is, vagyis a balról belépő bit mindig 0.

Az objektumtípusok körében ezek az operátorok használhatóak:

?: kiválasztó

== és **!=** összehasonlító (azonos objektumokra hivatkozó referenciákra ad igazat)

instanceof típus-hasonlító

+ string-konkatenáció

Javában nem tudjuk az operátorokat túl terhelni, vagy magunknak definiálni. Vegyük például ezt a C++ kódcsipetet.

```
cout << a << " " << b << endl;
```

Ez egy elegáns megoldás, amit az operátorok és referenciák tesznek lehetővé. Ha valami hasonlót akarunk kihozni javában az így néz ki:

```
System.out.println(new StringBuffer(a).append(" ").append(b));
```

Const

Javában ugyan létezik, de nincs jelentése. Viszont van egy **final** kulcsszó, ami

- adattagra vonatkoztatva az adott változó értékének a változatlanóságát jelenti
- módszerre vonatkoztatva a módszer felülbíráltatlanságát,
- míg osztályra, azt, hogy nem lehet belőle származtatni.

Final minősítésű osztály például a String és a primitív típusok burkolóosztályai: **Byte, Character, Double, Float, Integer, Long, Short**

Argumentumok alapértelmezett értéke

C++-ban egy függvény argumentumainak adhatunk alapértelmezett értéket.

```
class x
{
    void doit(int x=0, int y=0);
};
```

Viszont javában erre nincs lehetőség, így manuálisan kell megadni.

```
class x
{
    void doit()
    {
        doit(0,0);
    }
    void doit(int x)
    {
        doit(x,0);
    }
    void doit(int x,int y)
    {
        //...
    }
}
```

Utasítások

A **for**utasításban java nyelvben is deklarálhatunk ciklusváltozót, viszont nem deklarálhatunk változót az **if** utasításban.

Osztályok, interfészek, öröklődés

A java nyelvi szinten megkülönböztet osztályokat(absztrakt, nem absztrakt) és interfészeket.

Az öröklődést az **extends** kulcsszó jelzi, és itt nincs megkülönböztetés private, protected és public öröklődés között.

Ha egy osztályt **abstract** kulcsszóval deklarálunk, akkor egyrészt nem példányosítható, másrészt lehetnek benne csak deklarált, de nem implementált módszerek.

Az olyan "absztrakt osztályokra", amikben csak konstansok és implementáció nélküli függvények vannak, külön bevezették az interfészt. Az **interface** kulcsszóval tudjuk deklarálni őket. Öröklődés jelölésére itt is az **extends** kulcsszót használjuk. Tudni illik, hogy a java támogatja az interfészek esetén a többszörös

öröklődést, viszont osztályokra ez nem vonatkozik. Ha jelezni akarjuk, hogy egy osztály egy interfészt implementál, akkor az **implements** kulcsszót használjuk.

Hozzáférési kategóriák

A java támogatja a **félnyilvános** hozzáférési kategóriát, a másik 3 mellett, amit C++-ban megtalálható (private, public, protected). Másképp csomagszintű elérhetőség.

A java-ban a **public** vonatkozhat egy osztályra is, de ha nem adjuk meg az adott osztály csak a saját csomagjából érhető el.

Az interfészek minden tagja automatikusan public, de maga az interfész nem az.

Friend

Nincs barátság. Ennek kiküszöbölésére a java beágyazott osztályokat használ. Hasonlít a C++-éhoz, de mégis lényegi különbség van köztük. A beágyazott java osztály metódusai elérik a beágyazás helyén látható változókat.

Felsorolási típus

Javában sokáig nem volt, viszont az 5-ös verzióban bevezetésre került. Tipikusan egy **for each** ciklusban szokták használni. Rendelkeznek az **Object** osztály metódusaival és megvalósítják a **Comparable** és **Serializable** interfészeket.

Minden felsorolási típus rendelkezik egy statikus values metódussal, amely egy tömbbel tér vissza, amiben a felsorolási típusban deklarált értékek vannak deklarációjuk sorrendjében.

Rendelkezésünkre áll a **java.util** package-ben létrehozott két segédosztály. **EnumSet** egy Set megvalósítás és részintervallumok meghatározására szolgál, míg az **EnumMap** egy olyan nagyhatékonyságú Map megvalósítás, amit akkor használunk, ha egy felsorolási értékhez rendelünk hozzá valamit.

String

Javában is van beépített String osztály, két bájtos Unicode karakterek füzérét ábrázolja. Ez egy **final**, tehát nem származtathatunk belőle másik osztályt.

Létrejötté után nem módosítható, karakterei nem cserélhetőek ki, újabb karakterek nem fűzhetőek hozzá. Ezek miatt is hívjuk a java stringeket **szálbiztosnak**. Ha változtatható karaktersorozatra van szükségünk, alkalmazzuk a **StringBuffer** osztályt.

Ha egy osztályban egy metódust **final** minősítővel látunk el, akkor öröklő osztályban azonos nevű metódus nem szerepelhet. A gyári osztályok között sok final metódus fordul elő, például az **Object** általános őssztálynak a párhuzamos végrehajtási szálak szinkronizációjával kapcsolatos metódusok ilyenek, hiszengond lehet abból, ha valaki ezeket felülbírálná.

Egy osztály metódusából egy szülőosztálybeli tag szükség esetén a **super** kulcsszóval érhető el, akkor is, ha túl van terhelve vagy el van fedve.

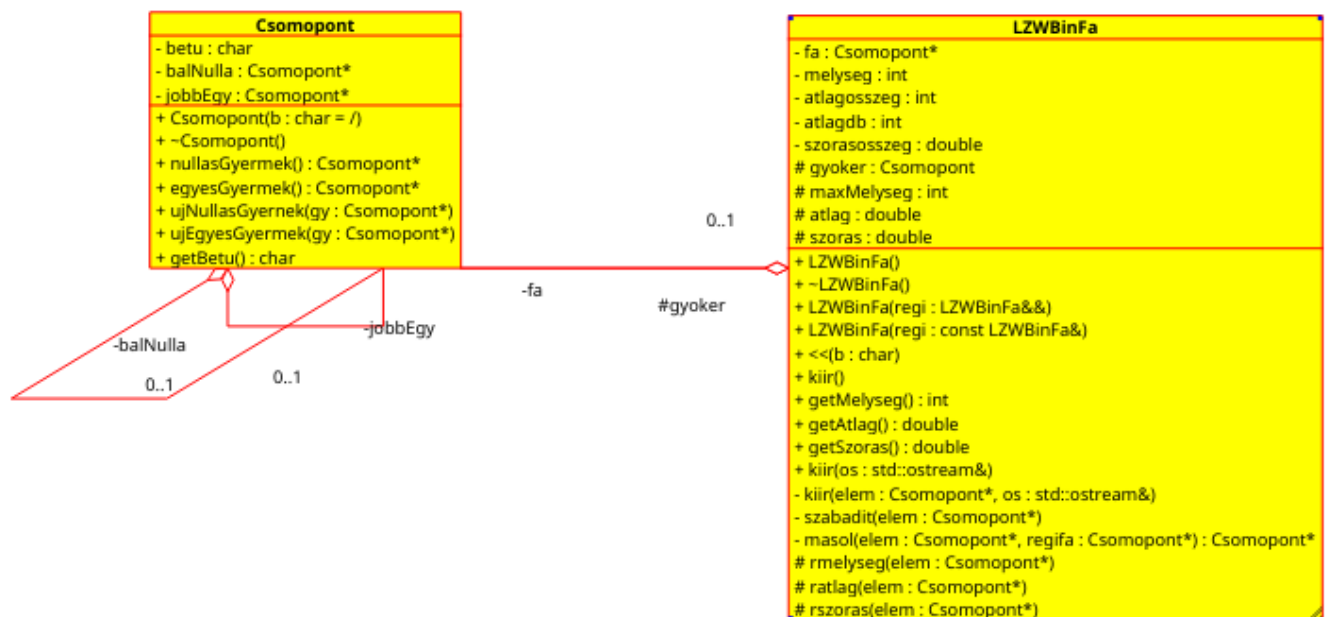
13. fejezet

Helló, Mandelbrot!

13.1. Reverse engineering UML osztálydiagram

Feladatunk az volt, hogy a tavalyi védési feladathoz kellett UML osztálydiagramot készíteni. A forrásból kellett legenerálnunk ezt a diagramot. Én az umbrello nevű programot használtam.

Először **sudo apt install umbrello** parancsot beírva a terminálba feltelepítettem a programot. Viszonylag könnyen rájöttem a program használatára, beszédes



Nézzük meg az aggregáció és kompozíció fogalmakat. Az aggregáció olyan asszociáció ,ami tartalmazást jelöl, jele: egy üres rombusz a tartalmazó oldalon. Míg kompozíciónál a tartalmazott objektum pontosan egy tartalmazó objektumhoz tartozik. A tartalmazó és a tartalmazott objektum együtt jön létre, és együtt szűnik meg. Jele tele rombusz. Általában a kompozíciót ugyanúgy implementáljuk, mint az asszociációt, de ilyenkor destruktorban meg kell írunk a felszabadító kódot.

13.2. Forward engineering UML osztálydiagram

Ebben a feladatban az előző feladat ellenkezőjét kellett végrehajtani. UML-ben kellett terveznünk osztályokat és generálni kellett belőlük forrást!



Csomopont.cpp



Csomopont.h



LZWBinFa.cpp



LZWBinFa.h

Az előző feladatban elkészült UML diagramból generáltam vissza a fentebb látható forrásokat. Futtatásnál több, hibát is kiírt a fordító. Az umbrello pl. hagyta a relációkat, amikor inklúdoltam például a string header fájlt. Másik hiba, hogy hagyott egy aposztrófot a csomópont konstruktoránál, ahol beállítjuk, hogy mi legyen az értéke a gyökérnek.

Végül egy csipet a kódból.

```
int getMelyseg ()
{
}

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

13.3. Esettan

Feladatunk, hogy feldolgozzuk a szoftverfejlesztés című könyv 14. fejezetét, azaz mindent, amit az uml diagramokról tudni érdemes.

Legfontosabb elemei az osztályok, ezeket téglalappal jelöljük. 3 részre tudjuk osztani: felül az osztályneve, középen a tagváltozói, míg alul az osztály tagfüggvényei, metódusai találhatók.

Szintaxis

A láthatósági szinttel kezdjük, ami lehet public(+), private (-), protected(#), valamint package(~) is, nyilván az utóbbi c++-ban nem definiált. A származtatott tagváltozókat / jellel jelöljük. A konstans tagváltozókat read-only tulajdonsággal jelöljük.

A tagfüggvényeket szintén láthatósági névvel látjuk el először, majd a paraméter listával, végül a visszatérési értékkel. A paraméterlista szintaxisa: irány név: típus = alap érték.

Az irány 3 féle lehet :

- in(érték szerinti paraméterátadás)
- out (referenci szerinti)
- inout(pointert adunk át)

Kapcsolatok

Asszociáció: Amikor két osztály elakarja érni egymás objektumait,akkor van szükség asszociációra. Vonallal jelöljük. A vonalra kerülnek az érintett objektumok nevei. Valamint egy nyíl jelöli az irányt.

Szerepnév: megmutatja, hogy miként vesz részt az adott osztály az asszociációban.

Az asszociációnak van két speciális fajtája, a **kompozíció** és az **aggregáció**

Aggregáció: Olyan asszociáció ,ami tartalmazást jelöl, jele: egy üres rombusz a tartalmazó oldalán

Kompozíció: A tartalmazott objektum pontosan egy tartalmazó objektumhoz tartozik. A tartalmazó és a tartalmazott objektum együtt jön létre, és együtt szűnik meg. Viszont ha a tartalmazott objektumból biztosan nem származik semmi, akkor a kompozíciót implementálhatjuk pointer helyett objektumként, ami tagváltozóként vagy statikus/ dinamikus tömb tagváltozóként jelenik meg.

A feladat második részében a könyben szereplő esett tanulmányt kellett tanulmányozni. Ebben nagy segítségemre volt Levendovszky CD melléklete, amit az Ncore-ról vettem és Feri könyve is, amit így utólag is köszönök. Ez egy raktár kezelő program, ami adatfolyamból olvassa és oda is írja a készletet, ahogy azt az alábbi screenen is láthatjuk.

A program futás közben:

```
nohar ... prog2 3.csokor Esettan g++ -c *.cpp
nohar ... prog2 3.csokor Esettan g++ *.o -o Feri
nohar ... prog2 3.csokor Esettan ./Feri
Test1: create inventory and printing it to the screen.
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20191003, Age: 0, Current price: 30000, InchWidth: 13, InchHeight: 12
1.: Type: HardDisk, Name: WD, Initial price: 25000, Date of acquisition: 20191003, Age: 0, Current price: 25000, SpeedRPM: 7500
Press any key to continue...

Test2: loading inventory from a file (computerproducts.txt), printing it, and then writing it to a file (computerproducts_out.txt).
End of reading product items.The content of the file is:
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20011001, Age: 6575, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 4750, Current price: 28000, InchWidth: 10, InchHeight: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age: 4750, Current price: 70000
Items:
0.: Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20011001, Age: 6575, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: HardDisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 4750, Current price: 28000, SpeedRPM: 7000
3.: Type: HardDisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20050228, Age: 5329, Current price: 20000, SpeedRPM: 7000
The content of the inventory has been written to computerproducts_out.txt
Done.
```

14. fejezet

Helló, Chomszky!

14.1. Encoding

<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

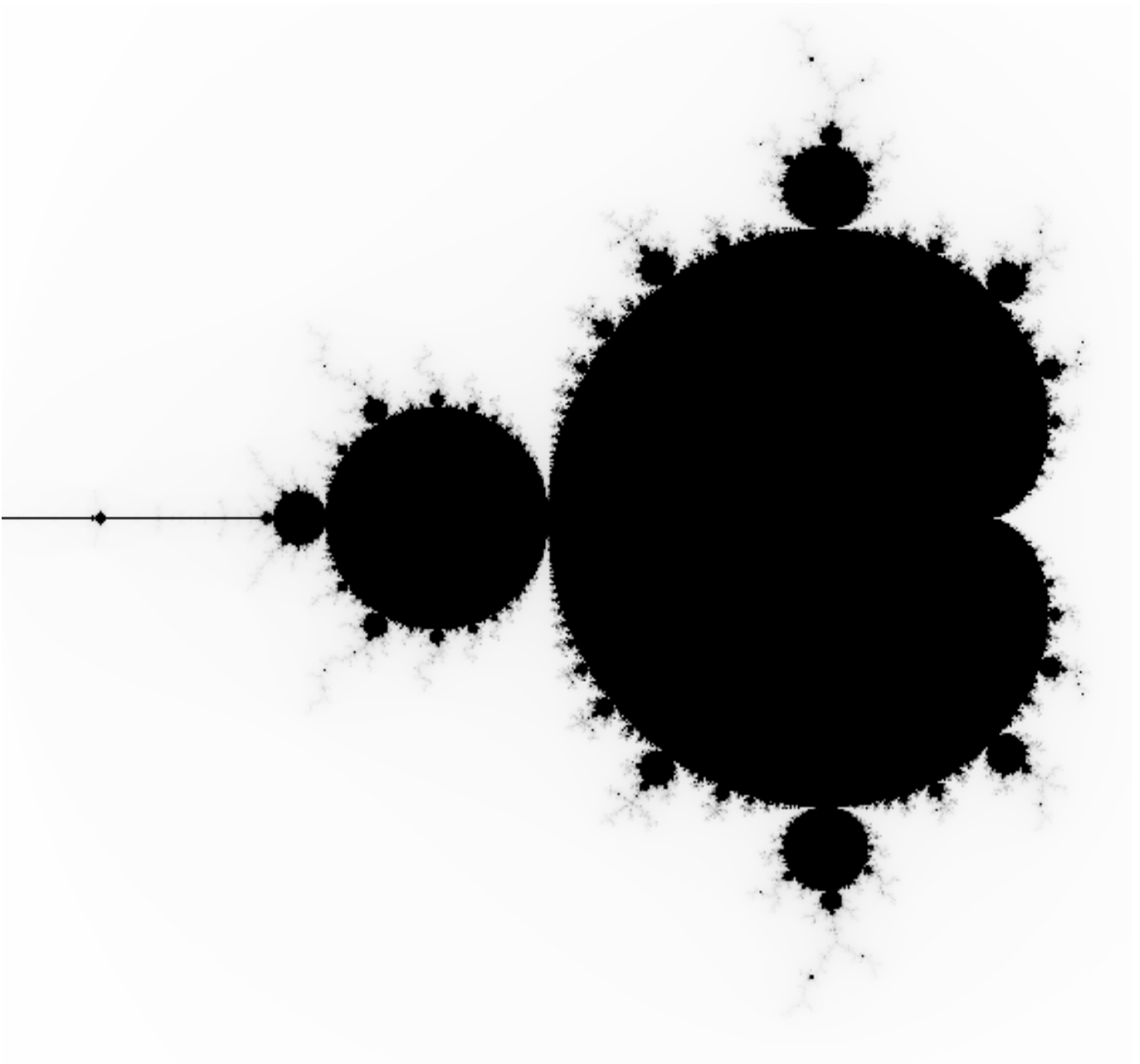
Feladatunk az volt, hogy futtasuk a Mandelbrothalmaznagyító java forrást úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket!

A feladatom gyakorlatilag annyi volt, hogy a fentebb linkelt oldalról, letöltöttem a fájlokat és átírtam ékezetesekre a források neveit. Ezután, ha a szokásos módon fordítanánk terminálon hibát kapnánk.

```
ding> javac MandelbrotHalmazNagyító.java
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xED) for encoding UTF-8
* MandelbrotHalmazNagyító.java
^
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xF3) for encoding UTF-8
* MandelbrotHalmazNagyító.java
^
MandelbrotHalmazNagyító.java:4: error: unmappable character (0xED) for encoding UTF-8
* DIGIT 2005, Javat tanítók
^
MandelbrotHalmazNagyító.java:5: error: unmappable character (0xE1) for encoding UTF-8
* Batfai Norbert, nbatfai@inf.unideb.hu
^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xED) for encoding UTF-8
* A Mandelbrot halmazt nagyító és kirajzoló osztály.
^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding UTF-8
* A Mandelbrot halmazt nagyító és kirajzoló osztály.
^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xE9) for encoding UTF-8
* A Mandelbrot halmazt nagyító és kirajzoló osztály.
^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding UTF-8
* A Mandelbrot halmazt nagyító és kirajzoló osztály.
^
```

Viszont megfelelő kapcsolók segítségével könnyen fordíthatóvá tehetjük a kódunkat.

```
nohar ... > prog2 > 4.csokor > Encoding > javac -encoding iso-8859-2 MandelbrotHalmazNagyító.java
```



14.2. l334d1c4

<https://simple.wikipedia.org/wiki/Leet>.

A feladat az volt, hogy írjunk olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja a fenti linken található betűhelyettesítést. Nézzük végig a forrás kódot. Kezdjük az osztály-definíciójával. Példányosítottam 2 string és egy hashmap objektumot, valamint beleírtam még egy print metódust is, amit később meg akarok majd hívni.

```
class LeetCypher {  
    private static String leet = new String();  
    private static String alphabet = new String();  
    Map<String, String> character = new HashMap<String, String>();
```



```
public void print(String leet) {  
    System.out.println(leet);  
};
```

Többféle adatszerkezet használatával próbálkoztam, de végül a map-ot definiáltam. Ez volt számomra a legkényelmesebb a feladat megoldása szempontjából, ahogy láthatjuk az alábbi kódcsipetben "kulcs-érték párokat" hoztam létre. A **transform** függvényem átalakítja az angol ABC betűit görög betűkké. A görög betűket unicode-ok segítségével oldottam meg.

```
public void transform(String alphabet) {  
    character.put("A", "\u0391");  
    character.put("B", "\u0392");  
    character.put("C", "\u0393");  
    character.put("D", "\u0394");  
    character.put("E", "\u0395");  
    character.put("F", "\u039A");  
    character.put("G", "\u0397");  
    character.put("H", "\u0398");  
    character.put("I", "\u0399");  
    ...}
```

Ezután létrehoztam egy konstruktort, amibe elhelyeztem a korábban említett print és transform metódusokat.

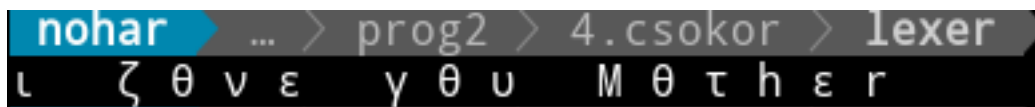
```
public LeetCypher() {  
  
    transform(alphabet);  
    print(leet);  
}
```

Majd a main függvényben egy try - catch, try részében példányosítok egy StringBuilder típusú builder nevű objektumot. Ezt az osztályt akkor használjuk, ha egy adott szövegen szeretnénk változtatni. A kapacitásának inicializálásával egy elfogadható méretre minimalizálhatjuk a memórafoglalások számát, amivel sokkal hatékonyabbá tehetjük programunkat, hisz a memórafoglalás elég költséges művelet. Például, ha elfogyna a szabad kapacitás az objektumunkban, akkor egy új, de kétszer akkora memóriaterület lesz lefoglalva, a régi tartalom átmásolásra kerül. Ebből is látszik, hogy érdemes megadni neki legalább egy becsült kapacitást.

```
try {  
    StringBuilder builder = new StringBuilder(alphabet.length());  
    for (String str : args) {  
        builder.append(str + ' ');  
    }  
  
    alphabet = builder.toString();  
    alphabet = alphabet.toUpperCase();  
    new LeetCypher();  
}  
catch (Exception e) {
```

```
usage();
}
```

Meghívjuk a LeetCyhper konstruktort, ami a felhasználó által begépelte karaktereket a görög ABC betűire cseréli ki, majd kiírja azt a standard outputra, figyelembe véve a kis-nagy betűket is. Ha nem megfelelő a begépelte string, akkor a catch részben dobunk egy üzenetet, a helyes használatról.

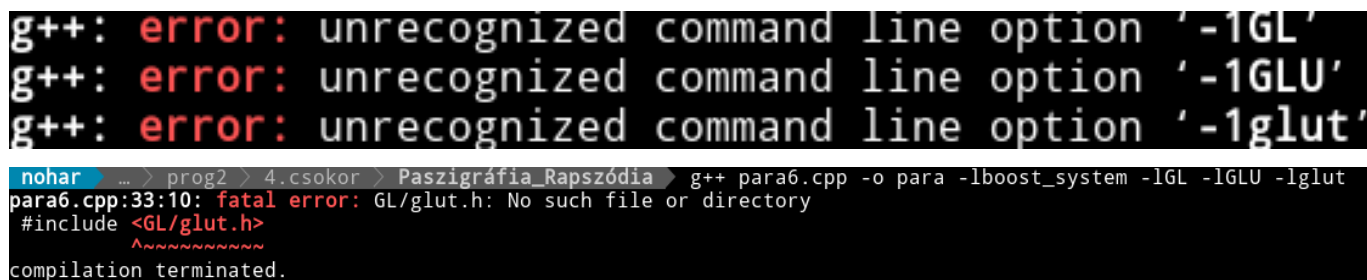


14.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

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

Paszigráfia Rapszódia – az esport kultúra nyelve.

Mielőtt hozzá kezdenénk a feladathoz, először le kell töltenünk 2 csomagot, nevezetesen a **libboost**-ot és a **freeglut3**-t, hogy fordítható legyen a programunk, különben hasonló hiba üzeneteket kaphatunk fordításnál, mint ahogy azt én is tapasztaltam:



Rákeresve az egyik legnépszerűbb keresőoldalon, ezen a linken találtam segítséget:

<https://askubuntu.com/questions/432732/fatal-error-gl-glut-h-no-such-file-or-directory>

Terminálból könnyen telepíthetőek. Bár ahogy, így utólag néztem Bártfai tanár úr kódjában be van kommentelve, hogy mit kell feltelepíteni és hogy hogyan kell fordítani, futtatni. Legközelebb jobban figyelek.

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

A freeglut3-at a grafikus megjelenítésre használjuk. Népszerű alternatívája OpenGL Utility Toolkit-nek, hisz használata könnyebb.

Egyik feladatom a színvilág megváltoztatása volt. Ezt az alábbi kódcsipet segítségével értem el. A kedvenc színem a király kék, így ezt a hátteret választottam.

```
glClearColor ( 0.8f, 2.0f, 0.5f, 1.0f );
```

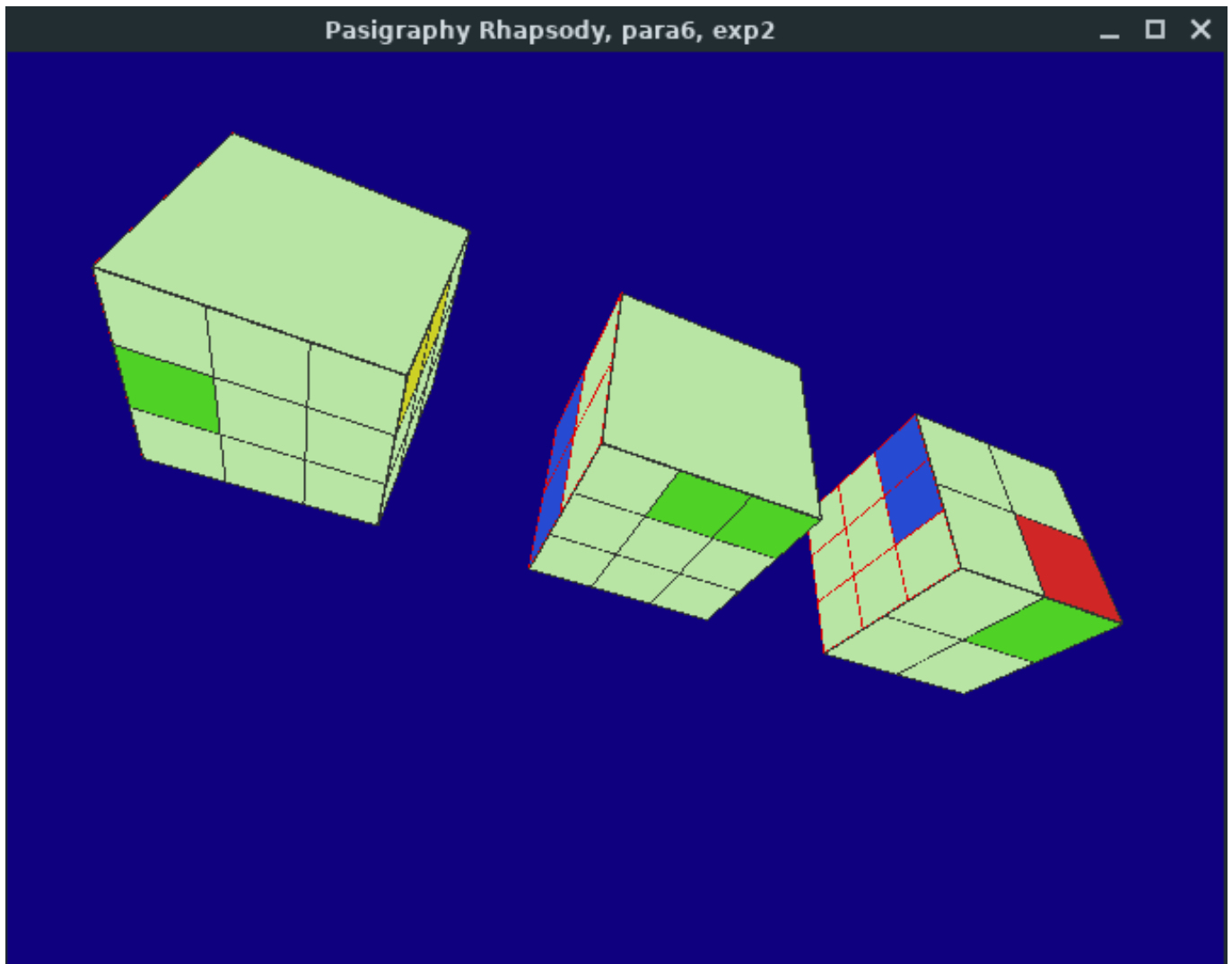
Megváltoztattam a kockák dőlés szögét és elhelyezkedését is:

```
gluLookAt ( 2.0f, 6.0f, 5.0f ,1.0f ,3.5f ,2.0f ,3.0f ,3.0f ,8.0f );
```

Majd a **keyboard** metódust bővítettem egy kilépő funkcióval. Vagyis mostantól azt **esc** lenyomásával kitundunk lépni a progiból.

```
else if (key == 27)
{
    exit(0);
}
```

Végül egy screenshot a program működéséről.



15. fejezet

Helló, Stroustrup!

15.1.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.2.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.3.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16. fejezet

Helló, Gödel!

16.1.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.2.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.3.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17. fejezet

Helló, Valami!

17.1.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.2.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.3.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18. fejezet

Helló, Lauda!

18.1.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.2.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.3.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19. fejezet

Helló, Calvin!

19.1.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.2.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.3.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

19.4. Általános

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

19.5. C

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

19.6. C++

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

19.7. Lisp

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

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

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