

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. május 09, v. 1.0.0

Copyright © 2019 Dr. Bátfai Norbert

Copyright © 2019 Orosz Máté

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

Copyright (C) 2019, Orosz Máté, ormate99@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 Ã©s Orosz , Máté	2019. december 1.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

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. Második felvonás	3
2. Helló, Arroway!	5
2.1. OO szemlélet	5
2.2. Homokozó	7
2.3. Yoda	13
3. Helló, Liskov!	15
3.1. Liskov helyettesítés sértése	15
3.2. Szülő-gyerek	15
3.3. Ciklomatikus komplexitás	17
4. Helló, Mandelbrot!	18
4.1. Reverse engineering UML osztálydiagram	18
4.2. Forward engineering UML osztálydiagram	19
4.3. BPMN	21
5. Helló, Chomsky!	22
5.1. Encoding	22
5.2. l334d1c45	22
5.3. Full screen	24

6. Helló, Stroustrup!	27
6.1. JDK osztályok	27
6.2. Másoló-mozgató szemantika	28
6.3. Hibásan implementált RSA törése	30
7. Helló, Gödel!	32
7.1. Gengszterek	32
7.2. STL map érték szerinti rendezése	32
8. Helló, !	34
8.1. FUTURE tevékenység editor	34
8.2. OOCWC Boost ASIO hálózatkezelése	35
8.3. SamuCam	35
9. Helló, Schwarzenegger!	39
9.1. Port scan	39
9.2. AOP	39
9.3. Junit teszt	40
10. Helló, Calvin!	42
10.1. MNIST	42
10.2. Deep MNIST	43
10.3. CIFAR-10	45
10.4. Android telefonra a TF objektum detektálója	45
10.5. SMNIST for Machines	49
10.6. Minecraft MALMO-s példa	49
11. Helló, Berners-Lee!	50
11.1. C++ és Java összehasonlítás	50
11.2. Python	50
III. Irodalomjegyzék	52
11.3. Általános	53
11.4. C	53
11.5. C++	53
11.6. Lisp	53

Ábrák jegyzéke

3.1. komplexitas	17
4.1. ReverseUML	19
4.2. BPMN	21
6.1. RSA	31
10.1. ailearning	46
10.2. ailearning	47
10.3. ailearning	48

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

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

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.

2. fejezet

Helló, Arroway!

2.1. OO szemlélet

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
import java.util.Random;
import java.io.*;
import java.lang.Math;
public class PolarGen {
    public final static int RAND_MAX = 32767;
    private static boolean bExists;
    private double dValue;
    static Random cRandomGenerator = new Random();
    public PolarGen() {
        bExists = false;
        cRandomGenerator.setSeed(20);
    }
    public double PolarGet() {
        if (!bExists)
        {
            double u1, u2, v1, v2, w;
            do{
                u1 = cRandomGenerator.nextInt (RAND_MAX) / (RAND_MAX + 1.0);
                u2 = cRandomGenerator.nextInt (RAND_MAX) / (RAND_MAX + 1.0);
                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);
dValue = r * v2;
bExists = !bExists;
return r * v1;
}
else
{
    bExists = !bExists;
    return dValue;
}
};

public static void main(String args[])
{
    PolarGen cPolarGen = new PolarGen();
    double dEredmeny = cPolarGen.PolarGet();
    System.out.println(dEredmeny);
}
</para>
}
```

A program ellenőrzi, hogy van-e tárolt érték, amennyiben nincs, létrehozza azt a következőképpen: 2 darab -1 és 1 közötti számot hoz létre, majd egyet elvesz, és aztán veszi a négyzetösszegüket. Ez addig ismétlődik, amíg s nagyobb vagy egyenlő mint 1, vagy s egyenlő 0. Ezután felveszük a multiplier-t, ami az "s" -2-szeres logaritmus hányadosának négyzetgyöke. A tárolt érték multiplier szorozva v2-vel, utána pedig v1 szer multiplier értékét.

C++ -ban:

```
#include<cmath>
#include <cstdlib>
#include<ctime>
using namespace std;
class PolarGen {
public:
    PolarGen () {
        nincsTarolt = true;
        srand (time (NULL));
    }
    ~PolarGen () {
    }
    double kovetkezo ();
private:
    bool nincsTarolt;
    double tarolt;
};
double PolarGen::kovetkezo() {
    if (nincsTarolt) {
        double u1, u2, v1, v2, w;
```

```
do {
    u1 = rand() / (RAND_MAX + 1.0);
    u2 = rand() / (RAND_MAX + 1.0);
    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(int argc, char** argv) {
    PolarGen pg;
    for (int i = 0; i < 10; i++)
        std::cout << pg.kovetkezo() << std::endl;
    return 0;
}
```

2.2. Homokozó

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Itt annyi dolgunk van, hogy a C++ szintaktikát használó pointereket és referenciaikat át kell írni Java szintaktikának megfelelőekre.

```
public class LZWBinFa {

public LZWBinFa() {

    fa = gyoker;

}
```

```
public void egyBitFeldolgoz(char b) {  
  
    if (b == '0') {  
  
        if (fa.nullasGyermek() == null)  
        {  
  
            Csomopont uj = new Csomopont('0');  
  
            fa.ujNullasGyermek(uj);  
  
            fa = gyoker;  
        } else  
        {  
  
            fa = fa.nullasGyermek();  
        }  
    }  
    else {  
        if (fa.egyesGyermek() == null) {  
            Csomopont uj = new Csomopont('1');  
            fa.ujEgyesGyermek(uj);  
            fa = gyoker;  
        } else {  
            fa = fa.egyesGyermek();  
        }  
    }  
}  
  
public void kiir() {  
  
    melyseg = 0;  
  
    kiir(gyoker, new java.io.PrintWriter(System.out));  
}  
  
public void kiir(java.io.PrintWriter os) {  
    melyseg = 0;  
    kiir(gyoker, os);  
}  
  
class Csomopont {  
  
    public Csomopont(char betu) {  
        this.betu = betu;
```

```
    balNulla = null;
    jobbEgy = null;
}

;

public Csomopont nullasGyermek() {
    return balNulla;
}

public Csomopont egyesGyermek() {
    return jobbEgy;
}

public void ujNullasGyermek(Csomopont gy) {
    balNulla = gy;
}

public void ujEyesGyermek(Csomopont gy) {
    jobbEgy = gy;
}

public char getBetu() {
    return betu;
}

private char betu;

private Csomopont balNulla = null;
private Csomopont jobbEgy = null;

};

private Csomopont fa = null;

private int melyseg, atlagosszeg, atlagdb;
private double szorasosszeg;

public void kiir(Csomopont elem, java.io.PrintWriter os) {
    if (elem != null) {
        ++melyseg;
        kiir(elem.egyesGyermek(), os);
    }
}
```

```
        for (int i = 0; i < melyseg; ++i) {
            os.print(" ---");
        }
        os.print(elem.getBetu());
        os.print("(");
        os.print(melyseg - 1);
        os.println(") ");
        kiir(elem.nullasGyermek(), os);
        --melyseg;
    }
}

protected Csomopont gyoker = new Csomopont('/');
int maxMelyseg;
double atlag, szoras;

public int getMelyseg() {
    melyseg = maxMelyseg = 0;
    rmelyseg(gyoker);
    return maxMelyseg - 1;
}

public double getAtlag() {
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag(gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}

public double getSzoras() {
    atlag = getAtlag();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras(gyoker);

    if (atlagdb - 1 > 0) {
        szoras = Math.sqrt(szorasosszeg / (atlagdb - 1));
    } else {
        szoras = Math.sqrt(szorasosszeg);
    }

    return szoras;
}

public void rmelyseg(Csomopont elem) {
    if (elem != null) {
        ++melyseg;
```

```
if (melyseg > maxMelyseg) {
    maxMelyseg = melyseg;
}
rmelyseg(elem.egyesGyermek());
--melyseg;
}

public void ratlag(Csomopont elem) {
    if (elem != null) {
        ++melyseg;
        ratlag(elem.egyesGyermek());
        ratlag(elem.nullasGyermek());
        --melyseg;
        if (elem.egyesGyermek() == null && elem.nullasGyermek() == null) {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

public void rszoras(Csomopont elem) {
    if (elem != null) {
        ++melyseg;
        rszoras(elem.egyesGyermek());
        rszoras(elem.nullasGyermek());
        --melyseg;
        if (elem.egyesGyermek() == null && elem.nullasGyermek() == null) {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

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

public static void main(String args[]) {
    if (args.length != 3) {
        usage();
        System.exit(-1);
    }
}
```

```
String inFile = args[0];

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

try {

    java.io.FileInputStream beFile =
        new java.io.FileInputStream(new java.io.File(args[0]));

    java.io.PrintWriter kiFile =
        new java.io.PrintWriter(
            new java.io.BufferedWriter(
                new java.io.FileWriter(args[2])));

    byte[] b = new byte[1];

    LZWBinFa binFa = new LZWBinFa();

    while (beFile.read(b) != -1) {
        if (b[0] == 0x0a) {
            break;
        }
    }

    boolean kommentben = false;

    while (beFile.read(b) != -1) {

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

        if (b[0] == 0x0a) {
            kommentben = false;
            continue;
        }

        if (kommentben) {
            continue;
        }
    }
}
```

```
if (b[0] == 0x4e) // N betű
{
    continue;
}

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

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

binFa.kiir(kiFile);

kiFile.println("depth = " + binFa.getMelyseg());
kiFile.println("mean = " + binFa.getAtlag());
kiFile.println("var = " + binFa.getSzoras());

kiFile.close();
beFile.close();

} catch (java.io.FileNotFoundException fnfException) {
    fnfException.printStackTrace();
} catch (java.io.IOException ioException) {
    ioException.printStackTrace();
}

}
```

2.3. Yoda

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

3. fejezet

Helló, Liskov!

3.1. Liskov helyettesítés sértése

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Liskov elv azt mondja ki, hogy: Ha S osztály T osztály leszármazottja, akkor S szabadon behelyettesíthető minden olyan helyre (paraméter, változó, stb...), ahol T típust várunk. Példa sértésre:

```
public class Bird{ public void fly(){ } } public class Duck extends Bird{ } }
```

Ebben az esetben még minden jól működik, hiszen a kacsá a madár, azonban

```
public class Ostrich extends Bird{ } }
```

esetén a strucc már nem tudja használni a "fly" függvényt, annak ellenére hogy ő is madár.

Javított példa:

```
public class Bird{ } public class FlyingBirds extends Bird{ public void fly(){ } } public class Duck extends FlyingBirds{ } public class Ostrich extends Bird{ } }
```

3.2. Szülő-gyerek

írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)4

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Itt két (egy Java és egy C++) programot kellett elkészíteni, melyeken keresztül bemutatható, hogy minden két programban van egy "Parent", és egy "Child" osztály. Mindkét osztályon belül van egy-egy definiált metódus. A child, a parent leszármazottja, így a parent tulajdonságai öröklődnek, emiatt a child osztály segítségével el tudjuk érni a parent-ben definiált tulajdonságokat, fordítva viszont nem. A példák ezt mutatják be Java példa:

```
class Szulo
{
    public static void szulo_uzen()
    {
        System.out.println("Ez a szülő üzenete");
    }
}

class Gyerek extends Szulo
{
    public static void gyerek_uzen()
    {
        System.out.println("Ez a gyerek üzenete");
    }
}

public class szulo_gyerek
{
    public static void main(String[] args)
    {
        Szulo p = new Gyerek();
        p.gyerek_uzen();
    }
}
```

Természetesen a Java-s példa hibaüzenettel fut. C++ példa:

```
#include <iostream>
using namespace std;

class Szulo {
public:
    void szulo_uzen()
    {
        cout << "Ez a szülő üzenete\n";
    }
};

class Gyerek : public Szulo {
    void gyerek_uzen()
    {
        cout << "Ez a gyerek üzenete\n";
    }
};
```

```
    }
};

int main()
{
    Szulo * szulo = new Gyerek();
    szulo -> gyerek_uzen();
}
```

Itt egyszerűen egy olyan osztálydefiníciót kell írni, amely demonstrálja, hogy egy szülőn keresztül, csak a saját üzenete küldhető, egy hozzá tartozó gyereké már nem.

3.3. Ciklomatikus komplexitás

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Ciklomatikus komplexitás Thomas J. McCabe gráfelméletre alapuló számítása, amellyel a következőképpen fejezi ki számokkal egy forráskód komplexitását: $M = E - N + 2P$, ahol:

E: A gráf éleinek száma N: A gráfban lévő csúcsok száma P: Az összefüggő komponensek száma

A terminálban a "sudo apt-get install complexity" parancsal tudunk a komplexitás kiszámolására megfelelő programot letölteni, melyet utána "complexity --histogram --score --thresh=0 'file.c'" parancccsal tudunk használni. példa(torpedó játék kódját használva):

```
nesting depth reached level 5
Complexity Scores
Score | ln-ct | nc-lns| file-name(line): proc-name
      1      3      3  2.cpp(216): LoadShips
      1      7      7  2.cpp(187): UserInputAttack
      1      9      7  2.cpp(225): ResetBoard
      2     14     11  2.cpp(198): UserInputShipPlacement
      3     13     10  2.cpp(169): GameOverCheck
      8     35     22  2.cpp(239): DrawBoard
     13    102     64  2.cpp(62): main

Complexity Histogram
Score-Range Lin-Ct
  0-9          60 ****
 10-19         64 ****
                           *****

Scored procedure ct:      7
Non-comment line ct:    124
Average line score:      9
25%-ile score:          3 (75% in higher score procs)
50%-ile score:          13 (half in higher score procs)
75%-ile score:           0 (25% in higher score procs)
Highest score:           13 (main() in 2.cpp)
```

3.1. ábra. komplexitas

4. fejezet

Helló, Mandelbrot!

4.1. Reverse engineering UML osztálydiagram

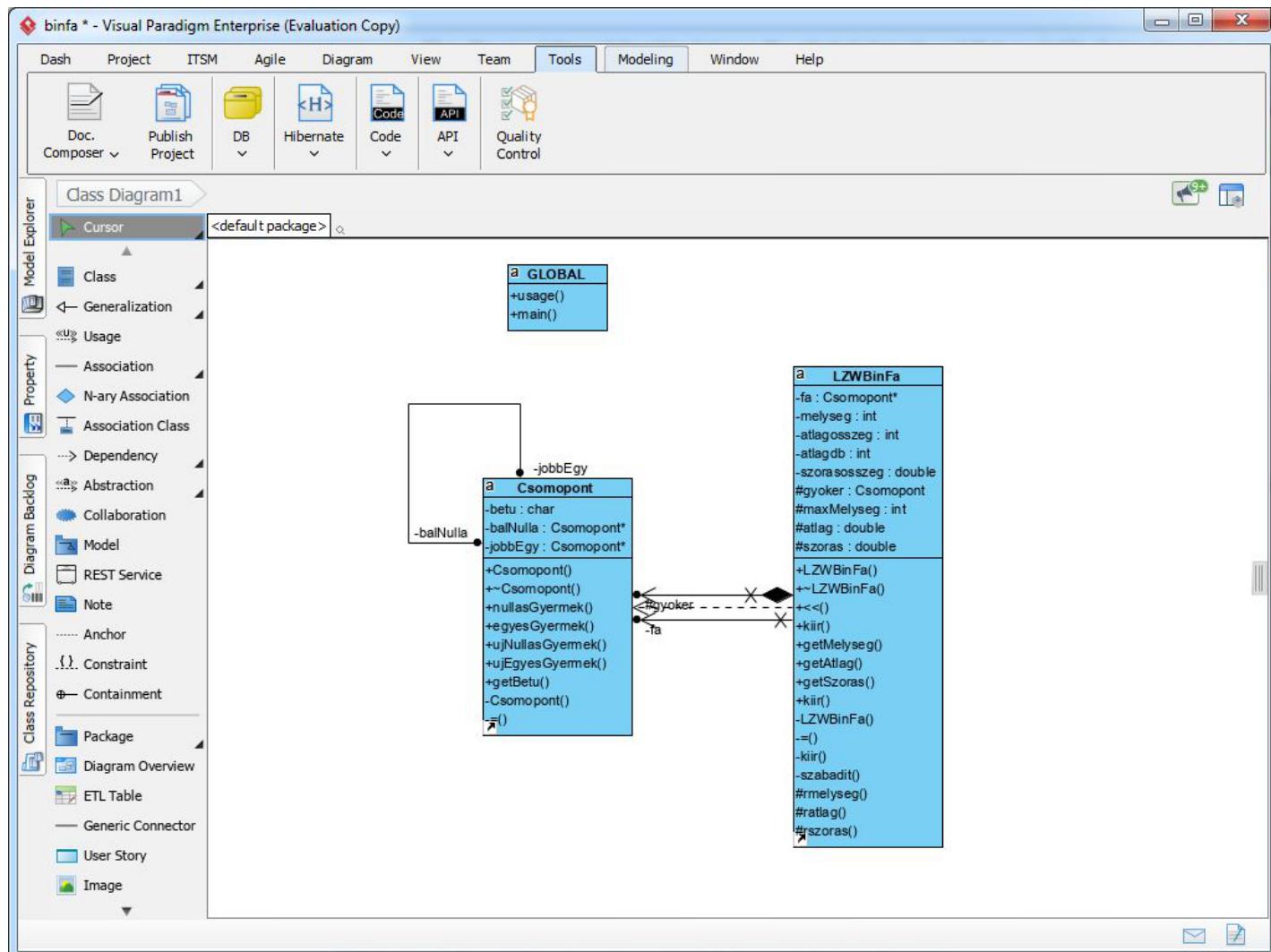
UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nlERIEOs. <https://arato.inf.unideb.hu/batfai.norbert/UD>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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Visual Paradigm nevű program "Instant Reverse" funkcióját használva a C++ forráskódot UML osztálydiagrammá alakítottam.:



4.1. ábra. ReverseUML

Az osztálydiagram az osztályok kapcsolatrendszerének összefoglalása. Az asszociációk két osztály, vagy két objektum közötti viszonyt fejeznek ki. Ebben az UML diagramban az aggregációt üres rombusz jelenti (gyenge tartalmazás), ebben az esetben, a rész az egészhez tartozik, de önmagában is létező entitás. A kompozíciót pedig a csúcsára állított teli rombusz jelöli (erős tartalmazás). Ebben az esetben a rész önmagában nem létezhet, csak az egész elemként.

4.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Továbbra is a visual paradigm segítségével, az alábbi kódokat kaptam:

```
#include <exception>
using namespace std;
#include "LZWBinFa.h"
#include "Csomopont.h"
LZWBinFa::LZWBinFa() {
}
void LZWBinFa::_LZWBinFa() {
    throw "Not yet implemented";
}
void LZWBinFa::_<(char aB) {
    throw "Not yet implemented";
}
void LZWBinFa::kiir() {
    throw "Not yet implemented";
}
int LZWBinFa::getMelyseg() {
    return this->_melyseg;
}
double LZWBinFa::getAtlag() {
    return this->_atlag;
}
double LZWBinFa::getSzoras() {
    return this->_szoras;
}
void LZWBinFa::kiir(std::ostream& aOs) {
    throw "Not yet implemented";
}
LZWBinFa::LZWBinFa(const LZWBinFa& aUnnamed_1) {
}
LZWBinFa& LZWBinFa::_(const LZWBinFa& aUnnamed_1) {
    throw "Not yet implemented";
}
void LZWBinFa::kiir(Csomopont* aElem, std::ostream& aOs) {
    throw "Not yet implemented";
}
void LZWBinFa::szabadit(Csomopont* aElem) {
    throw "Not yet implemented";
}
void LZWBinFa::rmelyseg(Csomopont* aElem) {
    throw "Not yet implemented";
}
void LZWBinFa::ratlag(Csomopont* aElem) {
    throw "Not yet implemented";
}
void LZWBinFa::rszoras(Csomopont* aElem) {
    throw "Not yet implemented";
}
}
```

A header fileok szinte teljesen megegyeznek az eredetivel.

4.3. BPMN

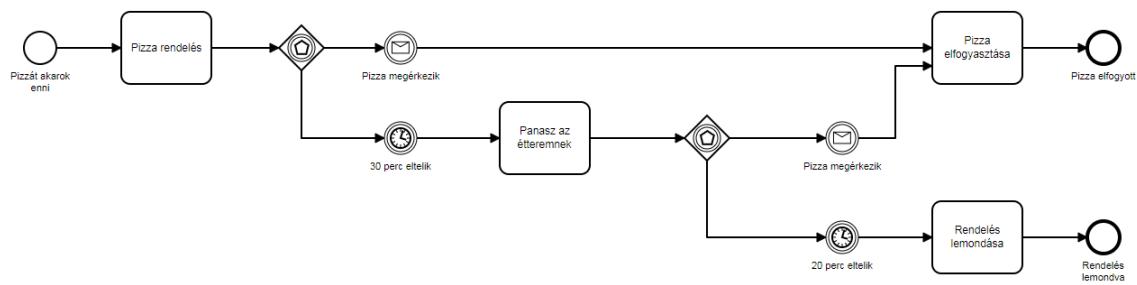
Rajzoljunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog34-47.pdf> (34-47 fólia)

Megoldás video:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Online elkészített ábra pizza rendelésről:



4.2. ábra. BPMN

5. fejezet

Helló, Chomsky!

5.1. Encoding

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Mandelbort Halmaz nagyító programját kellett fordítani és futtatni úgy, hogy minden a kódban és a fájl-névben meghagyjuk az ékezes karaktereket, ha azonban ezt módosítások nélkül próbáljuk meg, akkor hibaüzenetet fogunk kapni, mivel az UTF8-as kódolás számára ismeretlen karaktereket tartalmaz a kód. Ezt a problémát a -encoding kapcsoló használatával tudjuk kiküszöbölni, mellyel a kódolást ISO/IEC 8859-2 (Latin-2)-re állítva a java.lang api-ban a következő eredményt kapjuk:

5.2. I334d1c45

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 tettek meg, akkor írásd ki és magyarázd meg a használt struktúratömb memória foglalását!)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A leet osztályban található egy input string, ami tartalmazza a bekért szöveget, illetve egy output string vektort, ami a konvertált szöveget. A konvertálás a cipher függvényben zajlik. A bekért stringet nagybetűvé konvertáljuk, hogy ne kelljen különbséget tenni a kisbetűk és nagybetűk között. Ezután végezzük meg az input string minden karakterén és a karaktereknek mefelelő leet karaktert helyezzük az output vektorba. A main()-ben deklaráljuk a bekérés megszakításához szükséges stringet és boolt, ami minden fordítás után ellenőrzi, hogy szeretnék-e még fordítani. A konvertálás, pedig a konstruktorral történik. meghívásával hajtódiik végre.

```
import java.util.Scanner;
public class Leet {
    private static StringBuilder transform(String input, StringBuilder sb)
    {
        input=input.toUpperCase();
        for(int i=0;i<input.length();i++)
        {
            switch (input.charAt(i)) {
                case 'A' : sb.append("4");
                break;
                case 'B' : sb.append("13");
                break;
                case 'C' : sb.append("( ")");
                break;
                case 'D' : sb.append("| )");
                break;
                case 'E' : sb.append("3");
                break;
                case 'F' : sb.append("| =");
                break;
                case 'G' : sb.append("6");
                break;
                case 'H' : sb.append("| - |");
                break;
                case 'I' : sb.append("| ");
                break;
                case 'J' : sb.append(". ]");
                break;
                case 'K' : sb.append("| <");
                break;
                case 'L' : sb.append("1");
                break;
                case 'M' : sb.append(" | Y | ");
                break;
                case 'N' : sb.append("/V");
                break;
                case 'O' : sb.append("Ø");
                break;
                case 'P' : sb.append(" | O");
                break;
                case 'Q' : sb.append(" ( , ) ");
                break;
                case 'R' : sb.append("®");
                break;
                case 'S' : sb.append("$");
                break;
                case 'T' : sb.append("7");
                break;
            }
        }
    }
}
```

```
        break;
    case 'U' : sb.append("|_|");
        break;
    case 'V' : sb.append("V");
        break;
    case 'W' : sb.append("\\^/");
        break;
    case 'X' : sb.append(" }{ ");
        break;
    case 'Y' : sb.append("$\yen$");
        break;
    case 'Z' : sb.append("2");
        break;
    case ' ' : sb.append(" ");
        break;
    default: sb.append(input.charAt(i));
}
}
return sb;
}
public static void main(String[] args) {
    StringBuilder sb = new StringBuilder();
    Scanner scan = new Scanner(System.in);
    String input;
    System.out.println("Please enter the text that you want to 133tify" ←
    );
    input=scan.nextLine();
    System.out.println("The 1337 version of your input is: \n");
    sb=transform(input,sb);
    System.out.print(sb);
    System.out.println("\n");
}
}
```

5.3. Full screen

Készítsünk egy teljes képernyős Java programot! Tipp: https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanitok-javat/ch03.html#labirintus_jatek

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A felhasznált labirintus program forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/javat-tanitok-javat.zip> Használt displayek lekérdezése a getLocalGraphicsEnvironment() és a getDefaultScreenDevice() metódusokkal.

```
public class LabirintusJáték extends java.awt.Frame
    implements Runnable {
.
.
.
java.awt.GraphicsDevice graphicsDevice;
.
.
.
java.awt.GraphicsEnvironment graphicsEnvironment = java.awt. ←
    GraphicsEnvironment.getLocalGraphicsEnvironment();
.
.
.
graphicsDevice = graphicsEnvironment.getDefaultScreenDevice();
.
.
.
teljesKépernyősMód(graphicsDevice);
```

A teljesKépernyősMód-hoz tartozó kód:

```
public void teljesKépernyősMód(java.awt.GraphicsDevice graphicsDevice) ←
{
    int szélesség = 0;
    int magasság = 0;
    setUndecorated(true);
    setIgnoreRepaint(true);
    setResizable(false);
    boolean fullScreenTamogatott = graphicsDevice.isFullScreenSupported ←
        ();
    if(fullScreenTamogatott) {
        graphicsDevice.setFullScreenWindow(this);
        java.awt.DisplayMode displayMode
            = graphicsDevice.getDisplayMode();
        szélesség = displayMode.getWidth();
        magasság = displayMode.getHeight();
        int színMélység = displayMode.getBitDepth();
        int frissítésiFrekvencia = displayMode.getRefreshRate();
        System.out.println(szélesség
            + "x" + magasság
            + ", " + színMélység
            + ", " + frissítésiFrekvencia);
        java.awt.DisplayMode[] displayModes
            = graphicsDevice.getDisplayModes();
        boolean dm1024x768 = false;
        for(int i=0; i<displayModes.length; ++i) {
            if(displayModes[i].getWidth() == 1920
                && displayModes[i].getHeight() == 1080
```

```
    && displayModes[i].getBitDepth() == színMélység  
    && displayModes[i].getRefreshRate()  
    == frissítésiFrekvencia) {  
    graphicsDevice.setDisplayMode(displayModes[i]);  
    dm1024x768 = true;  
    break;  
}  
  
}  
  
if (!dm1024x768)  
    System.out.println("Nem megy az 1024x768.");
```

Ez a kód felelős a fullscreen mód beállításáért. Az ablak kereteit egyszerűen kivesszük, és kikapcsoljuk az átméretezhetőséget. Ezután lekérdezzük, hogy a fullscreen mód támogatott-e, ha igen, átadjuk a képernyő tulajdonságait a setFullScreenWindow() metódusnak. A támogatott felbontásokat egy tömbbe helyezzük, majd összevetve a jelenlegi felbontással, kiderül hogy támogatott e (amennyiben nem, hibaüzenetet kapunk)

6. fejezet

Helló, Stroustrup!

6.1. JDK osztályok

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Boost C++ program segítségével kell a JDK összes osztályát az src.zip-ből kilistázni. Ehhez a fénykard programot hívtam segítségül annyi különbséggel, hogy itt .java fájlokat fogunk megkeresni, kiíratni és megszmálni.

```
#include <iostream>
#include <vector>
#include <string>
#include <stdio.h>
#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>
#define GetCurrentDir getcwd
using namespace std;
vector<string> searchRootFolders (vector<string> folders);
void readClasses (string path, vector<string>& classes);
string GetCurrentWorkingDir( void ) {
    char buff[FILENAME_MAX];
    GetCurrentDir( buff, FILENAME_MAX );
    string current_working_dir(buff);
    return current_working_dir;
}
int main(int argc, char const *argv[])
{
    vector<string> roots = {
        GetCurrentWorkingDir() + "/" + "src"
```

```
};

vector<string> classes = searchRootFolders ( roots );
for(auto &i : classes)
{
    cout << i << endl;
}
cout << "Összesen " << classes.size() << " osztály van a src-zip-ben\
    n";
return 0;
}

void readClasses ( boost::filesystem::path path, vector<string>& classes)
{
    if ( is_regular_file ( path ) ) {
        std::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 ) )
            readClasses ( entry.path(), classes );
    }
    vector<string> searchRootFolders (vector<string> folders)
    {
        vector <string> classes;
        for ( const auto & path : folders)
        {
            boost::filesystem::path root ( path );
            readClasses ( root, classes);
        }
        return classes;
    }
}
```

A futtatáshoz "sudo apt-get install libboost-all-dev" -el kell letöltenünk a megfelelő fileokat, aztán fordításnál "-lboost_system -lboost_filesystem -lboost_file_options -std=c++14" parancsot használunk A program futtatása után megkapjuk hogy mennyi java osztály van a jdk-ben amit használunk.

6.2. Másoló-mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Prog 1-es LZW binfa-t használom a feladathoz.

A Binfa másoló konstruktora:

```
LZWBInFa ( const LZWBInFa & regi ) {
    std::cout << "LZWBInFa copy ctor" << std::endl;
    gyoker = masol(regi.gyoker, regi.fa);
}
```

Ezzel, a régi fa mintájára egy újat készítünk másolással, viszont nem default érték szerinti paraméter átadással, ugyanis abban az esetben a fa nem változna. Ahhoz hogy egy másik fát kapjunk saját konstruktorra van szükség:

```
Csomopont * masol ( Csomopont * elem, Csomopont * regifa ) {
    Csomopont * ujelem = NULL;
    if ( elem != NULL ) {
        ujelem = new Csomopont ( elem->getBetu() );
        ujelem->ujEgyesGyermek ( masol ( elem->egyesGyermek (), ←
            regifa ) );
        ujelem->ujNullasGyermek ( masol ( elem->>nullasGyermek (), ←
            regifa ) );
        if ( regifa == elem )
            fa = ujelem;
    }
    return ujelem;
}
```

Paraméterként megadva a régi fa gyökerét és a fát, csomópontonként rekurzívan átmásoljuk a gyoker-be, viszont minden node-nál ujelem néven új Csomopont-ot hozunk létre melynek így már memóracíme eltérő lesz, ezáltal a fánk is máshogyan néz ki.

A mozgató konstruktornak ezzel szemben az a lényege, hogy magát a régi fát mozgatjuk át egy másik memóriacímre, a régi címet üresen hagyva. Ehhez ezt a konstruktort használjuk:

```
LZWBInFa (LZWBInFa && regi) {
    std::cout << "LZWBInFa move ctor" << std::endl;
    gyoker = nullptr;
    *this = std::move(regi);
}
```

A gyökeret nullázzuk, ezután a move függvénynek átadjuk paraméterként a régi fát, azaz azt amit mozgatni szeretnénk és átmozgatjuk a regi-t az újba, a régit üresen hagyva.

6.3. Hibásan implementált RSA törése

Készítsünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: <https://arato.inf.unideb.hu/batfai.rsa> (71-73 fólia) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az RSA algoritmust 1976-ban fejlesztette ki Ron Rivest, Adi Shamir és Len Adleman, az RSA elvezetés a nevük kezdőbetűiből ered. Napjainkban is az egyik legelterjedtebb titkosító eljárás nagy biztonságának hála. A lényege, hogy két darab kulcs létezik, egy kódoló és egy dekódoló, melyek mindegyike egy-egy számpár. Az egyik a modulus és ez 512-4096 bit hosszú lehet, minnél nagyobb annál biztonságosabb a titkosítás. Egy elegendő nagyságú kulcs feltörésének jelenleg nincs ismert módja. A kodoló kulcs publikus viszont a dekódoló privát.

A kívánt szöveget ASCII kódjával alakítjuk a modPow függvényel, amiben e-edikre (exponent) emelünk m-mel (modulus) osztva és ennek a műveletnek a maradékát kapjuk. Amit fontos megjegyezni, hogy az e exponensünk, a kulcsunk egyik párja az nem más minta d kulcsunk inverze lesz, ez fogja nekünk biztosítani, hogy tudunk majd dekódolni is. A visszafejtés szinte ugyanez lesz, csak nem e-edikre emelünk, hanem d-edikre, a modulus marad ugyanaz, majd a kapott ASCII kódot visszaalakítjuk.

```
import java.util.Scanner;
public class RSA {
    public static void main(String[] args) {
        KulcsPar jszereplo = new KulcsPar();
        String szoveg;
        Scanner sc = new Scanner(System.in);
        System.out.println("Adja meg a titkosítani kívánt szöveget: ");
        szoveg=sc.nextLine();
        byte[] buffer = szoveg.getBytes();
        java.math.BigInteger[] titkos = new java.math.BigInteger[buffer.←
            length];
        for (int i = 0; i < titkos.length; ++i) {
            titkos[i] = new java.math.BigInteger(new byte[] {buffer[i]}));
            titkos[i] = titkos[i].modPow(jszereplo.e, jszereplo.m);
        }
        for (java.math.BigInteger t : titkos) {
            System.out.print(t);
            System.out.println();
        }
        for (int i = 0; i < titkos.length; ++i) {
            titkos[i] = titkos[i].modPow(jszereplo.d, jszereplo.m);
            buffer[i] = titkos[i].byteValue();
        }
        System.out.println("\n" + new String(buffer));
    }
    class KulcsPar {
```

```
java.math.BigInteger d,e,m;
public KulcsPar() {
    int meretBitekben = 700 * (int) (java.lang.Math.log((double) 10)
        / java.lang.Math.log((double) 2));
    java.math.BigInteger p = new java.math.BigInteger(meretBitekben, ←
        100, new java.util.
            Random());
    java.math.BigInteger q = new java.math.BigInteger(meretBitekben, ←
        100, new java.util.
            Random());
    m = p.multiply(q);
    java.math.BigInteger z = p.subtract(java.math.BigInteger.ONE). ←
        multiply(q.subtract(java.
            math.BigInteger.ONE));
    do {
        do {
            d = new java.math.BigInteger(meretBitekben, new java.util. ←
                Random());
            } while (d.equals(java.math.BigInteger.ONE));
        } while (!z.gcd(d).equals(java.math.BigInteger.ONE));
    e = d.modInverse(z);
}
}
```

6.1. ábra. RSA

7. fejezet

Helló, Gödel!

7.1. Gengszterek

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A Robotautó Világ bajnokság forráskódjában szereplő lambda kifejezések használatával történt std::sort-ról lesz szó. A lambda függvény segítségével többsoros névtelen függvények definiálhatók. Általános alakja: [] {} () Ahol a szögletes zárójel jelzi, hogy lambda kifejetés következik, a kapcsos zárójelbe kerül a függvény törzse, a kerek zárójel pedig függvényhívást jelenti. Ha ezek közül valamelyik hiányzik az autómatikusan kitöltsésre kerül. A forráskódban szereplő kód:

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

A fenti lambda függvény csak az std::vector beépítése után működőképes. Az std::sort segítségével folyik a sorbarendezés, melynek paramétere a lambda függvény. A .begin() és a .end() függvények alkalmazásával mondjuk meg, hogy mettől meddig szeretnénk számításba venni a sorbarendezést. Mi most a gangsters vektort használjuk erre, azaz a vektor első és utolsó indexét adjuk meg. A szögletes zárójelben két változó van: a this segítségével érhetjük el az osztály tagjait, és a cop objektumára lenne szükségünk. Az std::sort() {} kapcsos zárójelében annak az értékét kapjuk meg, hogy melyik gangster van közelebb.

7.2. STL map érték szerinti rendezése

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az STL a C++ nyelv Szabványos Sablonkönyvtára (Standard Template Library). Használatával fontos és elterjedt adatstruktúrákat és algoritmusokat építhetünk be a programunkba. 3 csoportja: konténerek, algoritmusok és iterátorok. A map konténerekben adatpárok tárolódnak, ahol az első tag a kulcs, a második pedig az adat. A kódcsipet a fenykard.cpp programból van.

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

f

Itt az látható, hogy a sort_map két paramétert vár: egy string és egy integer típusút. Ebből fogunk csinálni egy vektort, mégpedig úgy, hogy a két paramétert összepárosítjuk az std::pair függvényel, majd ezek összességét elnevezzük ordered-nek. A pair lehetővé teszi, hogy egyetlen objektumban két különböző "o" objektumot tároljunk, amelyek közül az elsőre a first, míg a másodikra a second névvel hivatkozhatunk. A push_back segítségével tesszük bele a kulcs/érték párokat az ordered vektorba. Ekkor jön a már megismert sorbarendezés. A szögletes zárójelben lévő egyenlőségjel segítségével az összes helyi változót érték szerint fogjuk elkapni és ez alapján rendezzük majd a vektort. A zárójelben lévő auto kulcsszóval pedig automatikusan fog megtörténni a típus-meghatározás.

8. fejezet

Helló, !

8.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>
Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az "ActivityEditor" futtatásához szükséges parancsok:

```
$ git clone https://github.com/nbatfai/future.git
$ cd future/cs/F6
$ javac ActivityEditor.java
$ java ActivityEditor --city=Debrecen --props=me.props,gaming.props, ←
    programming.props
```

A feladat szerint a ActivityEditor.java JavaFx programon, kellett javítást végezni. Feltűnt, hogy nem tudok új altevénységet létrehozni oda, ahova már létrehoztam korábban. Tehát egymás után nem lehetséges kettöt vagy többet is létrehozni. A megoldás az, hogy a mappa létrehozást végtelen ciklusba helyeztem, és a végéhez hozzáadtam egy sorszámot.

```
subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> {
    int i=1;
    while (true) {
        java.io.File file = getTreeItem().getValue();
        java.io.File f = new java.io.File(file.getPath() + System. ←
            getProperty("file.separator") + "Új altevénység"+ i);
        if (f.mkdir()) {
            javafx.scene.control.TreeItem<java.io.File> newAct
// = new javafx.scene.control.TreeItem<java.io. ←
File>(f, new javafx.scene.image.ImageView(actIcon));
            = new FileTreeItem(f, new javafx.scene.image. ←
ImageView(actIcon));
            getTreeItem().getChildren().add(newAct);
```

```
break;
} else {
++i;
//f.renameTo(new File(file.getPath() + System. ←
getProperty("file.separator") + uj_atevekenyseg+ ←
String.valueOf(i)));
// System.err.println("Cannot create " + f.getPath() +" ←
but we increased the i and we'll try again");
}
} );
}
```

8.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocaremulator/blob/master/justine/rc.cpp>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Tutor: [Kovács Ferencz](#)

Az OOCWC (Robocar World Championship) egy páréves platform, melynek célja volt a forgalomirányítási algoritmusok kutatása és a robotautók terjedésének vizsgálata. A "<https://github.com/nbatfai/robocar-emulator>" Robocar City Emulator lehetővé tette volna fejlesztők számára új modellek és elméletek tesztelését. A debreceni Justine prototípus része a CarLexer, melynek sscanf függvényét kell feldolgoznunk.

A sscanf függvény addig dolgozza fel a formázott stringből az adatokat, amíg meg nem kapja a Gangster négy argumentumát. A %n az olvasott karakterek számát rögzíti, az nn változóba kerülnek tehát az összesen beolvasottak száma. A data segítségével tudjuk olvasni a még beolvasatlan adatokat. Ezt a pointert a beolvasott karakterek méretével toljuk el, így a data+nn az olvasatlan rész elejére fog mutatni. Az <OK %d %u %u %u> alak teljesülése esetén egy gengszter összes adata beolvasásra került, tehát létrehozunk egy új objektumot és belehelyezzük a gangster vektorba.

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

8.3. SamuCam

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A "https://github.com/nbatfai/samucam" linken található Samucam kísérlet lényege, hogy a program képes felismerni, tanulmányozni az emberi arcokat videók és fotók alapján. Ugyan az emberi arcok megkülönböztetése nehézkessé válik, az arcfelismerés jól működik az OpenCV könyvtár segítségével. A videók és fotók megmutatásához használhatjuk Androidos készülékünk kameráját, de jelen esetben én egy otthoni laptop kameráját használtam.

A SamuCam.cpp-ben látható, hogy a videoCapture open függvénye nyitja meg a VideoStream-et. Mivel alapból telefonos használatra íródott a program, ezért a VideoStream helyett 0-t kell írnunk, ha webkamerával szeretnénk dolgozni. Ezután beállítjuk a kamerakép szélességét, magasságát és FPS értékét.

```
SamuCam::SamuCam ( std::string videoStream, int width = 176, int height = ←
    144 )
: videoStream ( videoStream ), width ( width ), height ( height )
{
    openVideoStream();
}
SamuCam::~SamuCam ()
{
}
void SamuCam::openVideoStream()
{
    videoCapture.open ( 0 );
    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
    videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
    videoCapture.set ( CV_CAP_PROP_FPS, 10 );
}
```

A CascadeClassifier alapvetően tárgyak felismerését segíti, jelen esetben pedig ez teszi lehetővé a kamera-képen látható arcok feldolgozását. A helyes működés érdekében le fogok szedni a GitHub-ról egy frontal face XML-t, melyben az értékek a kamerával szemben elhelyezett arcképek felismerését biztosítja.

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

Képkockánként kerül beolvasásra a kamerakép a read függvényben. Ha kap egy képkockát, akkor első lépésként átméretezi, majd szürkére átszínezi. Az equalizeHist függvény kiegyenlíti a szürke képkocka hisztogramját.

```
while ( videoCapture.read ( frame ) )
{
```

```
if ( !frame.empty() )
{
    cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0, cv::INTER_CUBIC );
    std::vector<cv::Rect> faces;
    cv::Mat grayFrame;
    cv::cvtColor ( frame, grayFrame, cv::COLOR_BGR2GRAY );
    cv::equalizeHist ( grayFrame, grayFrame );
```

A detectMultiScale függvény segítségével történik a képkockán lévő arcok keresése. Ha talált egy arcot, akkor az alapján létrehozunk egy QImage-t. A faceChanged egy signal, bekövetkezte után az arc köré rajzolunk egy keretet, és egy újabb QImage-t készítünk. Ha pedig a webcamChanged signal bekövetkezett, akkor 80 ms-t követően következhet egy újabb képkocka beolvasása.

```
faceClassifier.detectMultiScale ( grayFrame, faces, 1.1, 3, 0, cv::Size ( 60, 60 ) );
if ( faces.size() > 0 )
{
    cv::Mat onlyFace = frame ( faces[0] ).clone();
    QImage* face = new QImage ( onlyFace.data,
                                onlyFace.cols,
                                onlyFace.rows,
                                onlyFace.step,
                                QImage::Format_RGB888 );
    cv::Point x ( faces[0].x-1, faces[0].y-1 );
    cv::Point y ( faces[0].x + faces[0].width+2, faces[0].y + faces[0].height+2 );
    cv::rectangle ( frame, x, y, cv::Scalar ( 240, 230, 200 ) );
    emit faceChanged ( face );
}
QImage* webcam = new QImage ( frame.data,
                             frame.cols,
                             frame.rows,
                             frame.step,
                             QImage::Format_RGB888 );
emit webcamChanged ( webcam );
}
QThread::msleep ( 80 );
}
```

A SamuCam futtatásához először le kell szednünk a GitHub projektet, majd pedig a korábban említett XML fájlt. A Qt keretrendszer segítségével létrehozzuk a Makefile-t, ezután pedig indíthatjuk is a programot az alábbi módon:

```
git clone https://github.com/nbatfai/SamuCam.git
cd SamuCam/
wget https://github.com/Itseez/opencv/raw/master/data/lbpcascades/
     lbpcascade_frontalface.xml
git checkout vInitialHack
~/Qt/5.9.8/gcc_64/bin/qmake SamuLife.pro
```

```
make  
./SamuCam
```

9. fejezet

Helló, Schwarzenegger!

9.1. Port scan

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
public class Scan {  
    public static void main(String[] args) {  
        for(int i=0; i<65535; ++i)  
            try {  
                java.net.Socket socket = new java.net.Socket(args[0], i);  
                System.out.println(i + " figyeli");  
                socket.close();  
            } catch (Exception e) {  
                System.out.println(i + " nem figyeli");  
            }  
    }  
}
```

A try-catch-ben egy új socket-et akarunk létrehozni és, ha sikerül akkor csatlakozunk rá, kiírjuk, hogy figyeli az i-edik portot és bezárjuk azt. Ha nem sikerült csatlakozni rá annak több oka is lehet, az egyik az, hogy a portot nem figyeli senki és erre kapunk egy ConnectException errorrt. A másik lehetőség, hogy miért nem sikerült csatlakozni az, hogy a host neve rosszul van megadva. A harmadik lehetőség, hogy a portszám amire akarunk csatlakozni kívül esik a 0-65535 intervallumon amelyet megadtunk.

9.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
public aspect MyAspect {  
void around(char b, LZWBInFa f):  
target(f) && call(void egyBitFeldolg (char))&& args(b)  
{  
if (b == '0')  
{  
if (f.fa.nullasGyermek() == null)  
{  
LZWBInFa.Csomopont uj = f.new Csomopont('0');  
f.fa.ujNullasGyermek(uj);  
f.fa = f.gyoker;  
f.fa = f.fa.nullasGyermek();  
}  
else {  
f.fa = f.fa.nullasGyermek();  
}  
}  
else {  
if (f.fa.egyesGyermek() == null) {  
LZWBInFa.Csomopont uj = f.new Csomopont('1');  
f.fa.ujEgyesGyermek(uj);  
f.fa = f.gyoker;  
f.fa = f.fa.egyesGyermek();  
}  
else {  
f.fa = f.fa.egyesGyermek();  
}  
}  
}  
}  
}
```

A programunk annyit csinál, hogy az eredeti egyBitFeldolg függvény helyett, az aspect osztályban lévő hívódik meg. Ennek oka az around függvény, ami két paraméterként kap egy betűt és egy LZWBInFa objektumot. Az around függvény paraméter listájában látható, hogy a LZWBInFa a cél objektum a char pedig a metódus paramétere. Ezek után programunk úgy működik mint a rendes bitfeldolgozás a binfában annyi különbséggel, hogy ha beszúrtunk egy elemet akkor a gyökérre állítás után lépünk egy elemet előre, ha 0 elemet szúrtunk be akkor a gyökér nulladik elemére lépünk, ha 1-et szúrtunk be akkor a gyökér 1-es elemére lépünk.

9.3. Junit teszt

A https://progpater.blog.hu/2011/03/05/labormeres_oththon_avagy_hogyan_dolgozok_fel_egy_pedat_poszt_kézzel_számított_mélységét_és_szórását_dolgozd_be_egy_Junit_tesztbe (sztenderd védési feladat volt korábban).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy a Tanár úr által készített posztnak kézzel számított mélységet és szórást dolgozzuk be egy JUnit tesztbe. Vettünk egy rövid teszt sorozatot (01111001001001000111), ezt betesszük inputként a LZWBInFa.java fájl egyBitFeldoz() metódusának, ami karakterenként feldogozza a megadott sztringet. Ezután az assertEquals metódussal ellenőrizzük, hogy az értékek ugyanazok-e, mint az alap esetben. Ha ugyanaz jön ki, akkor jó a program. Egyébként ez a metódus 3 paramétert kap. Az első érték az, amit Tanár úr számolt, a második, amit a Binfa adott eredményül, a harmadik paraméter a megengedett eltérés az első két paraméter között.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.Test;
public class BinfaTest {
LZWBInFa binfa = new LZWBInFa();
@Test
public void test() {
for(char c: "01111001001001000111".toCharArray()) {
binfa.egyBitFeldolg(c);
System.out.println(c);
}
assertEquals(4.0, binfa.getMelyseg(), 0.000001);
assertEquals(2.75, binfa.getAtlag(), 0.001);
assertEquals(0.95742, binfa.getSzoras(), 0.0001);
}
```

10. fejezet

Helló, Calvin!

10.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13/hello_sbol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy próbáljuk ki az MNIST-et, illetve készítsünk hozzá egy saját 8-ast tesztelésre. A MNIST egy olyan program ami adatbázisból való tanítás után felismeri a kézzel írott egyjegű számokat. A programhoz Python-t és az ehhez szükséges Tensorflow library-t használjuk. A Tensorflow az adatfolyam programozáshoz kínál nekünk lehetőségeket. Az adatfolyam programozásban az egyes programrészeket, mint irányított gráfok értelmezzük és ezek között pedig adatfolyamok áramlanak.

A programban dekalralunk egy függvényt a saját 8-ast ábrázoló képünk beolvasására. A file változóba a read_file() függvénnyel olvassuk be a képet, majd a decode_png() függvénnyel, ezt dekódoljuk unit8-as vagy 16-os tensorrá.

```
def readimg():
    file = tf.io.read_file("sajat8a.png")
    img = tf.image.decode_png(file, channels=1)
    return img
```

A main függvényben beolvassuk az MNIST adabázis képeit majd létrehozunk egy modellt. A modell tensorokból áll amik, olyan adatszerkezetek, amik (ebben az esetben) mátrixokat tartalmaznak. A modellben az x egy Placeholder típusú tensor, ami akkor fog értéket kapni, mielőtt a kód lefut. Ezután jön két változó típusú tensor, a W tartalmazza a súlyokat, a b pedig a bias-t, tehát a hibás következetéseket. Az yban vesszük az x és W tensor mátrixszorzatát a matmul() függvénnyel, majd ehhez hozzáadjuk a b-t.

```
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot= ←
    True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
```

```
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
```

A cross_entropy-ban vesszük a y logits-ok és y label-ök közötti keresztszorzatot, majd ennek a tensornak az elemeinek számoljuk ki a dimenziók közötti átlagát a reduce_mean() függvényvel. Tanításnál ezt az átlagot próbáljuk minimalizálni a gradient descent algoritmus segítségével. Ez a programban a train.GradientDescentOptimizer.minimize() függvényként jelenik meg.

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y, labels=y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Ezután elkezdjük a tanítást az MNIST képeivel egyszerre 100-at fog feldolgozni, és ezt a lépést ismételjük meg 10-szer, és kiírjuk hány százaléknál jár a tanítás.

```
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
        print(i/10, "%")
```

Végül teszteljük az adatázis 42. elemére ami egy 4-es, illetve a saját kezűleg rajzolt 8-asra is.

A progpater-es forrás működéséhez néhány függvényt át kellett írni, mert depraceted-ek voltak.

A kép dekódolásánál meg kellett adni egy channels=1 argumentumot, ez a színcsatornák számát adja meg. Az érték 1 mert a kép szürkeárnyalatos.

```
img = tf.image.decode_png(file, channels=1)
```

A keresztszorzat létrehozásánál pedig a softmax_cross_entropy_with_logits függvénynél argumetumként meg kellett adni a logits és labels értékeket.

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y, labels=y_))
```

10.2. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a https://arato.inf.unideb.hu/batfai.norb8_folia.pdf!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy futtassuk le az MNIST mély verzióját és ismertessük fel vele a saját kézzel írott számunkat.

Ahhoz, hogy a program az új TensorFlow-val is működjön ki kell cserélnünk a legtöbb függvény előtagját tf.compat.v1.-re, mivel ezek az új verzióban ebbe a modulba kerültek

Ahhoz, hogy saját képet is felismerjen bele kellett írni az előző feladatban használt readimg() függvényt, ami be fogja olvasni a saját képunket.

```
def readimg():
    file = tf.compat.v1.read_file("sajat8a.png")
    img = tf.image.decode_png(file, 1)
    return img
```

Az alábbi függvénytel pedig a beolvasott képet fogjuk átkonvertálni 28*28-as méretűre és grayscale-re. Ez a paramatérekben is látszik az utolsó paraméter 1 amiért a képünk grayscale, ha RGB színtérű lenne akkor 3-at kéne megadni.

```
with tf.name_scope('reshape'):
    x_image = tf.reshape(x, [-1, 28, 28, 1])
```

Létrehozzuk a súlyokat és a biast tartalmazó Tensorokat, az első rétegben itt a képünkötől 32 képet készít a program.

```
with tf.name_scope('conv1'):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
```

A második rétegben 32 képre még helyezünk 64 filtert ami által kapunk egy 32x64 képet. Ezek után fog átmenni két teljesen kapcsolt rétegen ahol a súlyokat és biast beállítjuk, majd ezután történik a klasszifikáció tehát felismeri a képet. Az első feladat példájához hasonlóan is beolvassuk a mintaképeket, majd létehozunk egy modellt. Létrehozunk egy keresztszorzatot a logits és labels elemek között, majd ezeknek az átlagát vesszük. Az optimalizálás során most az Adam algoritmust használjuk a GradientDescent helyett.

```
# Import data
tf.compat.v1.disable_eager_execution()
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

# Create the model
x = tf.compat.v1.placeholder(tf.float32, [None, 784])

# Define loss and optimizer
y_ = tf.compat.v1.placeholder(tf.float32, [None, 10])

# Build the graph for the deep net
y_conv, keep_prob = deepnn(x)

with tf.name_scope('loss'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                          logits=y_conv)
cross_entropy = tf.reduce_mean(cross_entropy)

with tf.name_scope('adam_optimizer'):
```

```
train_step = tf.compat.v1.train.AdamOptimizer(1e-4).minimize( ←  
    cross_entropy)
```

Ezt követi a tanítás, melynek során 20000 mintaképet használunk és ezeket 50-essével olvassuk be. Illetve 100 képenként kiírjuk hány képet olvassot már be a program és hány százalékos a pontosság. Ez a verzió a sima MNIST-nél 20-szor több mintával dolgozik, ezért sokkal pontosabb de a tanítás is jóval több időt igényel. Az én esetben ez körülbelül 20-25 perc volt.

```
with tf.compat.v1.Session() as sess:  
    sess.run(tf.compat.v1.global_variables_initializer())  
    for i in range(20000):  
        batch = mnist.train.next_batch(50)  
        if i % 100 == 0:  
            train_accuracy = accuracy.eval(feed_dict={  
                x: batch[0], y_: batch[1], keep_prob: 1.0})  
            print('step %d, training accuracy %g' % (i, train_accuracy))  
            train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})  
  
        print('test accuracy %g' % accuracy.eval(feed_dict={  
            x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

10.3. CIFAR-10

Az alap feladat megoldása, +saját fotót is ismerjen fel, https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tf_tutorial_peldabol

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.4. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

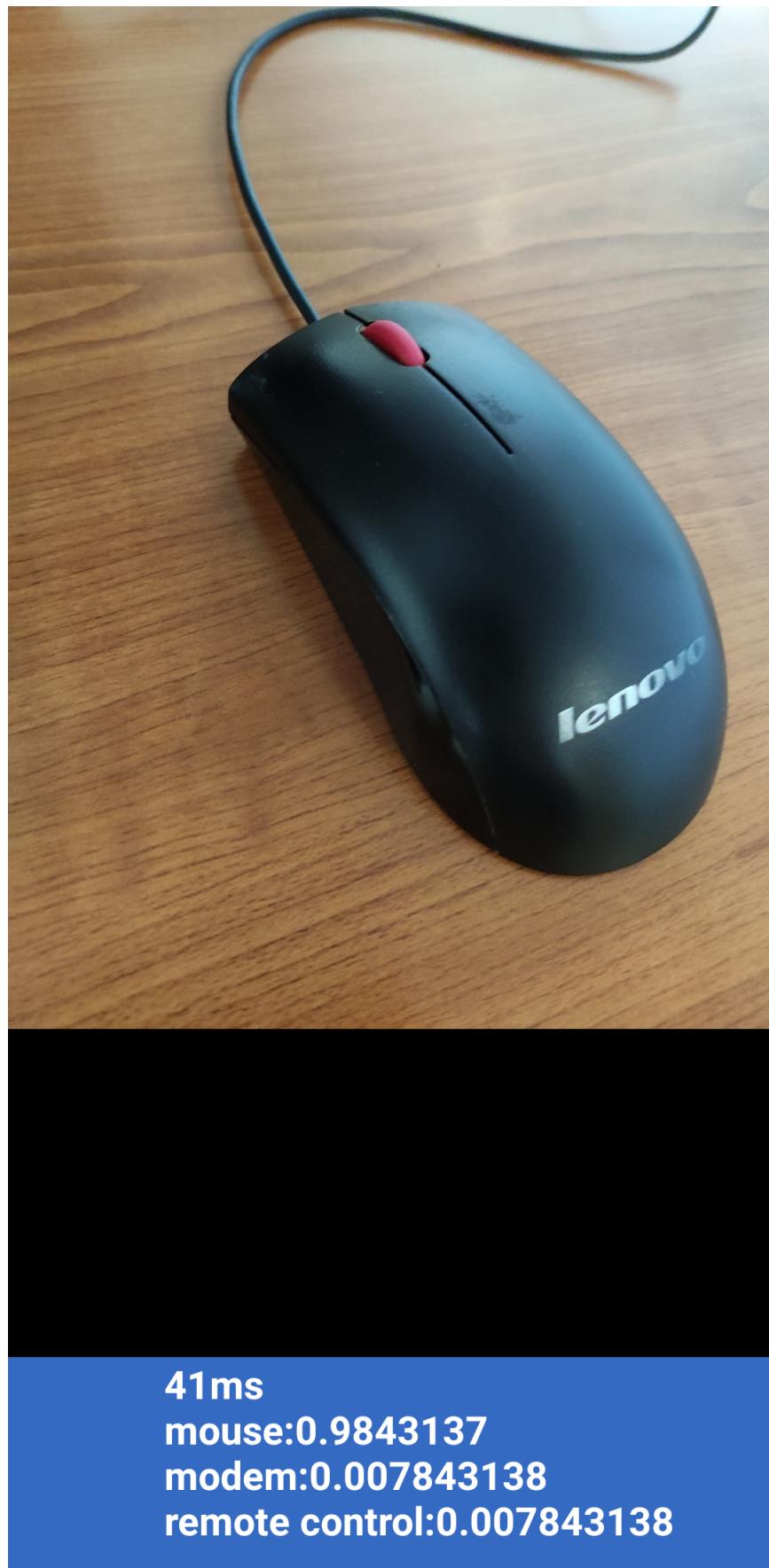
Megoldás videó:

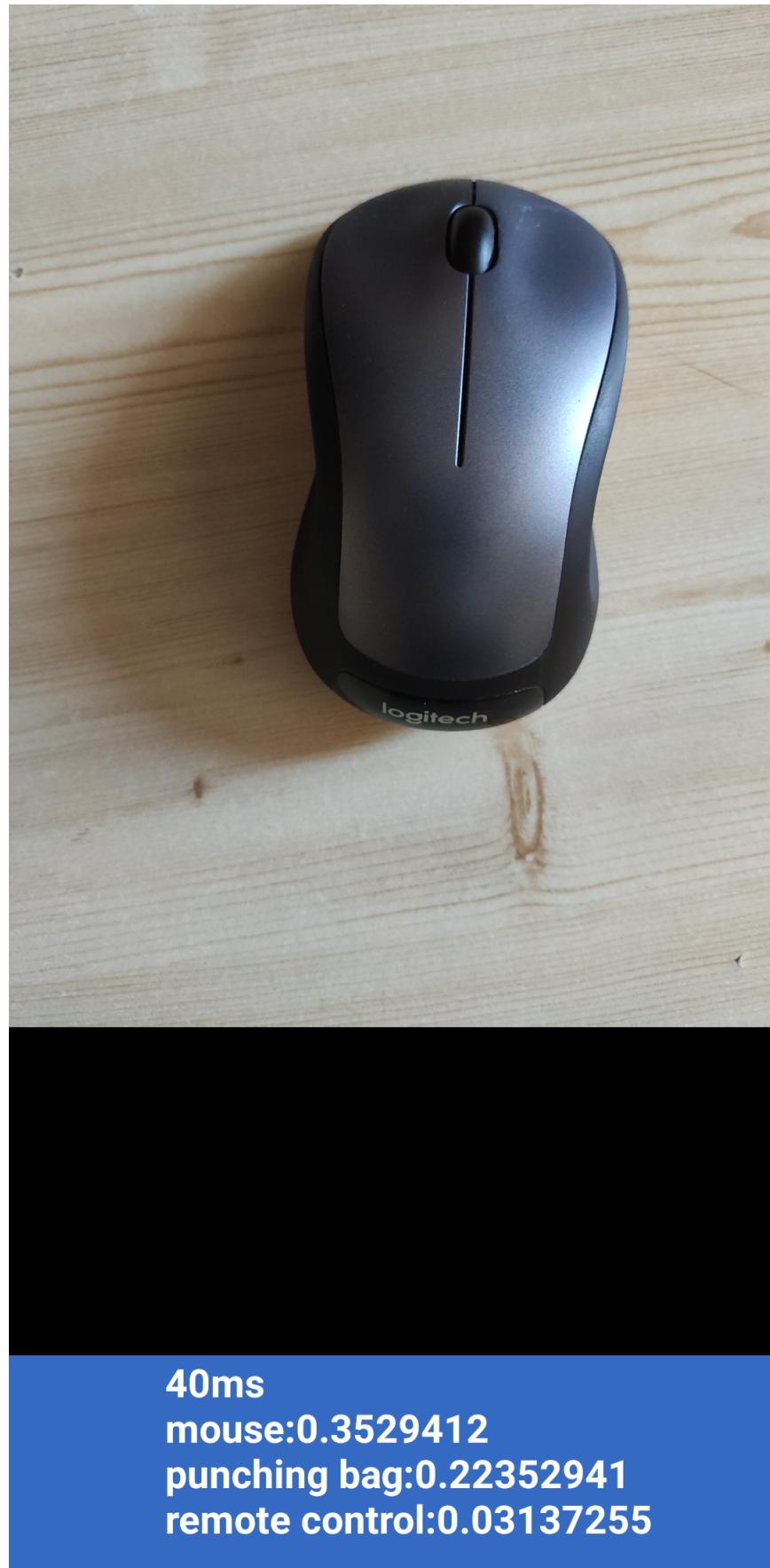
Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat az volt, hogy próbáljuk ki a TensorFlow objektum detektáló appját androidra. Ehhez először le kell klónozni a TensorFlow android-os repóját a következő linkről: <https://github.com/tensorflow/tensorflow/tree/r>

Az oldalról letöltött apk-t használva sikerült pár tárgyat pontosan meghatározni.







10.5. SMNIST for Machines

Készíts saját modellt, vagy használj meglévőt, lásd: <https://arxiv.org/abs/1906.12213>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.6. Minecraft MALMO-s példa

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lásd pl.: <https://youtu.be/bAPSu3Rndl8>, https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innentol_mi, <https://bhaxor.blog.hu/2018/10/28/minecraft>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11. fejezet

Helló, Berners-Lee!

11.1. C++ és Java összehasonlítás

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

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

A Java a C és C++ nyelvek által inspirált programozási nyelv, azonban könnyen észrevehetőek eltérések közöttük. Az egyik legfontosabb különbség abból ered, hogy a Java a biztonságra nagyobb hangsúlyt fektet, ez azonban hátrányt jelent ha a Java-t a C/C++ sebességével vetjük össze. Természetesen a nyelvek között egyértelműen felfedezhetőek hasonlóságok, például: - Megegyező módon történő kommentkezelés - A C++ függvények és utasítások jelentős része szintaktikailag megfelel a Java programokban elvártaknak. - A végrehajtás a main() függvényrel kezdődik. A legmeghatározóbb működésbeli különbségek egyike az, hogy a Java teljes mértékben objektumorientált nyelv, nem találhatók meg a pointerek, csak referenciaiák ellentétben a C++-al, amely többparadigmás nyelv, így ott több egyéb módszer mellett ezek az eszközök is használhatók. További különbség, hogy a Java programok kész állapotban is futtathatók más gépeken, ehhez azonban szabványos virtuális gépre van szükség. C++-ban ez csak a forráskód átvitelével lehetséges, kész program futtatására magában nincs lehetőség. Az öröklés tekintetében nagy különbség, hogy a C++ minden új öröklődési fát készít, a Java esetében azonban egyetlen fát vizsgál újra. Ebből eredő különbség, hogy a Java nem képes többszörös öröklődés kezelésére, ellentétben a C++-al ami azt alapból támogatja a feljebb említett különbség miatt.

11.2. Python

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

A Python egy általános célú, magas szintű programozási nyelv, melyet Guido van Rossum fejleszett ki 1990-ben. Számos pozitív tulajdonsága közül érdemes kiemelni, hogy objektumorientált és platformfüggetlen. Leginkább prototípus-fejlesztés-re használt, mivel ideális hibák javítására, mivel más nyelveken megírt modulokkal is együtt tud működni, de használható kisebb és egyszerűbb alkalmazások készítésére is. Ezek a pozitívumok segítették elő a Python S60 implementáció kifejlesztését, amely mobileszközökre specializálódott. Prototípus alkalmazások tesztelését jelentősen megkönnyíti a Python nyelv használata,

ugyanis nincs szükség fordításra, amely egy bonyolultabb program esetén többszöri előfordulása miatt elégé hosszadalmas folyamat, ellentétben például a fentebb tárgyalt C++ vagy Java nyelvekkel. A Pythonnal való munkát továb könnyíti a behúzás alapú szintaktika, amellyel az egyformán behúzott kód részleteket tudjuk csoportosítani. Természetesen ennek is megvannak a maga szabályai, például: - A script első utasítása nem lehet behúzott. - A tab és a szóköz nem használható egyszerre. - Egy blokk végét kisebb behúzású sor jelzi. (típusok és változók hozzáadása később)

III. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

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

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