

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 19, v. 0.0.4

Copyright © 2019 Bakos Bálint

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	Bakos, Bálint	2019. október 22.	

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

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, Berners-Lee!	5
2.1. Olvasónapló: C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyel- ven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I--II.II.	5
2.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobil- programozásba. Gyors protot ípus fejlesztés Python és Java nyelven	6
3. Helló, Arroway!	7
3.1. OO szemlélet	7
3.2. "Gagyí"	14
3.3. Yoda	15
3.4. Kódolás from scratch	17
4. Helló, Liskov!	22
4.1. Liskov helyettesítés sértése	22
4.2. Szülő-gyerek	24
4.3. Anti OO	26
4.4. Ciklomantikus komplexitás	26

5. Helló, Mandelbrot!	28
5.1. Reverse engineering UML osztálydiagram	28
5.2. Forward engineering UML osztálydiagram	29
5.3. BPMN	31
5.4. BPEL Helló, Világ!	31
6. Helló, Chomsky!	32
6.1. Encoding	32
6.2. I334d1c4^5	34
6.3. Full screen	37
6.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció	41
7. Helló, Stroustrup!	43
7.1. JDK osztályok	43
7.2. Másoló-mozgató szemantika és Összefoglaló	45
7.3. Változó argumentumszámú ctor	52
8. Helló, Gödel!	56
8.1. Gengszterek	56
8.2. C++11 Custom Allocator	56
8.3. STL map érték szerinti rendezése	56
8.4. Alternatív Tabella rendezése	56
8.5. Prolog családfa	56
8.6. GIMP Scheme hack	56
9. Helló, Valaki!	57
9.1. FUTURE tevékenység editor	57
9.2. OOCWC Boost ASIO hálózatkézelése	57
9.3. SamuCam	57
9.4. BrainB	57
9.5. OSM térképre rajzolása	57
10. Helló, Schwarzenegger!	58
10.1. Port scan	58
10.2. AOP	58
10.3. Android Játék	58
10.4. Junit teszt	58

11. Helló, Calvin!	59
11.1. MNIST	59
11.2. Deep MNIST	59
11.3. CIFAR-10	59
11.4. Android telefonra a TF objektum detektálója	59
11.5. SMNIST for Machines	59
11.6. Minecraft MALMO-s példa	59
 III. Irodalomjegyzék	 60
11.7. Általános	61
11.8. C	61
11.9. C++	61
11.10Lisp	61

Táblázatok jegyzéke

4.1. Összehasonlítás	26
--------------------------------	----

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]

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

Mi a programozás?

Milyen doksikat olvassak el?

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

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, Berners-Lee!

Olvasónapló: C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven és Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.II.

A C++ valamint a Java nyelv is magasszintű programozási nyelvnek számít. Mindkettő objektumorientált, azonban mivel a C++ nyelv hamarabb alakult ki így kisebb-nagyobb eltérések találhatók a két nyelv között.

A C++ a C nyelvtől örökölt gépközei konstrukciókat, ebből adódik sebessége. A Java nyelv pedig a C++-ból vett át sok mindent. Azonban van egy lényeges eltérés, hogy az Java nyelv a mutatók helyett referenciákat használ, így biztonságosabb, megbízhatóbb programokat lehet írni. Valamint a programok hordozhatósága is eltér. Például ha egy Linuxon írt C++ kódot akarunk átvinni Windowsra az nem biztos, hogy működni fog, de ha egy Java kódot akarunk futtani máshol az jól fog szuperálni feltéve, hogy van JVM(Java Virtual Machine) a gépen. Mivel ez a Java bájt kódot fogja futtani, így ez platformfüggetlen lesz.

Szintaktikában is találunk eltéréseket a két nyelv között.

Hasonlítsuk össze mondjuk ezt a két Hello world! programot.

C++-ban:

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Java-ban:

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

```
        System.out.println("Hello World!");  
    }  
}
```

Amint látható a Java nagyrészen osztályalapú. Az osztályoknak különböző elérése lehet: public, protected, private. Az osztályokon belül létrehozhatunk változókat, függvényeket.

Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus fejlesztés Python és Java nyelven

A Python támogatja a funkcionális valamint az imperatív nyelveket is. Legfőbb jellemzője, ami teljesen eltér a többi magasszintű programozási nyelvtől, hogy behúzásalapú a szintaxisa, tehát semmilyen kapcsos zárójelre vagy explicit kulcsszóra nincs szükség. Valamint a sorok végén már megszokott pontosvessző sem kell.

```
if 1 < 2  
    print("Nagyobb!")  
else:  
    print("Kisebb!")
```

A Python egy objektumorientált nyelv, tehát minden adatot objektumok reprezentálnak. A változók típusának explicit megadására sincs szükség, a rendszer futási időben dönti el a változók típusát.

Ilyen egyszerűen néz ki Pythonban egy deklarálás:

```
name = "Ádám"
```

A nyelvben definiálhatunk osztályokat is és ezeknek példányai az objektumok. Az osztályok attribútumai lehetnek objektumok vagy függvények is.

3. fejezet

Helló, Arroway!

OO szemlélet

A polártranszformációs generátor egy széles körben elterjedt random generátor. Olyannyira elterjedt formája ez a random szám generálásnak, hogy a `Java.util.Random` osztály is ezt a módszert alkalmazza.

Íme a Java kód teljes egészében:

```
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 következő sorokban részenként magyarázom el mi történik a kódban.

```
boolean nincsTarolt = true;
double tarolt;
```

A PolárGenerátor classban létrehozunk két változót. Az egyik boolean típusú, amely azt fogja megmondani, hogy éppen van-e tárolt értékünk. A másik maga a tárolt értéket tartalmazza, amit korábban kiszámítottunk.

```
public PolarGenerator() {
    nincsTarolt = true;
}
```

Ez a class publikus konstruktora, amely a nincsTarolt-at igazra állítja. Erre azért van szükség, hogy tudjuk éppen van-e tárolt érték vagy nincs.

```
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;
}
}
```

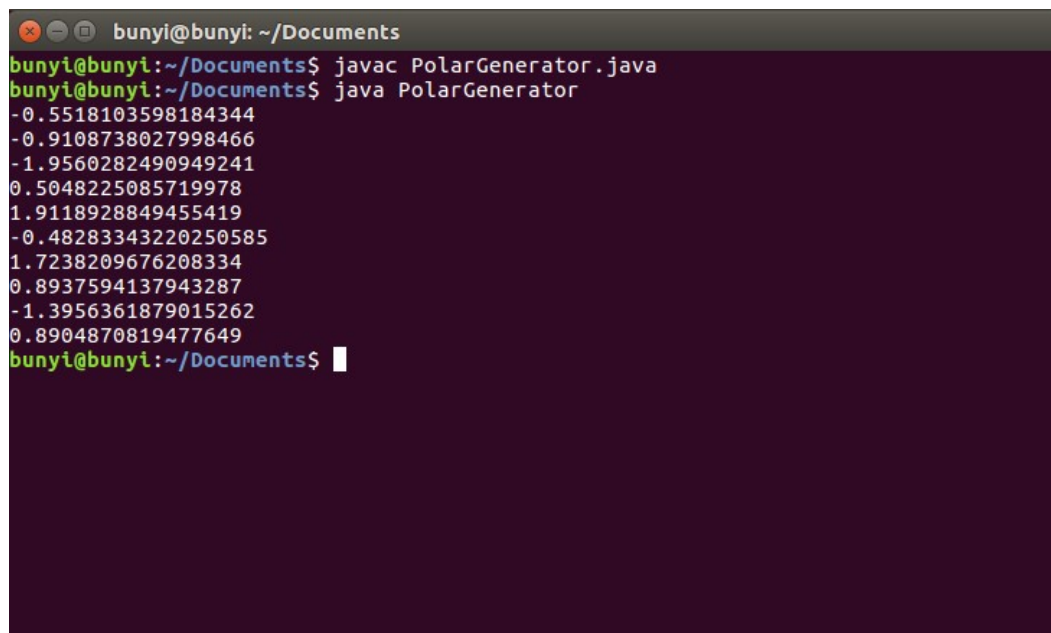
Ez a program lelke ahol a `kovetkezo()` metódus végzi a meghatározó számítást. Ha a `nincsTarolt` értéke igaz, akkor számol két random értéket. Az egyiket elmenti a `tarolt` változóba, a másikat pedig visszaadja. Amennyiben a `nincsTarolt` értéke hamis, akkor pedig a tárolt értéket fogja visszaadni.

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

    PolarGenerator g = new PolarGenerator();

    for(int i=0; i<10; ++i)
        System.out.println(g.kovetkezo());
}
```

A `main` metódusul indul el a program. Itt példányosítjuk a `PolárGenerátor` classot. Ezután egy `for` ciklussal 10-szer kiíratjuk a meghívott `kovetkezo()` metódus értékét. Minden második érték a `tarolt` változóból visszatért érték lesz. Ezeket az értékeket generálta nekem:



```
bunyi@bunyi: ~/Documents
bunyi@bunyi:~/Documents$ javac PolarGenerator.java
bunyi@bunyi:~/Documents$ java PolarGenerator
-0.5518103598184344
-0.9108738027998466
-1.9560282490949241
0.5048225085719978
1.9118928849455419
-0.48283343220250585
1.7238209676208334
0.8937594137943287
-1.3956361879015262
0.8904870819477649
bunyi@bunyi:~/Documents$
```

Az érdekesség az még itt, hogy a Java fejlesztők egy nagyon hasonló módon oldották meg a `java.util.Random` osztályban a Random szám generálást. Íme:

```
public double nextDouble() {
    return (((long) (next(26)) << 27) + next(27)) * DOUBLE_UNIT;
}

private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

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

C++-ban így néz ki a teljes kód:

```
#include <iostream>
#include <tgmath.h>
#include <cstdlib>
#include <time.h>

using namespace std;

class PolarGenerator {
private:
    bool nincsTarolt;
    double tarolt;

public:
    PolarGenerator() {
        nincsTarolt = true;
        srand (time(NULL));
    }

    double 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) {
    PolarGenerator g;

    for (int i = 0; i < 10; ++i)
        cout << g.kovetkezo() << endl;

    return 0;
}
```

A következő sorokban részenként magyarázom el mi történik a kódban.

```
private:
    bool nincsTarolt;
    double tarolt;
```

Ez a Polárgenerátor class private része. Ez tartalmaz egy bool és egy double típusú változót. Ezek a változók csak az osztályon belül lesznek elérhetőek.

```
public:
    PolarGenerator() {
        nincsTarolt = true;
        srand (time(NULL));
```

```
    }

    double 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;
        }
    }
};
```

Ez pedig a class public része, amelyben a változók és metódusok példányosítás után elérhetőek az osztályon kívül is.

```
PolarGenerator() {
    nincsTarolt = true;
    srand (time(NULL));
}
```

Az osztály nevével megegyező metódust konstruktornak nevezzük. Az ebben lévő kódok példányosításkor hajtódnak végre.

```
double 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;
}
}
```

A következő metódusban szinte semmilyen lényegi eltérés nincs a Java kódhoz képest. Ugyanaz történik ha nincsTarolt igaz akkor generál randomot, ha pedig hamis, akkor visszaadja az eltárolt értéket.

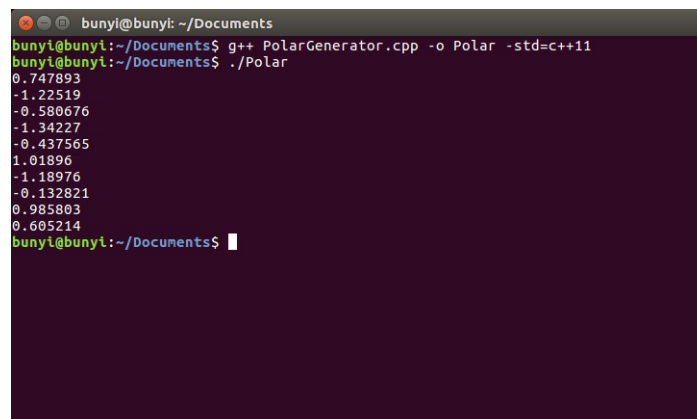
```
int main(int argc, char** argv) {
    PolarGenerator g;

    for (int i = 0; i < 10; ++i)
        cout << g.kovetkezo() << endl;

    return 0;
}
```

A main metódusban szintén, mint a Javanál példányosítunk és egy for ciklussal 10-szer visszadjuk a kovetkezo() metódus értékét.

C++-ban generált értékek:



```
bunyt@bunyt: ~/Documents
bunyt@bunyt:~/Documents$ g++ PolarGenerator.cpp -o Polar -std=c++11
bunyt@bunyt:~/Documents$ ./Polar
0.747893
-1.22519
-0.580676
-1.34227
-0.437565
1.01896
-1.18976
-0.132821
0.985803
0.605214
bunyt@bunyt:~/Documents$
```

"Gagyi"

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

Erre a tesztkérdésre kellett választ adnunk, hogy bizonyos számoknál miért jön létre végtelen ciklus és bizonyosnál miért nem.

```
public class Gagyi {  
  
    public static void main (String[]args){  
  
        Integer x = -128;  
        Integer t = -128;  
  
        while (x <= t && x >= t && t != x);  
    }  
}
```

Például itt -128-nál nem jön létre végtelen ciklus.

```
public class GagyiInfinity {  
  
    public static void main (String[]args){  
  
        Integer x = -129;  
        Integer t = -129;  
  
        while (x <= t && x >= t && t != x);  
    }  
}
```

Azonban -129-el már végtelen ciklus jön létre.

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

A válasz, hogy miért jön létre -129-el végtelen ciklus, míg -128-al semmi sem történik ebben a kódcsipetben rejlik, ami a `java.lang.Integer` osztályban található.

Elsősorban mindenképpen tudni kell, hogy a `!=`, `==` operátorok az objektumok címét hasonlítják össze, valamint a Java feltételezi, hogy a programok sokat dolgoznak, majd kis számokkal, így a poolban már előre elkészített számok vannak 127-től -128-ig. Tehát amikor létrehozunk két `Integer` objektumot és az a

poolon belül van, akkor a két objektum címe meg fog egyezni. Pontosan ez történik a -128-nál, létrehozuk a két objektumot x-et és y-ot, azonban a poolból kapjuk meg mindkettőt egy már előre elkészített objektumot így a cím megegyezik. Ezért a `x != y` hamis értéket fog adni, így a while ciklus feltétele nem teljesül és nem jön létre végtelen ciklus.

```
return new Integer(i);
```

Ha nem esik bele viszont a poolba a szám akkor, itt látszik, hogy létrehoz egy új Integer. Mivel a -129 nem esik bele így két különböző című objektumot fog létrehozni és így a while ciklus feltétele igaz lesz és végtelen ciklust kapunk.

Yoda

A feladat az volt, hogy írjunk egy olyan Java kódot ami NullPointerException hibával kilép, ha nem követjük a Yoda conditionst. Íme a kód:

```
public class Yoda {

    public static void main(String[] args) {

        String myString = null;

        if ("something".equals(myString)) {
            System.out.println("True");
        } else {
            System.out.println("False");
        }

        //NullPointerException
        if (myString.equals("something")) {
            System.out.println("True");
        } else {
            System.out.println("False");
        }
    }
}
```

A Yoda conditions egy kódolási stílus, ahol a programkódot "fordítva" írjuk be, tehát az értékadásnál a konstans értéket írjuk balra és jobbra kerül a változó amibe elmentjük. A nevét is erről a szokatlan megfordított kódírásról kapta, Yoda-ról aki a Star Wars-ban hasonlóan nem szabályszerűen alkalmazza az angolt.

```
int érték = 3;
if( érték == 3) {
    System.out.println("Igaz");
}
```

Ez ahogy rendesen írnánk egy kódot.

```
int érték = 3;
```

```
if( 3 == érték) {  
    System.out.println("Igaz");  
}
```

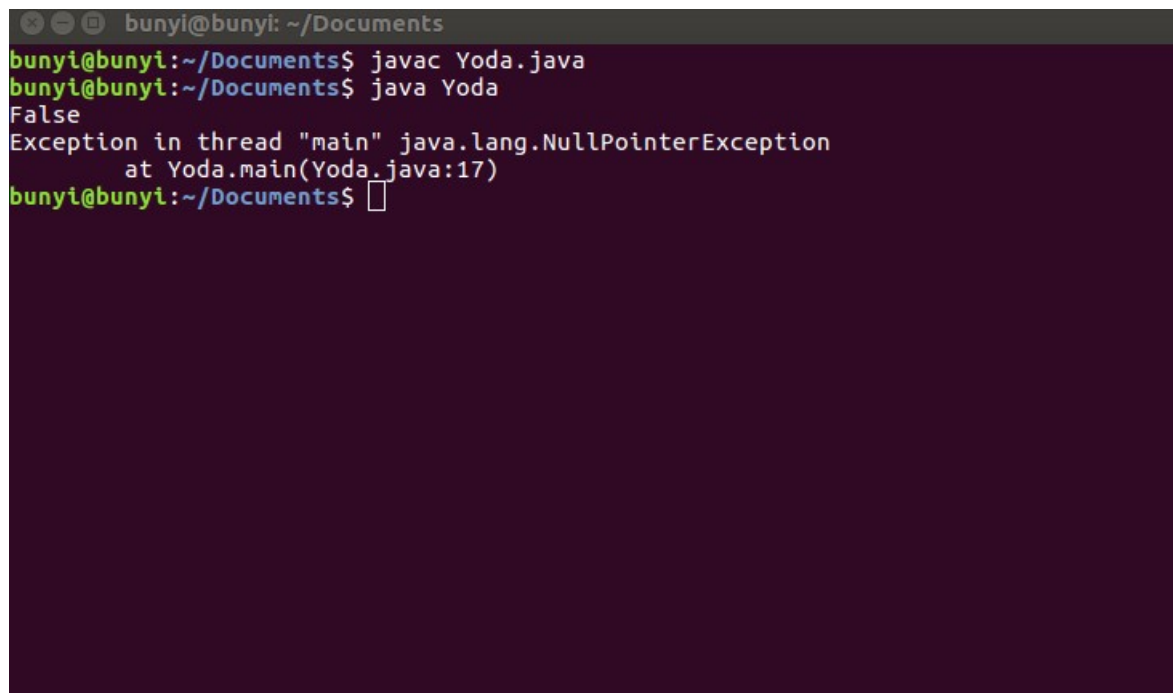
Ugyanaz a kód Yoda conditions-t használva. Mindakettővel teljesen normálisan fog működni a program.

```
if ("something".equals(myString)) {  
    System.out.println("True");  
} else {  
    System.out.println("False");  
}
```

Ez a rész egy tipikusan Yoda conditions-t használva lett megírva. Látszik, hogy az equal metódus bal oldalára került a konstans, jelen esetben egy string. Az equal ezt hasonlítja össze a myStringben lévő null értékkel. Ekkor semmilyen hiba nem fordul elő egyszerűen hamis lesz a visszatért érték.

```
if (myString.equals("something")) {  
    System.out.println("True");  
} else {  
    System.out.println("False");  
}
```

Ezek a sorok azonban nem használják a Yoda conditions-t, így NullPointerException-t dobnak a Java-ban. Így a Yoda conditions-al elkerülhető néhány nem biztonságos null viselkedés.



```
bunyi@bunyi: ~/Documents  
bunyi@bunyi:~/Documents$ javac Yoda.java  
bunyi@bunyi:~/Documents$ java Yoda  
False  
Exception in thread "main" java.lang.NullPointerException  
    at Yoda.main(Yoda.java:17)  
bunyi@bunyi:~/Documents$
```

Amint látszik a console-on is NullPointerException hibával kilép.

Azonban a Yoda conditions bírálói nagyban panaszkodnak az olvashatóság elvesztésére, ha ezt a módszert alkalmazzuk.

Kódolás from scratch

Egy olyan feladatot kaptunk, hogy írjuk meg a BBP algoritmus megvalósítását. Ez egy olyan algoritmus ami kiszámítja a Pi hexadecimális számjegyeit egy megadott helyen. Íme a kód:

```
public class BBP {
    String HexaJegyek;

    public BBP(int d) {

        double HexPi = 0.0;

        double S1 = Sj(d, 1);
        double S4 = Sj(d, 4);
        double S5 = Sj(d, 5);
        double S6 = Sj(d, 6);

        HexPi = 4.0*S1 - 2.0*S4 - S5 - S6;

        HexPi = HexPi - Math.floor(HexPi);

        StringBuffer sb = new StringBuffer();

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

        while(HexPi != 0.0) {

            int jegy = (int)Math.floor(16.0d*HexPi);

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

            HexPi = (16.0d*HexPi) - Math.floor(16.0d*HexPi);

        }

        HexaJegyek = sb.toString();
    }

    public String toString() {
        return HexaJegyek;
    }

    public double Sj(int d, int j) {

        double Sj = 0.0;
```

```
        for (int k = 0; k <= d; k++)
            Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);

        return Sj - Math.floor(Sj);
    }

    public long n16modk(int n, int k) {

        int t = 1;
        while(t <=n)
            t *= 2;

        long r = 1;

        while(true) {

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

            t = t/2;

            if(t < 1)
                break;

            r = (r*r) % k;
        }

        return r;
    }

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

Nézzük részekre bontva:

```
String HexaJegyek;
```

Először is létrehozunk a BBP classban egy változót. Ebben a változóban fogjuk tárolni a végeredményt, tehát a Pi hexadecimális jegyeit az adott helyen.

Nézzük először a metódusokat, mert csak azután lehet megérteni a konstruktor működését.

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

Legegyszerűbb a a toString() metódussal kezdeni. Ez egyszerűen visszaadja a végső eredményt egy string-ként, de ebben az esetben az eredményünk eleve string típusú így nincs más dolgunk csak azt visszaadni.

```
public long n16modk(int n, int k) {  
  
    int t = 1;  
    while(t <=n)  
        t *= 2;  
  
    long r = 1;  
  
    while(true) {  
  
        if(n >= t) {  
            r = (16*r) % k;  
            n = n - t;  
        }  
  
        t = t/2;  
  
        if(t < 1)  
            break;  
  
        r = (r*r) % k;  
    }  
  
    return r;  
}
```

Az n16modk metódusban számoljuk ki bináris hatványozással a $16^n \bmod k$ értékét.

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

Az Sj metódus egy double értékkel fog visszatérni. A BBP algoritmus képlet alapján fogja visszaadni ezt a számot.

```
public BBP(int d) {  
  
    double HexPi = 0.0;  
  
    double S1 = Sj(d, 1);  
    double S4 = Sj(d, 4);  
    double S5 = Sj(d, 5);  
    double S6 = Sj(d, 6);
```

```
HexPi = 4.0*S1 - 2.0*S4 - S5 - S6;

HexPi = HexPi - Math.floor(HexPi);

StringBuffer sb = new StringBuffer();

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

while(HexPi != 0.0) {

    int jegy = (int)Math.floor(16.0d*HexPi);

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

    HexPi = (16.0d*HexPi) - Math.floor(16.0d*HexPi);

}

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

Ez a rész a konstruktora a BBP classnak, ez mindenképpen le fog futni amikor példányosítják a függvényt. Nézzük ezt is részekre bontva:

```
double HexPi = 0.0;

double S1 = Sj(d, 1);
double S4 = Sj(d, 4);
double S5 = Sj(d, 5);
double S6 = Sj(d, 6);

HexPi = 4.0*S1 - 2.0*S4 - S5 - S6;

HexPi = HexPi - Math.floor(HexPi);
```

Először is létrehoz 5 db változót. A HexPi-t azért, hogy legyen miben tárolni a számot amit a képlet kiszámolása után megkapunk. Az S1, S4, S5, S6 változók részelemek az alatta lévő képletben. A d változó az a szám itt amit a felhasználó ad meg, hogy hányadik helyen számolja a Pi hexadecimális értékét.

```
StringBuffer sb = new StringBuffer();

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

while(HexPi != 0.0) {

    int jegy = (int)Math.floor(16.0d*HexPi);
```

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

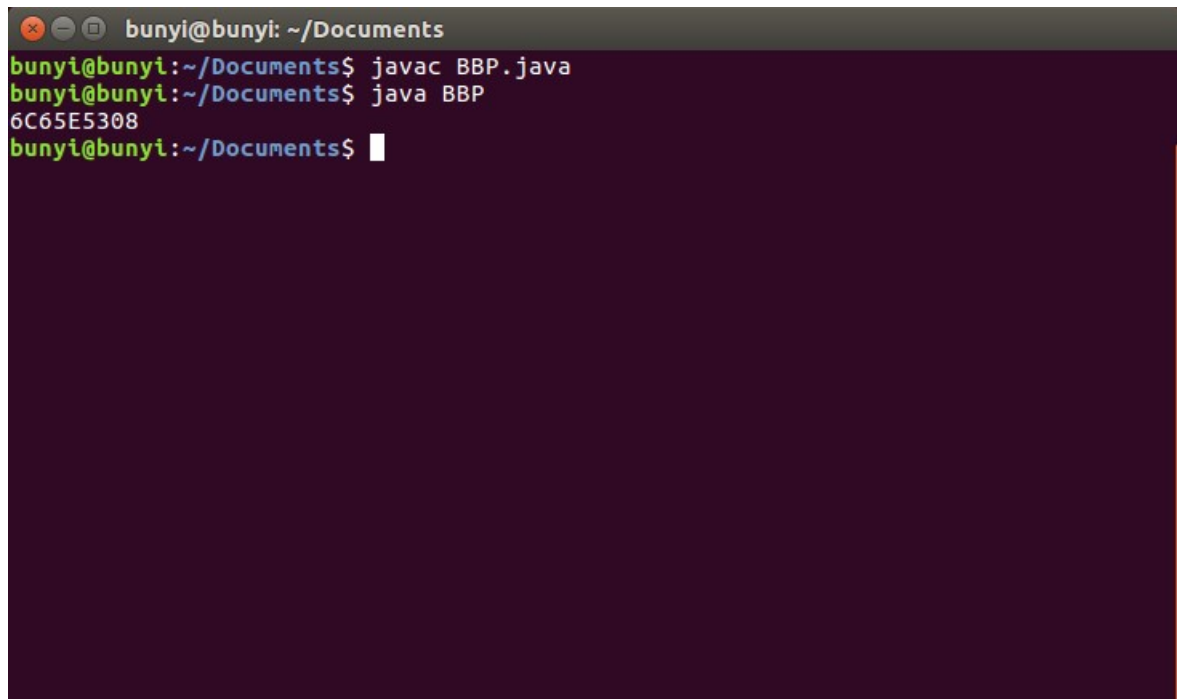
    HexPi = (16.0d*HexPi) - Math.floor(16.0d*HexPi);
}

HexaJegyek = sb.toString();
```

Itt létrehozunk egy StringBuffer-t amiben ideiglenesen elmentjük a hexadecimális számokat stringként. Egy Character típusú tömböt is alkotunk, ebben tároljuk a 16-os számrendszerben jelenlévő karaktereket. A while cikluson belül a stringBuffer-hez hozzáfűzzük a hexa számjegyeket, majd a végén átadjuk a HexaJegyek nevű stringnek.

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

A main metódusban a kiíratáson belül egy példányosítást láthatunk 10^6 értékkel. Tehát a program a Pi 1 milliommodik helyén lévő hexadecimális számjegyeit fogja visszaadni. Ahogy látható is:



```
bunyi@bunyi: ~/Documents
bunyi@bunyi:~/Documents$ javac BBP.java
bunyi@bunyi:~/Documents$ java BBP
6C65E5308
bunyi@bunyi:~/Documents$
```

4. fejezet

Helló, Liskov!

Liskov helyettesítés sértése

Ebben a feladatban egy objektum orientált kódot kellett írunk Java és C++ nyelven, amely megsérti a Liskov elvet. Először is mi az a Liskov elv?

A liskov elvet Barbara Liskov mutatta be először. Fő célja a rossz OO tervezés megakadályozása. Az elv kimondja, hogy ha S altípusa T-nek, akkor bármely helyen ahol T-t alkalmazzuk S-t is minden probléma nélkül használhatjuk úgy, hogy a programrész tulajdonságai nem változnak.

Java kód:

```
static class Macska {
    public void szőrös() {}
}

static class Program {
    void fgv (Macska macska) {
        macska.szőrös();
    }
}

static class Perzsa extends Macska {}
static class Szfinx extends Macska {}

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

    Perzsa perzsa = new Perzsa();
    program.fgv(perzsa);

    Szfinx szfinx = new Szfinx();
    program.fgv(szfinx);
}
```


C++ kód:

```
class Macska {
public:
    virtual void szőrös() {};
};

class Program {
public:
    void fgv(Macska& macska) {
        macska.szőrös();
    }
};

class Perzsa : public Macska {};
class Szfinx : public Macska {};

int main(int argc, char** argv) {
    Program program;
    Macska macska;
    program.fgv(macska);

    Perzsa perzsa;
    program.fgv(perzsa);

    Szfinx szfinx;
    program.fgv(szfinx);
}
```

A kódokban a Macska nevű őosztály a T, ennek gyermekosztályai pedig a Perzsa és a Szfinx ami itt az S. A Macska nevű osztály tartalmaz egy szőrös() nevű függvényt, ez azt jelenti, hogy minden macska szőrös, de ez nem igaz. A szfinx fajtájú macskák szőrtelenek. Azonban a program őt is szőrösnek titulálja. A Liskov elv így sérül, mivel van olyan leszármazott, amely nem rendelkezik az őse tulajdonságával, így behelyettesíteni sem lehet az ős helyére a leszármazottat.

Úgy tudjuk kiküszöbölni ezt a hibát itt, hogy létrehozunk, még egy osztályt SzőrösMacska néven amely a Macskából származik és ennek lesz leszármazottja a Perzsa.

```
static class Macska {}

static class Program {
    void fgv (Macska macska) {}
}

static class SzőrösMacska extends Macska {
    public void szőrös() {}
}

static class Perzsa extends SzőrösMacska {}
static class Szfinx extends Macska {}
```

```
public static void main(String[] args) {  
    Program program = new Program();  
    Macska macska = new Macska();  
    program.fgv(macska);  
  
    Perzsa perzsa = new Perzsa();  
    program.fgv(perzsa);  
  
    Szfinx szfinx = new Szfinx();  
    program.fgv(szfinx);  
}
```

A fent látható megoldással már nem sérül a Liskov elv.

Szülő-gyerek

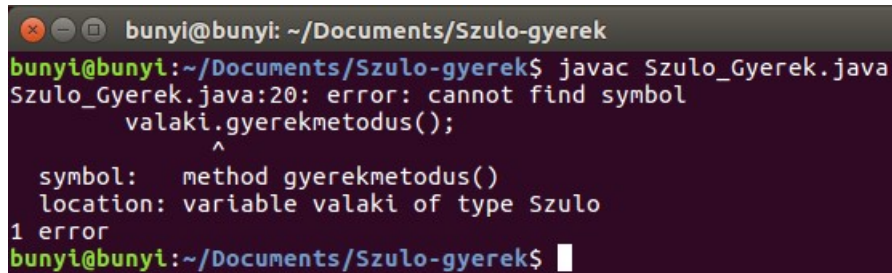
Ennek a feladatnak a megoldásához tisztázni kell az öröklődés és a polimorfizmus fogalmát. Az öröklődés az, amikor egy osztályt egy már létező osztály kiterjesztésével definiálunk. Ekkor a már létező osztály lesz az ősz- vagy szülőosztály. Az osztály amit pedig kiterjesztettünk leszarmazott vagy gyermekosztálynak nevezzük. A létrejött utód egy új osztály lesz, amely öröklí az ősz metódusait, tagváltozóit. A public, protected, private kulcsszavakkal lehet megadni, hogy a gyermek melyik metódusokat, változókat lássa. A public-al rendelkezőket mindenki használhatja, a protected-et csak az ősz leszarmazottai. A private tagot csak az az osztály, amelyben létrehozták a tagot.

A polimorfizmus lényege az, hogy mivel a gyermekosztály örökölt minden metódust és tagváltozót így egy olyan környezetben ahol az őst lehet használni a gyermeket is.

Példa arra, hogy az őszön keresztül csak az ősz üzenetei küldhetőek. Javában:

```
class Szulo {  
    void szulometodus() {  
        System.out.println("Szulo vagyok!");  
    }  
}  
  
class Gyerek extends Szulo {  
    void gyerekmetodus() {  
        System.out.println("Gyerek vagyok!");  
    }  
}  
  
class Szulo_Gyerek {  
    public static void main(String[] args) {  
        Szulo valaki = new Gyerek();  
  
        valaki.szulometodus();  
        valaki.gyerekmetodus(); //Nem látja a gyermekmetódusát!  
    }  
}
```

```
}
```



A screenshot of a terminal window with a dark background. The title bar shows 'bunyi@bunyi: ~/Documents/Szulo-gyerek'. The prompt is 'bunyi@bunyi:~/Documents/Szulo-gyerek\$'. The command 'javac Szulo_Gyerek.java' has been executed. The output shows an error at line 20: 'error: cannot find symbol'. The code snippet is 'valaki.gyerekmetodus();' with an arrow pointing to 'gyerekmetodus()'. Below the error, it says 'symbol: method gyerekmetodus()' and 'location: variable valaki of type Szulo'. It ends with '1 error' and the prompt 'bunyi@bunyi:~/Documents/Szulo-gyerek\$'.

```
bunyi@bunyi: ~/Documents/Szulo-gyerek
bunyi@bunyi:~/Documents/Szulo-gyerek$ javac Szulo_Gyerek.java
Szulo_Gyerek.java:20: error: cannot find symbol
    valaki.gyerekmetodus();
           ^
    symbol:   method gyerekmetodus()
    location: variable valaki of type Szulo
1 error
bunyi@bunyi:~/Documents/Szulo-gyerek$
```

Ahogy a console-on is látszik a program nem látja a gyermekmetódusát, azonban a gyermek látja a szülő metódusát.

C++-ban:

```
#include <iostream>

class Szulo {
public:
    void szulometodus() {
        std::cout << "Szulo vagyok!" << std::endl;
    }
};

class Gyerek : public Szulo {
public:
    void gyerekmetodus() {
        std::cout << "Gyerek vagyok!" << std::endl;
    }
};

int main() {
    Szulo* valaki = new Gyerek();

    valaki->szulometodus();
    valaki->gyerekmetodus(); //Nem látja a gyermekmetódusát!
}
```

```

bunyi@bunyi: ~/Documents/Szulo-gyerek
bunyi@bunyi:~/Documents/Szulo-gyerek$ g++ Szulo_Gyerek.cpp -o Szulo_Gyerek
Szulo_Gyerek.cpp: In function 'int main()':
Szulo_Gyerek.cpp:21:10: error: 'class Szulo' has no member named 'gyerekmetodus'
    valaki->gyerekmetodus();
            ^
bunyi@bunyi:~/Documents/Szulo-gyerek$

```

Itt is ugyanúgy error kapunk, mivel hiába castoltuk a gyermeket szülővé az nem éri el a gyermek metódusát.

Anti OO

Össze kellett hasonlítani a BBP algoritmus kód futási idejét C, C++, Java és C# nyelven. Egy virtuális linux gépen futattam a kódokat. Ilyen eredményt kaptam:

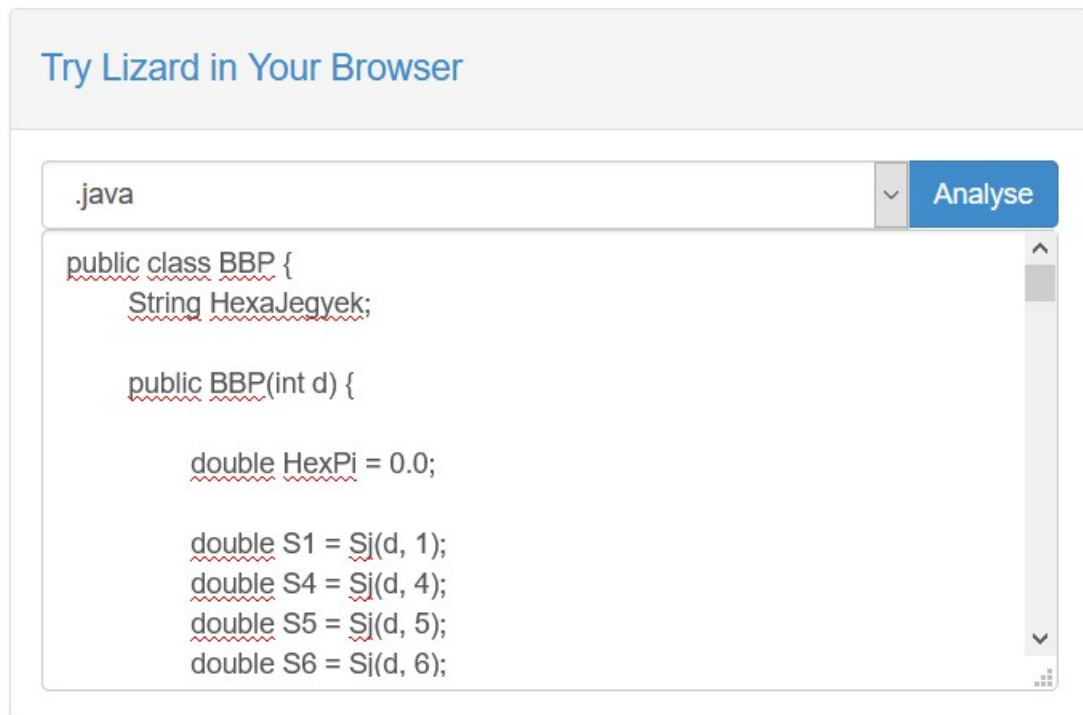
	C	C++	Java	C#
10 ⁶	3.051	2.823	2.575	2.617
10 ⁷	34.460	33.332	28.978	30.589
10 ⁸	385.376	386.714	338.139	353.068

4.1. táblázat. Összehasonlítás

Ahogy látszik a C nyelv volt a leglassabb. Ez várható volt, hisz ez a legöregebb nyelv a négy közül. Leggyorsabb volt a Java kód, amely a 10⁸ pozíciónál 10 másodperccel leelőzte a C# is. Több oka is van, hogy a Java legyőzött mindenkit. Elsőnek lehet mondani, hogy a memória kiosztást sokkal jobban kezeli, mint a többi nyelv. Valamint a JVM jobban optimalizálja a metódus hívásokat. Futás időben dinamikus elemzést tud végezni, hogy mire van szükség és mire nem, így gyorsabb működést képes nyújtani, mint egy C++ fordítóprogram.

Ciklomantikus komplexitás

Ebben a feladatban ki kellett számolnunk valamelyik programunk függvényeinek ciklomantikus komplexitását. Ezt én egy online program segítségével oldottam meg, a Lizard-dal. A BBP java kódját elemzte a program. Íme:



Egyszerűen csak ki kell választanunk a forráskód nyelvét, majd beillesztenünk magát a kódot.

Code analyzed successfully.

File Type **.java** Token Count **401** NLOC **54**

Function Name	NLOC	Complexity	Token #	Parameter #
BBP::BBP	22	3	196	
BBP::toString	3	1	8	
BBP::Sj	6	2	70	
BBP::n16modk	17	5	87	
BBP::main	3	1	22	

A végeredményen a számok minél kisebbek annál jobb, hiszen ha túl bonyolultak a függvények nehezen olvasható a program.

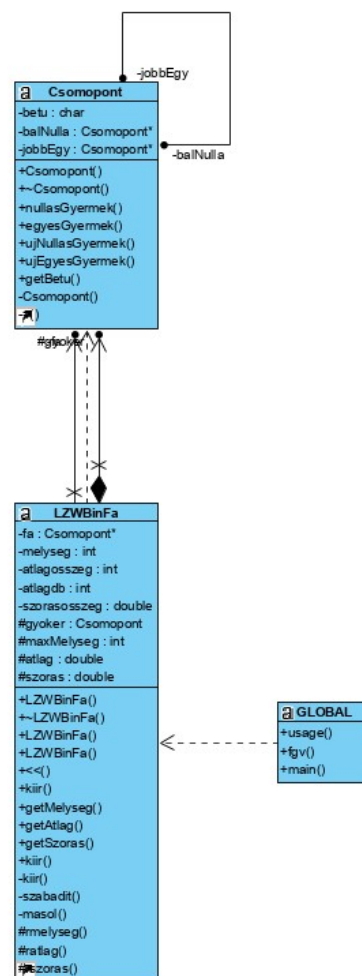
Számítása a gráfelméleten alapul. A forráskód alapján határozza meg az egyes függvények ciklomantikus komplexitását. Ez a független utak számát jelenti, hogy a program mennyire bonyolult vezérlési szempontból. Két út akkor számít függetlennek, ha mindkettőben van olyan pont, amely nem eleme a másiknak.

5. fejezet

Helló, Mandelbrot!

Reverse engineering UML osztálydiagram

A feladatban az első védési program C++ kódjából kellett UML osztálydiagramot generálnunk. Az UML egy grafikus modellező nyelv, amely diagramokat tartalmaz. A diagramok dobozokat, szövegeket, ikonokat és vonalakat foglalja magába. Az UML generálásához a Visual Paradigm alkalmazást használtam. Nekem ezt a diagramot alkotta:



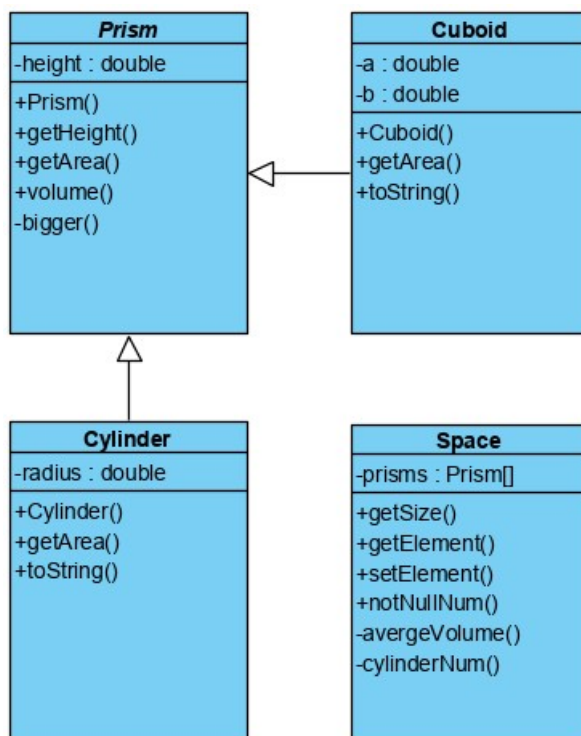
Az ábrán a kék dobozok az osztályok, tehát a Csomopont, az LZWBinFa és a GLOBAL. Közvetlenül az osztályok neve alatt találhatóak a változók. Az elérhetőségüket egy -, + vagy egy # jelöli. A - jelenti a private-ot, a + a publicot és a # a protectedet. A változók után egy vízszintes vonallal elválasztva következnek az osztály metódusai. Az elérhetőség itt is ugyanúgy van jelölve. A törzse mindegyiknek üresen van hagyva, hiszen ez csak egy modell, nem kell kódot írni bele, sőt nem is lehet.

Az asszociáció társítást jelent. A legegyszerűbb fajtája az osztályok között rajzolt egyszerű vonal. Azt jelenti valamilyen kapcsolat áll fent a két osztály között. A vonal végén ha nyíl van az az asszociáció irányát, navigálhatóságát jelzi. Ha nincs nyíl akkor a kapcsolat két irányú. Egy X-el jelöljük a nyilat, ha az asszociáció nem navigálható.

Az asszociáción belül megkülönböztetjük a tartalmazást, amelynek két fajtája van, gyenge és erős. A gyenge tartalmazást nevezzük aggregációnak. Ezt akkor mondhatjuk amikor a rész hozzátartozik valamihez, de létezik önállóan is. Jele az UML-ben az üres rombusz. Az erős tartalmazást nevezzük kompozíciónak. Ekkor a tartalmazott nem létezhet önmagában, csak részként valamiben. Ennek jele a telített rombusz. Ez látható is a generált diagramon. Itt azt fejezi ki, hogy a Csomopont osztály az LZWBinFa osztály része és nem létezhet nélküle.

Forward engineering UML osztálydiagram

Itt az előző feladat ellenkezőjét kellett végrehajtanunk, egy általunk létrehozott UML diagramból kellett forrást generálnunk. Visual Paradigm-ban dolgoztam. Létrehoztam 4 db osztályt: Prism, Cuboid, Cylinder és Space. A Prism osztályt absztraktnak jelöltem ez látszik abból, hogy dőltbetűvel van írva. A nyilak pedig azt jelentik, hogy a Cuboid és Cylinder osztály a Prism leszármazottja.



```

public abstract class Prism {

    private double height;
  
```

```
public Prism() {  
    // TODO - implement Prism.Prism  
    throw new UnsupportedOperationException();  
}  
  
public void getHeight() {  
    // TODO - implement Prism.getHeight  
    throw new UnsupportedOperationException();  
}  
  
public void getArea() {  
    // TODO - implement Prism.getArea  
    throw new UnsupportedOperationException();  
}  
  
public void volume() {  
    // TODO - implement Prism.volume  
    throw new UnsupportedOperationException();  
}  
  
private void bigger() {  
    // TODO - implement Prism.bigger  
    throw new UnsupportedOperationException();  
}  
}
```

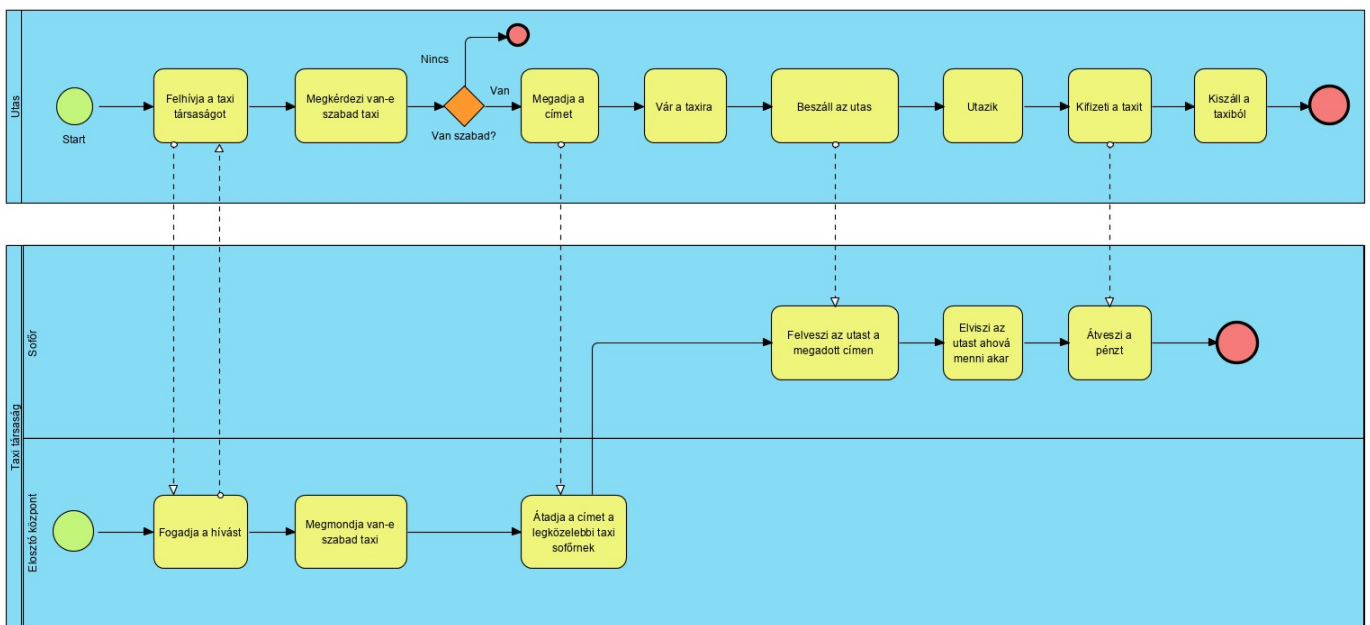
```
public class Cuboid extends Prism {  
  
    private double a;  
    private double b;  
  
    public Cuboid() {  
        // TODO - implement Cuboid.Cuboid  
        throw new UnsupportedOperationException();  
    }  
  
    public void getArea() {  
        // TODO - implement Cuboid.getArea  
        throw new UnsupportedOperationException();  
    }  
  
    public void toString() {  
        // TODO - implement Cuboid.toString  
        throw new UnsupportedOperationException();  
    }  
}
```

Ilyen kódokat generált a létrehozott diagramból. Ahogy látszik tényleg absztraktként hozta létre a Prism osztályt. A változókat is mind legenerálta. A metódus törzsekbe pedig írt, hogy implementálni kell a

metódusokat. Ez a fajta kódgenerálás inkább azoknak jó akik lusták megírni a kódot teljes egészében.

BPMN

A BPMN a Business Process Model and Notation rövidítése. Ez egy grafikus ábrázolás hasonló az UML-hez, az üzleti folyamatok modelljének meghatározására szolgál. A BPMN célja, hogy támogassa az üzleti folyamatok menedzselését, mind a műszaki felhasználók, mind az üzleti felhasználók számára. Egy szabványos jelölést biztosít, amely mindenkinek egyaránt könnyen érthető. Tehát egy közös nyelvként szolgál, áthidalva az üzleti folyamatok tervezése és megvalósítása között gyakran felmerülő kommunikációs rést.



Ez a példa egy hétköznapi folyamatot ír le, a taxi rendelés menetét. Az utas felhívja a taxi társaságot, ahol az elosztó központtal információt cserélnek. Az elosztó központ továbbítja a címet a legközelebbi sofőrnek. A sofőr felveszi az utast elviszi a megadott helyre. Az utas kifizeti az útiköltséget és vége. Persze ez elég egyszerű példa, sokkal komolyabbakat is lehet modellezni.

BPEL Helló, Világ!

<https://youtu.be/eawsm8fb3iY>

Létre kellett hoznunk egy webszervert, amelynek egy stringet megadva ugyanazt a stringet dobja vissza. Egy 5 perces videóban meg is csináltam ezt. Azonban voltak előkészületek, mivel már egy deprecated feladatról van szó. Először is hozzá kellett adnom az Eclipse-hez a BPEL pluginokat. Másodszor fel kellett telepíteni az Apache Tomcatet hozzáadva az ode kiegészítőt. Ezután a videó alapján megcsináltam a szervert, de szembesültem azzal, hogy nincs Web Services Explorer-em. Elég nehezen találtam meg, hogy az axis2 plugin tartalmazza ezt a funkciót.

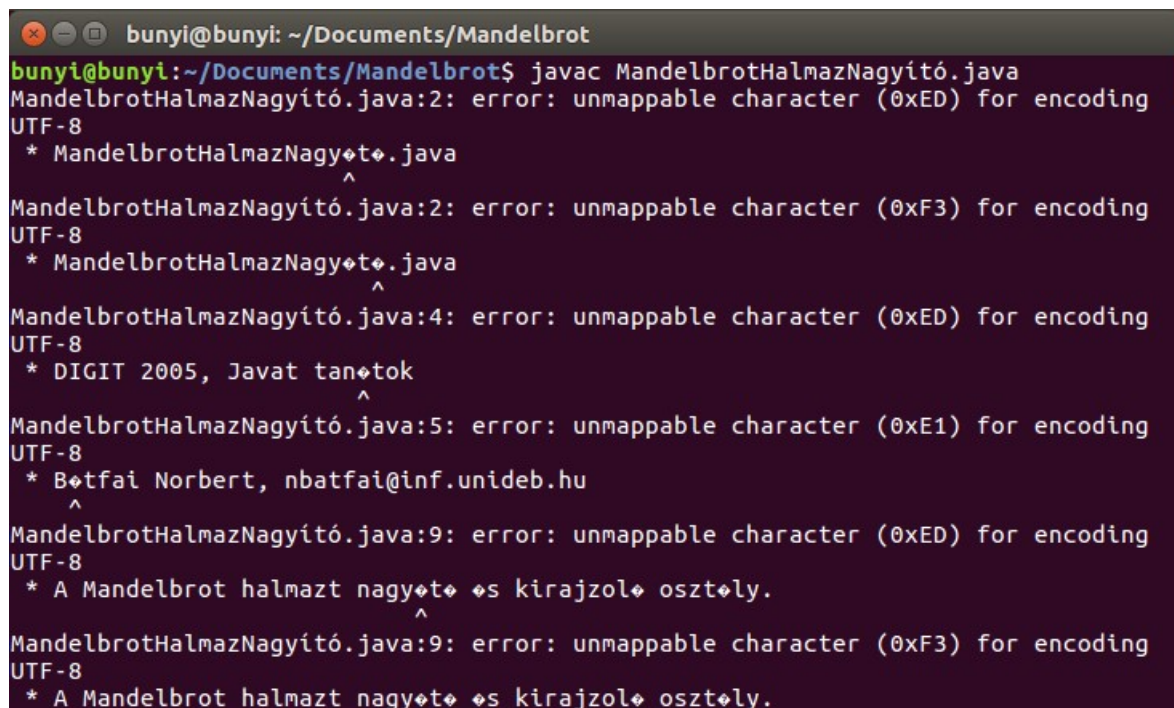
A BPEL nyelvről még mondanék egy pár szót. A BPEL a Business Process Execution Language rövidítése. Ez az üzleti folyamatok leírására és webszerverrel kapcsolatos műveletekhez használják. A folyamatok a BPEL-ben az információt csak a webszolgáltatási felületek felhasználásával fogadják és továbbítják.

6. fejezet

Helló, Chomsky!

Encoding

A feladatban futtatni kellett a MandelbrotHalmazNagyító programot. Az volt benne a kihívás, hogy a forráskódban ékezetes karakterek is szerepeltek és ha fordítottuk akkor ezt a hibát kaptuk:



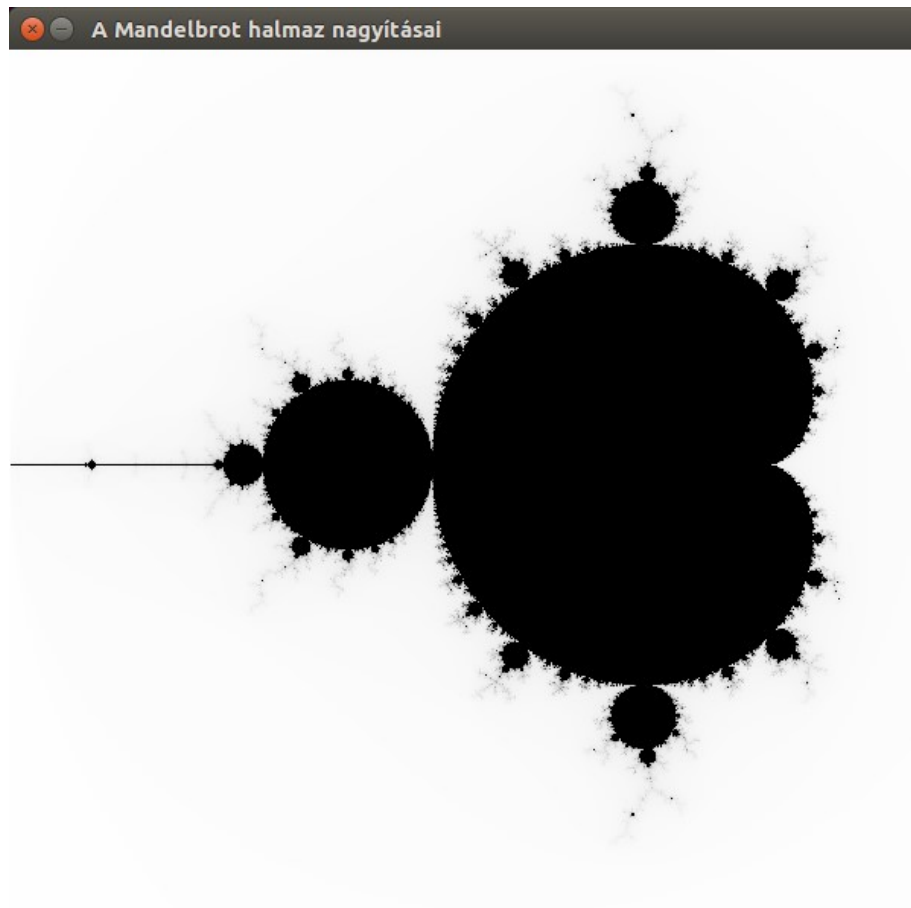
```
bunyi@bunyi: ~/Documents/Mandelbrot
bunyi@bunyi:~/Documents/Mandelbrot$ javac MandelbrotHalmazNagyító.java
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xED) for encoding UTF-8
 * MandelbrotHalmazNagyító.java
      ^
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xF3) for encoding UTF-8
 * MandelbrotHalmazNagyító.java
      ^
MandelbrotHalmazNagyító.java:4: error: unmappable character (0xED) for encoding UTF-8
 * DIGIT 2005, Javat tanítok
      ^
MandelbrotHalmazNagyító.java:5: error: unmappable character (0xE1) for encoding UTF-8
 * Botfai Norbert, nbatfai@inf.unideb.hu
      ^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xED) for encoding UTF-8
 * A Mandelbrot halmazt nagyító és kirajzó osztály.
      ^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding UTF-8
 * A Mandelbrot halmazt nagyító és kirajzó osztály.
```

Ahogy látszik a képen a kódolással lesz a hiba. Mivel az UTF-8-as kódolás számára nem találhatók azok a karakterek amelyeket itt ? jelöl, ezek a kódban az ékezetes betűk.

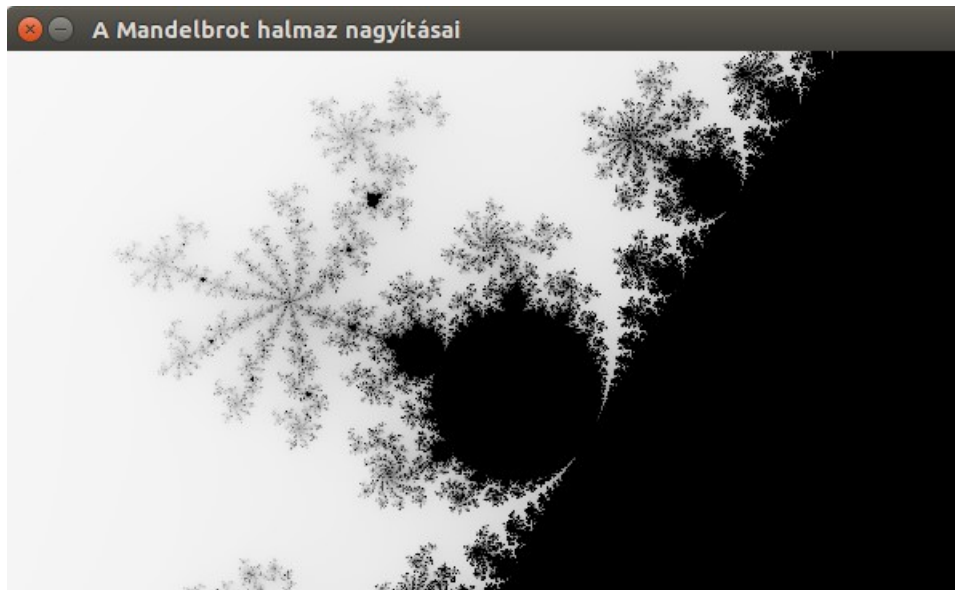
```
bunyi@bunyi: ~/Documents/Mandelbrot
MandelbrothHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
UTF-8
    // vizsgáljuk egy adott pont iterációját:
    ^
MandelbrothHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
UTF-8
    // vizsgáljuk egy adott pont iterációját:
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
UTF-8
    // Az egérmutató pozíciója
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
UTF-8
    // Az egérmutató pozíciója
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
UTF-8
    // Az egérmutató pozíciója
    ^
100 errors
bunyi@bunyi:~/Documents/Mandelbrot$ javac -encoding "ISO-8859-2" MandelbrothHalma
zNagyító.java
bunyi@bunyi:~/Documents/Mandelbrot$
```

A megoldás az encoding kapcsoló volt rá a megfelelő kódolással. Kódolásnak pedig a Latin1 vagy Latin2 kellett megadni. Ezek a kódolások tartalmazzák az ékezetes betűket. Itt a Latin2 kellett használni, mert az tartalmazza az "ő" és "ű" is. Ennek a kódja pedig az ISO-8859-2 volt. Így már sikeresen fordult és futott is a program.

A program futás közben:



Ahogy a felnagyítja a képet:



I334d1c4^5

Olyan osztályt kellett írni, amely Leet chiperként működik. A Leet az egy másik angol ábécé, amit legfőképpen az interneten használnak. Az ASCII karakterek különféle kombinációját használja a latin betűk cseréjéhez. Legfőképpen az internetes fórumokon, chat-szobákban és online játékokban használják. Íme a kód:

```
import java.io.*;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Scanner;

public class Leet {

    static String readFile(String path, Charset encoding) throws ↵
        IOException {
        byte[] encoded = Files.readAllBytes(Paths.get(path));
        return new String(encoded, encoding);
    }

    static String translator(String text) {
        if (text.contains("a")) {
            text = text.replace("a", "4");
        }
        if (text.contains("b")) {
            text = text.replace("b", "8");
        }
        if (text.contains("c")) {
```

```
        text = text.replace("c", "<");
    }
    if (text.contains("d")) {
        text = text.replace("d", "|");
    }
    if (text.contains("e")) {
        text = text.replace("e", "3");
    }
    if (text.contains("f")) {
        text = text.replace("f", "|=");
    }
    if (text.contains("g")) {
        text = text.replace("g", "9");
    }
    if (text.contains("h")) {
        text = text.replace("h", "#");
    }
    if (text.contains("i")) {
        text = text.replace("i", "1");
    }
    if (text.contains("j")) {
        text = text.replace("j", "_|");
    }
    if (text.contains("k")) {
        text = text.replace("k", "<");
    }
    if (text.contains("l")) {
        text = text.replace("l", "|_");
    }
    if (text.contains("m")) {
        text = text.replace("m", "\\|");
    }
    if (text.contains("n")) {
        text = text.replace("n", "\\|");
    }
    if (text.contains("o")) {
        text = text.replace("o", "0");
    }
    if (text.contains("p")) {
        text = text.replace("p", "|2");
    }
    if (text.contains("q")) {
        text = text.replace("q", "(,");
    }
    if (text.contains("r")) {
        text = text.replace("r", "|?");
    }
    if (text.contains("s")) {
        text = text.replace("s", "$");
    }
}
```

```
        if (text.contains("t")) {
            text = text.replace("t", "7");
        }
        if (text.contains("u")) {
            text = text.replace("u", "|_|");
        }
        if (text.contains("v")) {
            text = text.replace("v", "\\\/");
        }
        if (text.contains("w")) {
            text = text.replace("w", "\\\/\\\/");
        }
        if (text.contains("x")) {
            text = text.replace("x", "><");
        }
        if (text.contains("y")) {
            text = text.replace("y", "'/");
        }
        if (text.contains("z")) {
            text = text.replace("z", "2");
        }
        return text;
    }

    public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Kérem a fájlj nevét: ");
        String fileName = scanner.nextLine();
        String fullFileContents =
            readFile(fileName, StandardCharsets.UTF_8);
        System.out.println("File beolvasva.");
        fullFileContents = fullFileContents.toLowerCase();
        fullFileContents = translator(fullFileContents);
        System.out.println("File lefordítva.");

        PrintWriter writer = new PrintWriter("output.txt", "UTF-8");
        writer.print(fullFileContents);
        writer.close();
        System.out.println("A fordított szöveg az
                           output.txt-ben megtalálható.");

        scanner.close();
    }
}
```

A feladat megoldásához a legegyszerűbb megoldást alkalmaztam az if() függvényt.

```
static String readFile(String path, Charset encoding) throws IOException {
    byte[] encoded = Files.readAllBytes(Paths.get(path));
    return new String(encoded, encoding);
}
```

```
}
```

A `readFile()` metódussal először beolvasok egy szöveges fájlt.

```
static String translator(String text) {  
    if (text.contains("a")) {  
        text = text.replace("a", "4");  
    }  
}
```

A `translator()` metódusban történik a lényeg, a szöveget amit beolvastam átadom neki. Ekkor ha a szöveg tartalmaz egy bizonyos karaktert, jelen esetben egy "a" betűt akkor mindenhol ahol "a" van a szövegben ott a `replace()` beépített metódussal kicserélem a karaktert egy "4"-esre.

```
public static void main(String[] args) throws IOException {  
    Scanner scanner = new Scanner(System.in);  
  
    System.out.print("Kérem a fájl nevét: ");  
    String fileName = scanner.nextLine();  
    String fullFileContents =  
        readFile(fileName, StandardCharsets.UTF_8);  
    System.out.println("File beolvasva.");  
    fullFileContents = fullFileContents.toLowerCase();  
    fullFileContents = translator(fullFileContents);  
    System.out.println("File lefordítva.");  
  
    PrintWriter writer = new PrintWriter("output.txt", "UTF-8");  
    writer.print(fullFileContents);  
    writer.close();  
    System.out.println("A fordított szöveg az  
        output.txt-ben megtalálható.");  
  
    scanner.close();  
}
```

A `main` metóduson belül bekérem a beolvasandó fájl nevét, beolvasom a fájlt. Kisbetűssé alakítom, átadom a szöveget a `translator()` metódusnak. Majd kiírom egy külön fájlba a már átalakított szöveget.

Full screen

Egy Java-ban futó fullscreen-es programot kellett írunk. Én egy bejelentkezési képernyőt írtam. Íme a kód:

```
import java.awt.*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
import javax.swing.*;
```



```
public class Login {

    public static void main(String[] args) {

        //Create, set buttons

        //Login button
        JButton login = new JButton("Login");
        login.setBounds(50, 300, 275, 50);

        //Exit button
        JButton exit = new JButton("Exit");
        exit.setBounds(375, 300, 275, 50);
        exit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });

        //Create, set labels

        //User name label
        JLabel username = new JLabel("Username:");
        username.setBounds(50, 50, 200, 75);
        username.setFont(new Font("Arial", Font.PLAIN, 40));

        //Password label
        JLabel password = new JLabel("Password:");
        password.setBounds(50, 150, 200, 75);
        password.setFont(new Font("Arial", Font.PLAIN, 40));

        //Create, set text fields

        //User text field
        JTextField userText = new JTextField();
        userText.setBounds(275, 55, 375, 60);

        //Pass text field
        JTextField passText = new JTextField();
        passText.setBounds(275, 155, 375, 60);

        //Create, set panel
        JPanel panel = new JPanel();

        Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

        //Set panel
        int width = 700;
```



```
int height = 400;
panel.setBounds((dim.width-width)/2,
                (dim.height-height)/2, width, height);
panel.setLayout(null);

//Add buttons, labels, text fields to panel
panel.add(login);
panel.add(exit);
panel.add(username);
panel.add(password);
panel.add(userText);
panel.add(passText);

//Create, set frame
JFrame frame = new JFrame();
frame.setExtendedState(Frame.MAXIMIZED_BOTH);
//set width, height full
frame.setUndecorated(true); // disable decorations on frame
frame.getContentPane().setBackground(new Color(87, 147, 77));
frame.setVisible(true); //show window
frame.setLayout(null);

//Add panel to frame
frame.add(panel);
}
}
```

Nézzük meg részenként a kódot.

```
//Create, set buttons

//Login button
JButton login = new JButton("Login");
login.setBounds(50, 300, 275, 50);

//Exit button
JButton exit = new JButton("Exit");
exit.setBounds(375, 300, 275, 50);
exit.addActionListener(new ActionListener() {
@Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});
```

Ennél a résznél létrehoztam a Login és Exit gombokat és megadtam az Exit gombnak, hogy kilépjen a programból, ha rákattintanak.

```
//Create, set labels

//User name label
```

```
JLabel username = new JLabel("Username:");
username.setBounds(50, 50, 200, 75);
username.setFont(new Font("Arial", Font.PLAIN, 40));

//Password label
JLabel password = new JLabel("Password:");
password.setBounds(50, 150, 200, 75);
password.setFont(new Font("Arial", Font.PLAIN, 40));
```

Címkék létrehozása. Ezek tartalmazzák a username és password szöveget.

```
//Create, set text fields

//User text field
JTextField userText = new JTextField();
userText.setBounds(275, 55, 375, 60);

//Pass text field
JTextField passText = new JTextField();
passText.setBounds(275, 155, 375, 60);
```

Létrehozom a mezőt ahová a user írhat.

```
//Create, set panel
JPanel panel = new JPanel();

Dimension dim = Toolkit.getDefaultToolkit().getScreenSize();

//Set panel
int width = 700;
int height = 400;
panel.setBounds((dim.width-width)/2, (dim.height-height)/2, width, height);
panel.setLayout(null);
```

Egy panel létrehozása, ami magába fogja foglalni a gombokat, címkéket és text fieldeket.

```
//Add buttons, labels, text fields to panel
panel.add(login);
panel.add(exit);
panel.add(username);
panel.add(password);
panel.add(userText);
panel.add(passText);
```

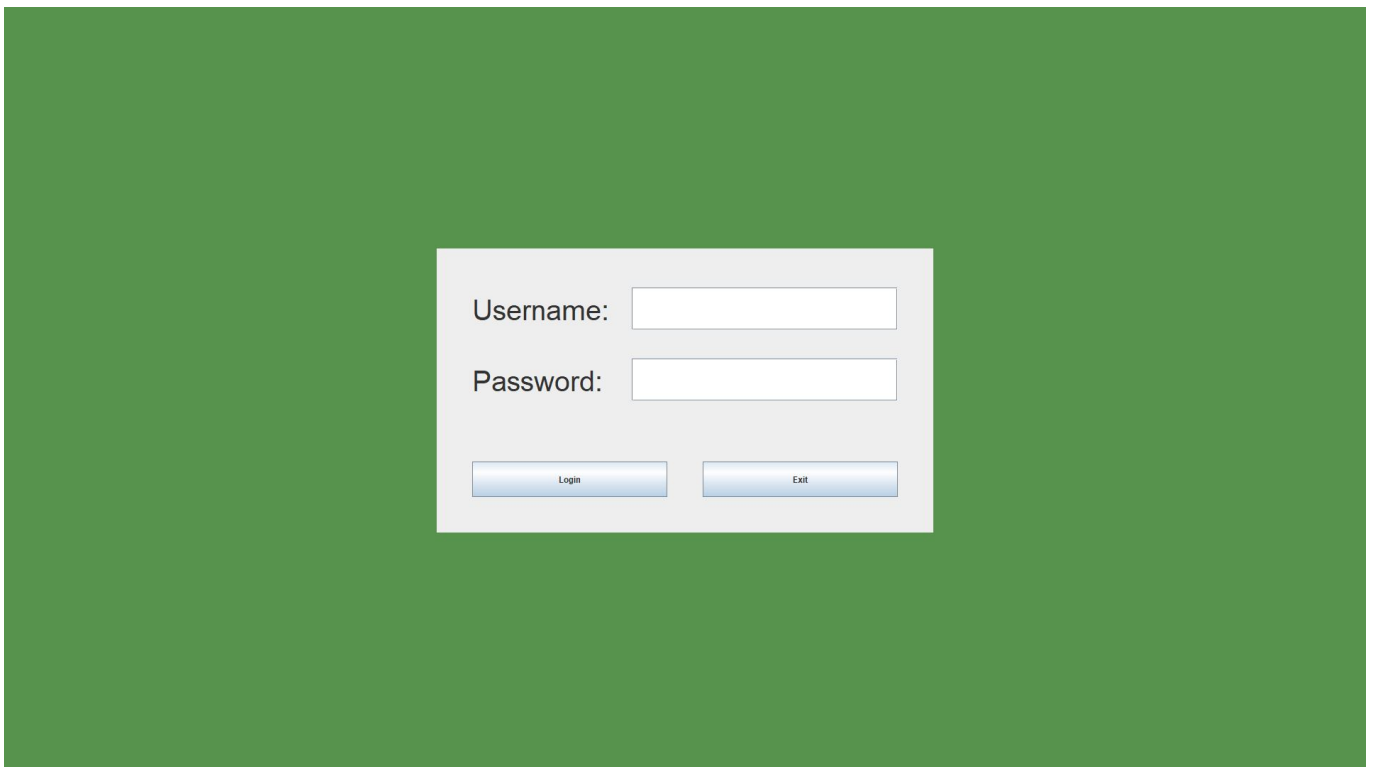
A panelhez hozzáadom a gombokat, címkéket, text fieldeket.

```
//Create, set frame
JFrame frame = new JFrame();
frame.setExtendedState(Frame.MAXIMIZED_BOTH); //set width, height full
frame.setUndecorated(true); // disable decorations on frame
frame.getContentPane().setBackground(new Color(87, 147, 77));
frame.setVisible(true); //show window
```

```
frame.setLayout (null);  
  
//Add panel to frame  
frame.add (panel);
```

A lényegi rész pedig itt következik. Létrehozom a framet vagyis az ablakot. A `setExtendedState`-el maximalizálom a szélességét és hosszúságát az ablaknak. A `setUndecorated(true)`-val válik a program ténylegesen teljes képernyőssé. Ez "tünteti" el az ablak fejlécét, így fullscreen lesz a program.

Az utolsó sorokban már csak színezés, megjelenítés van és hozzáadjuk a panelt a framehez. Így néz ki a program:



Paszigráfia Rapszódia OpenGL full screen vizualizáció

A program első fordítása nem volt túl sikeres. Sok könyvtárat kellett leszednem ahhoz, hogy működjön: `sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev`. Ez a parancs telepítette nekem a glut könyvtárat ami hiányzott a fordításhoz. A fordításhoz a következő parancsot kellett használni: `g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut -std=c++11`. A futtatáshoz pedig ezt:

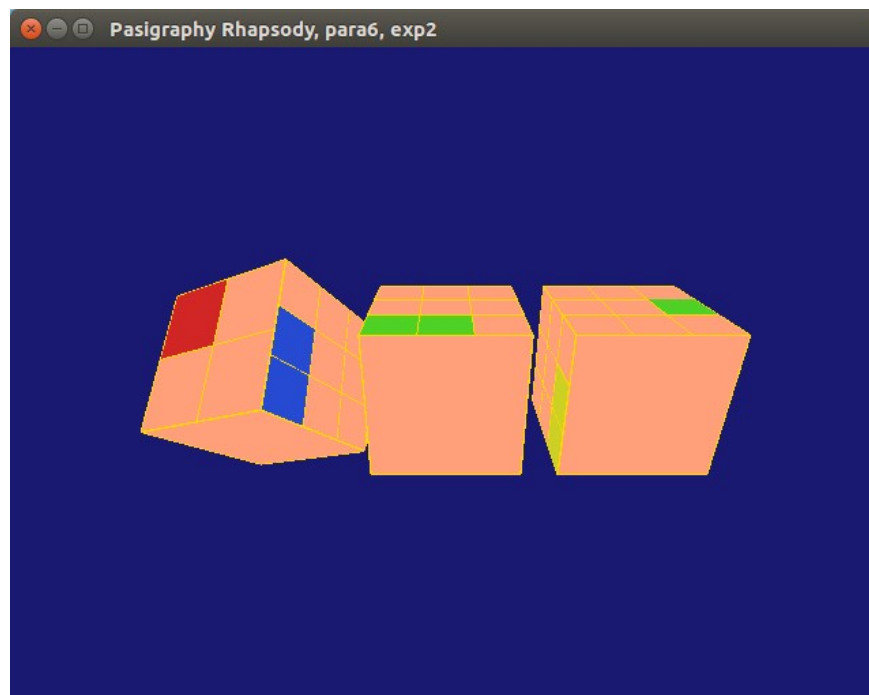
```
./para 3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:0:1:0:1:0:1:  
0:2:2:0:1:1:1:3:2:1:0:2:0:2:1:1:1:2:3:0:1:1:1:1:0:3:3:0:1:0:2:1:  
:0:1:0:2:2:0:0:0:1:3:1:0:1:3:2:1:0:2:0:3:3:0:1:0:2:1:0.
```

A program elindításakor rögtön feltűnt, hogy a kocka a nyilakkal ellentétes irányba mozdul. Egy kis keresgélés után rá lehetett jönni, hogy ebben a metódusban kell változtatni, hogy jól működjön. Csak annyit kellett, hogy ahol hozzáadok ott kivonok az értékből, ahol pedig kivonok ott hozzáadok az értékhez. Így az ellenkező irányba fog mozdulni minden.

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

    glutPostRedisplay();
}
```

A színezéssel játszadoztam még el. Ahhoz, hogy megváltoztassam a színét a glColor3f (R, G, B) metóduson kellett módosítanom a programban. Ez lett a végeredmény:



7. fejezet

Helló, Stroustrup!

JDK osztályok

Egy olyan programot kellett írni C++-ban boost használatával amely kilistázza a JDK zip tartalmát. Íme a forráskód:

```
#include <iostream>
#include <vector>
#include <boost/filesystem.hpp>

using namespace std;
using namespace boost::filesystem;

int main(int argc, char *argv[])
{
    path p("SrcZip");
    if(!exists(p) || !is_directory(p)) {
        cout << p << " is not a path" << endl;
        return 1;
    }

    int i=0;
    recursive_directory_iterator begin(p), end;
    vector<directory_entry> v(begin, end);
    for(auto& f:v) {
        if(path(f).has_extension()) {
            cout << "    -" <<path(f) .filename()<< endl;
            i++;
        } else {
            cout << f << endl;
        }
    }
    cout << "Összes fájl:" << i << endl;
}
```

Részenként elemezve a program:

```
#include <iostream>
#include <vector>
#include <boost/filesystem.hpp>

using namespace std;
using namespace boost::filesystem;
```

Először is includeolom a megfelelő osztályokat, közöttük a boost/filesystem.hpp, amely majd a path-al kapcsolatos műveletekben fog segíteni.

```
path p("SrcZip");
if(!exists(p) || !is_directory(p)) {
    cout << p << " is not a path" << endl;
    return 1;
}
```

Megadom pathnak a mappa nevét, amelyben a kicsomagolt src.zip fájl van. Valamint ellenőrzöm, hogy létezik-e megadott útvonal, ha nem kiíratom, hogy nincs ilyen útvonal.

```
int i=0;
recursive_directory_iterator begin(p), end;
vector<directory_entry> v(begin, end);
for(auto& f:v) {
    if(path(f).has_extension()) {
        cout << "    -" <<path(f) .filename()<< endl;
        i++;
    } else {
        cout << f << endl;
    }
}
cout << "Összes fájl:" << i << endl;
```

A recursive_directory_iterator egy directory_entry elemein az összes alkönyvtár bejegyzésén rekúrázván iterál. Minden könyvtárbejegyzést csak egyszer látogat meg. Egy vectorba elmentem az összes directory_entryt. Ezt átadom egy for-each ciklusnak. Egy if-el megvizsgálom, hogy az éppen vizsgált elérési útvonalnak van-e fájl a végén. Ha van beljebb húzom és kiíratom a fájl nevét, ha nincs akkor csak a mappa nevét íratom ki. Az i-vel pedig az összes fájlt számlálom meg. A program fordítása és futtatása:

```
bunyi@bunyi: ~/Documents/JDK
bunyi@bunyi:~/Documents/JDK$ g++ JdkClass.cpp -o JdkClass -std=c++11 -lboost_system -lboost_filesystem
bunyi@bunyi:~/Documents/JDK$ ./JdkClass
```

A program végeredménye:

```
bunyi@bunyi: ~/Documents/JDK
- "Sequencer.java"
- "MidiUnavailableException.java"
- "MidiChannel.java"
- "SoundbankResource.java"
"SrcZip/launcher"
- "jli_util.h"
- "java.h"
- "java_md.h"
- "splashscreen.h"
- "manifest_info.h"
- "java.c"
- "version_comp.c"
- "java_md.c"
- "wildcard.h"
- "parse_manifest.c"
- "wildcard.c"
- "defines.h"
- "splashscreen_stubs.c"
- "main.c"
- "version_comp.h"
- "emessages.h"
- "jli_util.c"
Összes fájl:7718
bunyi@bunyi:~/Documents/JDK$
```

Másoló-mozgató szemantika és Összefoglaló

Kódcsipeteken keresztül kellett összevetni a C++11 másoló és mozgató szemantikáját, valamint a mozgató konstruktort a mozgató értékadásra kellett alapozni. Íme a teljes kód ami bemutatja mind a másoló konstruktort, másoló értékadást, mozgató konstruktort és mozgató értékadást:

```
#include <iostream>
#include <algorithm>
```

```
using namespace std;

class program{
private:
    int a;
    int* b;

public:
    program(int elem1, int& elem2) : a(elem1), b(new int){           // ←
        konstruktor 2 paraméterrel
        cout << "\tLefutott a default konstruktor!" << endl;
        *b = elem2;
    }

    program(const program &adott){           //másoló konstruktor
        cout << "\tLefutott a másoló konstruktor!" << endl;
        a = adott.a;
        b = new int;
        *b = *adott.b;
    }

    program& operator= (program &adott){           //másoló értékadás
        cout << "\tMásoló értékadás történt!" << endl;
        a = adott.a;
        *b = *adott.b;
        return *this;
    }

    program(program && adott){           //mozgató konstruktor
        cout << "\tLefutott a mozgató konstruktor!" << endl;
        a = 0;
        b = nullptr;
        *this = move(adott);
    }

    program& operator= (program && adott){           //mozgató értékadás
        cout << "\tMozgató értékadás történt!" << endl;
        swap(b,adott.b);
        swap(a,adott.a);

        return *this;
    }

    void Print(){
        if(b!=NULL)
            cout << "Az a értéke: " << a << ", A b értéke " << *b << ", és ←
                b " << b << "-re mutat\n" << endl;
        else
```



```
        cout << "Az a értéke: " << a << ", A b értéke " << b << "\n" << ↵
        endl;
    }

    ~program(){          //destruktor
        cout << "\tLefutott a destruktor!" << endl;
        delete b;
    }
};

int main(){

    int Nyolc = 8;
    int Kilenc = 9;

    cout << "Alap létrehozása 10, Nyolc értékekkel:" << endl;
    program Alap(10, Nyolc);
    cout << "Alap ertekei:" << endl;
    Alap.Print();

    cout << "\n\n-----Másoló konstruktor-----" << endl;
    cout << "Alap_masolat létrehozása Alap alapján:" << endl;
    program Alap_masolat(Alap);
    cout << "Alap ertekei:" << endl;
    Alap.Print();
    cout << "Alap_masolat értékei" << endl;
    Alap_masolat.Print();

    cout << "\n\nUj1 létrehozása 20, Kilenc értékekkel:" << endl;
    program Uj1(20, Kilenc);
    cout << "Uj1 ertekei:" << endl;
    Uj1.Print();

    cout << "\n\n-----Másoló értékadás-----" << endl;
    cout << "Alap_masolat = Uj1:" << endl;
    Alap_masolat = Uj1;
    cout << "Uj1 ertekei:" << endl;
    Uj1.Print();
    cout << "Alap_masolat ertekei:" << endl;
    Alap_masolat.Print();

    cout << "\n\n-----Mozgató konstruktor és értékadás-----" << ↵
        endl;
    cout << "Uj2 létrehozása mozgató konstruktorral:" << endl;
    program Uj2(move(Alap));
    cout << "Alap ertekei:" << endl;
    Alap.Print();
    cout << "Uj2 ertekei:" << endl;
    Uj2.Print();
```

```
        cout << "\n\nProgram vége:" << endl;

    return 0;
}
```

De nézzük elemezve a kódot.

```
private:
    int a;
    int* b;
```

A program class private részében először is van két változó. Egy alap int típusú ez lesz az "a". A másik egy pointer amit * jelölünk ez a "b".

```
program(int elem1, int& elem2) : a(elem1), b(new int){
    cout << "\tLefutott a default konstruktor!" << endl;
    *b = elem2;
}
```

A legelső egy default két paraméteres konstruktor. Itt csak annyi történik, hogy amikor ez meghívódik akkor az "a"-t egyenlővé teszi az első elemmel, a "b" viszont mivel pointer jelölni kell egy *-al, így mentjük el az értékébe az elem2-t. A kiíratás egy nyomkövető üzenet.

```
program(const program &adott){
    cout << "\tLefutott a másoló konstruktor!" << endl;
    a = adott.a;
    b = new int;
    *b = *adott.b;
}
```

Ezután a sorban a másoló konstruktor következik. Ezt akkor használjuk amikor úgy inicializálunk egy osztályt, hogy egy már kész ugyanolyan típusú osztály példány értékeit másoljuk át. Ez a konstruktor annyira fontos a C++-ban, hogy ha nem hozunk létre ilyet, automatikusan létre fog jönni egy. Azonban ezek a fordító által létrehozott konstruktorok csak sekély másolatot készítenek. Ez azt jelenti, hogy csak az eredeti objektumra való hivatkozást másolja és nem annak az értékét.

Ezért ha pointerekkel és referenciákkal dolgozunk csak maga a pointer másolódik, ekkor kell létrehoznunk egy saját mélymásoló konstruktort. A mélymásoló külön memóriát foglal le a másolt információk számára. Vagyis a forrás és a másolat különböznek. Ha a mélymásolásnál megváltoztatjuk a másolat értékét akkor a forrás nem változik, nem úgy mint a sekély másolásnál. Tehát az egyik memóiahelyen végrehajtott módosítások nem befolyásolják a másikat. Ha pedig dinamikus memóriát pointerek segítségével allokálunk, akkor kell a felhasználó által definiált másoló konstruktor, hogy mind a két objektum különböző memóiahelyekre mutasson.

Egy ilyen konstruktor látható fent is. A paraméterében egy objektumra vonatkozó referenciát kell tartalmaznia másképpen végtelen ciklus jön létre. A törzsben jelenleg egy nyomkövető üzenet van, egy sima int típusú érték átmásolás, valamint a b pointerbe másoljuk a másolandó példány b értékét.

```
program& operator= (program &adott){
    cout << "\tMásoló értékadás történt!" << endl;
    a = adott.a;
    *b = *adott.b;
```

```
    return *this;
}
```

A másoló értékadást akkor használjuk amikor egy osztály értékeit egy másik már létező osztály értékeibe szeretnénk másolni. Meghívása: `osztaly1 =osztaly2` Ez is ugyanúgy mint a másoló konstruktornál automatikusan létrejön a fordításkor, de ugyanaz lesz a helyzet itt is csak sekély másolás történik, így ahhoz, hogy pointerekkel és referenciákkal tudjunk dolgozni itt is létre kell hozni egy saját másoló értékadást.

Ebben az esetben mint a másoló konstruktornál van egy sima int érték átmásolás és a b pointerbe másoljuk a másolandó példány értékét.

```
program(program && adott){
    cout << "\tLefutott a mozgató konstruktor!" << endl;
    a = 0;
    b = nullptr;
    *this = move(adott);
}

program& operator= (program && adott){
    cout << "\tMozgató értékadás történt!" << endl;
    swap(b,adott.b);
    swap(a,adott.a);

    return *this;
}
```

Míg a másoló konstruktor és másoló értékadás célja, hogy másolatot készítsen egyik objektumról a másikra, addig a mozgató konstruktor és értékadásé, hogy átruházza az értékeket egyikről a másik objektumra. Ez a művelet sokkal kevesebb erőforrást igényel, mint a másolás, mivel itt nem kell lemásolni az értékeket csak átadni neki. Eközben az eredeti elveszti a tartalmát és használhatatlanná válik.

A mozgató konstruktor és értékadás definiálása analóg történik. A másoló szemantika esetén a paraméter egy konstans bal érték referencia, amíg a mozgató szemantikánál egy nem konstans jobb érték referenciáról beszélünk. Ezt láthatjuk is a programban mivel a referencia & jellel van jelölve a paraméterben.

A mozgató értékadás csak annyit tesz, hogy felcseréli a két példány tagjait a már beépített swap módszerrel. A mozgató konstruktort pedig a mozgató értékadásra kellett alapozni. Ez úgy történik, hogy amikor meghívódik a konstruktor akkor amelyikbe mozgatni fogjuk az értékeket annak először kinullázzuk az értékeit, majd a `*this =move(adott);` sorral meghívjuk a másoló értékadás függvényt, ez felcseréli a példányok értékeit, így a forrás kapja a nullákat és a másik a forrás értékeit.

```
void Print(){
    if(b!=NULL)
        cout << "Az a értéke: " << a << ", A b értéke " << *b << ", és b " << b << "-re mutat\n" << endl;
    else
        cout << "Az a értéke: " << a << ", A b értéke " << b << "\n" << endl;
}

~program(){
```

```
    cout << "\tLefutott a destruktorktor!" << endl;
    delete b;
}
```

A program osztályban ezután csak egy print metódus van, hogy szebben láthassuk mi is történik. Az "a" értékét írjuk ki, majd a "b" értékét, majd a memóriacímet ahová a "b" mutat. Valamint kell még egy destruktorktor, ami törli a pointereket a program végén, ezzel memóriát szabadítva fel.

```
int Nyolc = 8;
int Kilenc = 9;

cout << "Alap létrehozása 10, Nyolc értékekkel:" << endl;
program Alap(10, Nyolc);
cout << "Alap ertekei:" << endl;
Alap.Print();
```

A main metódusban először létrehozzuk az Alap példányt amit másolni fogunk és kiíratjuk az értékeit.

```
cout << "\n\n-----Másoló konstruktor-----" << endl;
cout << "Alap_masolat létrehozása Alap alapjan:" << endl;
program Alap_masolat(Alap);
cout << "Alap ertekei:" << endl;
Alap.Print();
cout << "Alap_masolat értékei" << endl;
Alap_masolat.Print();
```

Ezután lemásoljuk az Alap példányt az Alap_masolatba és kiíratjuk az értékeit.

```
cout << "\n\nUj1 létrehozása 20, Kilenc értékekkel:" << endl;
program Uj1(20, Kilenc);
cout << "Uj1 ertekei:" << endl;
Uj1.Print();

cout << "\n\n-----Másoló értékadás-----" << endl;
cout << "Alap_masolat = Uj1:" << endl;
Alap_masolat = Uj1;
cout << "Uj1 ertekei:" << endl;
Uj1.Print();
cout << "Alap_masolat ertekei:" << endl;
Alap_masolat.Print();
```

Létrehozunk egy Uj1 nevű példányt, amibe másoló értékadással átmásoljuk az Alap_masolat értékeit.

```
cout << "\n\n-----Mozgató konstruktor és értékadás-----" << endl;
cout << "Uj2 létrehozása mozgató konstruktorral:" << endl;
program Uj2(move(Alap));
cout << "Alap ertekei:" << endl;
Alap.Print();
cout << "Uj2 ertekei:" << endl;
Uj2.Print();
```

Már csak a mozgató konstruktor és értékadás maradt. Ezzel a sorral: `program Uj2(move(Alap));` hívjuk meg a konstruktort. Mivel a beépített `move` függvény az `Alap`-ból jobb érték referenciát hoz létre, így tudjuk meghívni.

A program futás után:

```
bunyi@bunyi: ~/Documents/Copy_test
Alap létrehozása 10, Nyolc értékekkel:
    Lefutott a default konstruktor!
Alap ertekei:
Az a értéke: 10, A b értéke 8, és b 0x1cd5030-re mutat

-----Másoló konstruktor-----
Alap_masolat létrehozása Alap alapján:
    Lefutott a másoló konstruktor!
Alap ertekei:
Az a értéke: 10, A b értéke 8, és b 0x1cd5030-re mutat

Alap_masolat értékei
Az a értéke: 10, A b értéke 8, és b 0x1cd5050-re mutat

Uj1 létrehozása 20, Kilenc értékekkel:
    Lefutott a default konstruktor!
Uj1 ertekei:
Az a értéke: 20, A b értéke 9, és b 0x1cd5070-re mutat

-----Másoló értékadás-----
Alap_masolat = Uj1:
    Másoló értékadás történt!
Uj1 ertekei:
Az a értéke: 20, A b értéke 9, és b 0x1cd5070-re mutat

Alap_masolat ertekei:
Az a értéke: 20, A b értéke 9, és b 0x1cd5050-re mutat

-----Mozgató konstruktor és értékadás-----
Uj2 létrehozása mozgató konstruktorral:
    Lefutott a mozgató konstruktor!
    Mozgató értékadás történt!
Alap ertekei:
Az a értéke: 0, A b értéke0

Uj2 ertekei:
Az a értéke: 10, A b értéke 8, és b 0x1cd5030-re mutat

Program vége:
    Lefutott a destruktor!
    Lefutott a destruktor!
    Lefutott a destruktor!
```

Ahogy látszik a másoló konstruktornál az értékek ugyanazok a másolat példányánál, de a memória cím más. A másoló értékadásnál ugyanez látható. A mozgató konstruktornál és értékadásnál pedig az látszik, hogy a forrást nullára állítottuk és az `Uj2` kapta meg az értékeket.

Ahogy látszik, ha jól megtanuljuk az alapokat nem okozhat nehézséget ezeknek a konstruktoroknak és értékadásoknak a használata, sőt megkönnyíthetik az egyes programok működését ha megfelelően használjuk őket.

Változó argumentumszámú ctor

Egy olyan példát kellett készítenünk, amely egy képet tesz a Perceptron osztály bemenetére és nem egy értéket, hanem egy ugyanakkora méretű képet ad vissza. Először is nézzük meg a konstruktorunkat:

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

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

        va_list vap;

        va_start ( vap, nof );

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

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

        va_end ( vap );

        weights = new double**[n_layers-1];

#ifdef RND_DEBUG
        std::random_device init;
        std::default_random_engine gen {init() };
#else
        std::default_random_engine gen;
#endif

        std::uniform_real_distribution<double> dist ( -1.0, 1.0 );

        for ( int i {1}; i < n_layers; ++i )
        {
            weights[i-1] = new double *[n_units[i]];

            for ( int j {0}; j < n_units[i]; ++j )
            {
                weights[i-1][j] = new double [n_units[i-1]];

                for ( int k {0}; k < n_units[i-1]; ++k )
                {
                    weights[i-1][j][k] = dist ( gen );
                }
            }
        }
    }
};
```

```

    }
}

```

Ahogy látható a konstruktor paraméter mezőjében egy nem szokványos megadás van. `Perceptron (int nof, ...)` Ezt nevezzük változó paraméter számú konstruktornak. Az első szám azt fogja megadni, hogy hány paraméter lesz még utána.

```

double* operator() ( double image [] )
{
    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
    {

#ifdef CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );

#else

#pragma omp parallel for
        for ( int j = 0; j < n_units[i]; ++j )
        {
            units[i][j] = 0.0;

            for ( int k = 0; k < n_units[i-1]; ++k )
            {
                units[i][j] += weights[i-1][j][k] * units[i-1][k];
            }

            units[i][j] = sigmoid ( units[i][j] );
        }

#endif

    }

    for (int i = 0; i < n_units[n_layers - 1]; i++) {
        image[i] = units[n_layers - 1][i];
    }

    return image;
    //return sigmoid ( units[n_layers - 1][0] );
}

```

Itt pedig az `operator()` függvény látható, amelyben csak annyit kellett megváltoztatni, hogy ne egy értékkel

térjen vissza, hanem egy tömbbel.

```
#include <iostream>
#include "mlp.hpp"
#include <png++/png.hpp>
#include <fstream>

int main (int argc, char **argv){
    png::image <png::rgb_pixel> png_image (argv[1]);

    int size = png_image.get_width()* png_image.get_height();

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

    double* image = new double[size];

    for (int i {0}; i<png_image.get_width();++i)
        for (int j {0}; j<png_image.get_height();++j)
            image[i*png_image.get_width()+j]= png_image[i][j].red;

    double* newPNG = (*p) (image);

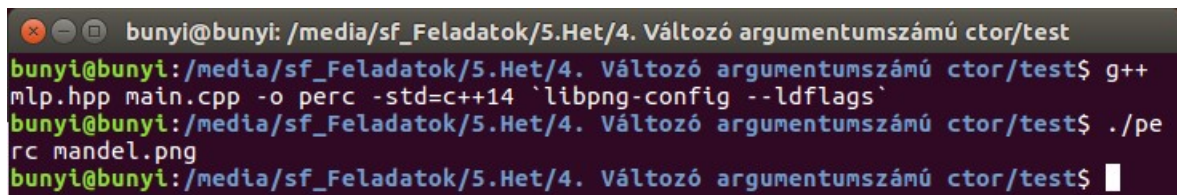
    for(int i = 0; i < png_image.get_width(); ++i)
        for(int j = 0; j < png_image.get_height(); ++j)
            png_image[i][j].green = newPNG[i*png_image.get_width()+j];

    png_image.write("output.png");

    delete p;
    delete [] image;
}
```

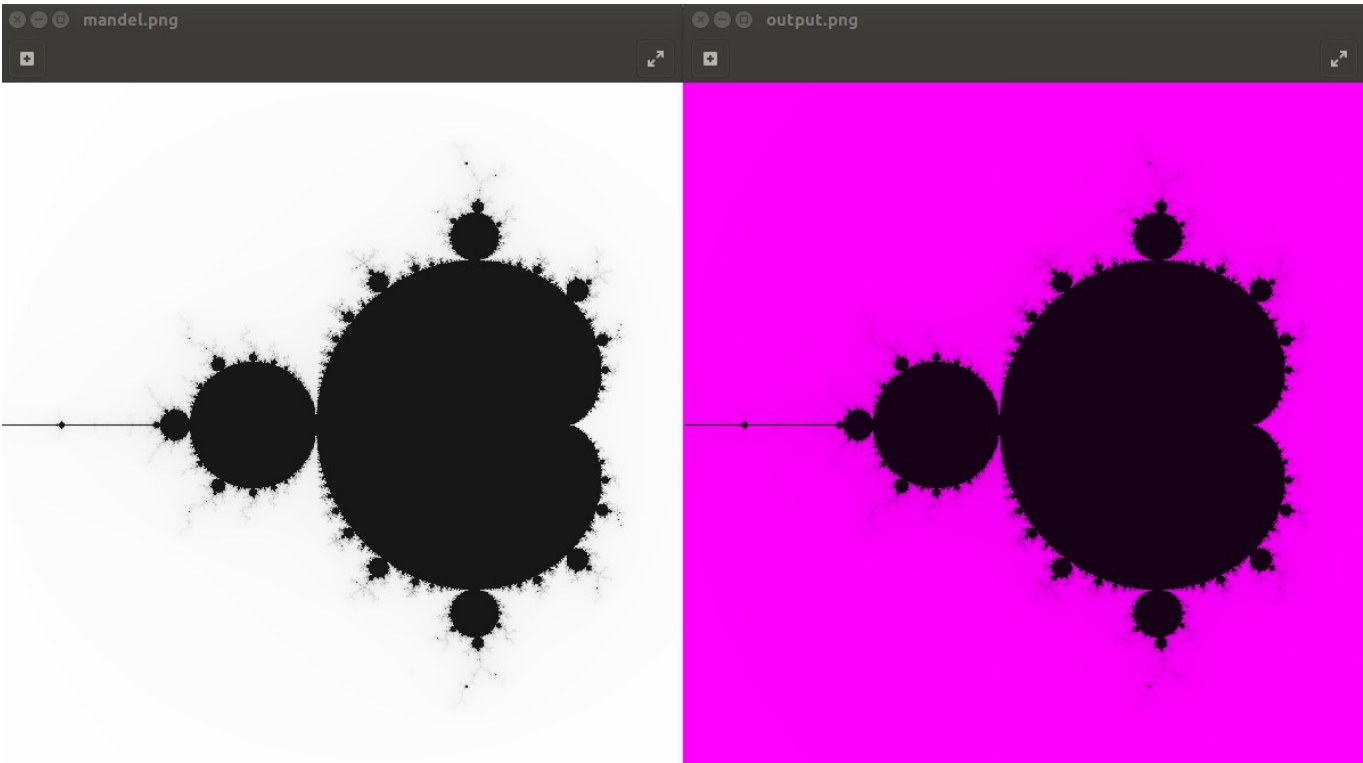
A main metódusban először beolvassuk a képet. Létrehozzuk a size-ot, amely a kép pixel méretét fogja tartalmazni. Majd példányosítjuk a Perceptron osztályt. Itt látszik a változó paraméterű konstruktor haszna. Ezután végig megyünk a kép szélességén és magasságán, a megfelelő piros értékeket kiírjuk az image tömbbe. A második for ciklussal felülírjuk a zöld értékeket. Végül kimentjük az új képet és felszabadítjuk a memóriát.

A program fordítása és futtatása:



```
bunyi@bunyi: /media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test
bunyi@bunyi:/media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test$ g++
mlp.hpp main.cpp -o perc -std=c++14 `libpng-config --ldflags`
bunyi@bunyi:/media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test$ ./pe
rc mandel.png
bunyi@bunyi:/media/sf_Feladatok/5.Het/4. Változó argumentumszámú ctor/test$
```

Az eredeti és az eredményül kapott kép:



8. fejezet

Helló, Gödel!

Gengszterek

C++11 Custom Allocator

STL map érték szerinti rendezése

Alternatív Tabella rendezése

Prolog családfa

GIMP Scheme hack

9. fejezet

Helló, Valaki!

FUTURE tevékenység editor

OOCWC Boost ASIO hálózatkezelése

SamuCam

BrainB

OSM térképre rajzolása

10. fejezet

Helló, Schwarzenegger!

Port scan

AOP

Android Játék

Junit teszt

11. fejezet

Helló, Calvin!

MNIST

Deep MNIST

CIFAR-10

Android telefonra a TF objektum detektálója

SMNIST for Machines

Minecraft MALMO-s példa

III. rész

Irodalomjegyzék

Általános

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

C

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

C++

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

Lisp

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