

# Univerzális programozás

---

**Írd meg a saját programozás tankönyvedet!**

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

## COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Szűcs, Levente	2019. október 23.	

## 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 <a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai

## Ajánlás

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

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

# Tartalomjegyzék

<b>I. Második felvonás</b>	<b>1</b>
<b>1. Helló, Berners-Lee!</b>	<b>3</b>
1.1. 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.	3
1.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. I--II.II.	3
<b>2. Helló, Liskov!</b>	<b>5</b>
2.1. Liskov helyettesítés sértése	5
2.2. Szülő-gyerek	6
2.3. Anti OO	8
2.4. Ciklomatikus komplexitás	9
<b>3. Helló, Mandelbrot!</b>	<b>10</b>
3.1. Reverse engineering UML osztálydiagram	10
3.2. Forward engineering UML osztálydiagram	11
3.3. BPMN	12
3.4. BPEL Helló, Világ! egy visszhang folyamat	13
<b>4. Helló Chomsky!</b>	<b>15</b>
4.1. Encoding	15
4.2. Full screen	18
4.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	21
<b>5. Helló, Stroustrup</b>	<b>24</b>
5.1. JDK osztályok	24
5.2. Másoló-mozgató szemantika	28
5.3. Változó argumentum ctor	28

<b>II. Irodalomjegyzék</b>	<b>33</b>
5.4. Általános . . . . .	34
5.5. C . . . . .	34
5.6. C++ . . . . .	34
5.7. Lisp . . . . .	34

DRAFT

# Táblázatok jegyzéke

2.1. Összehasonlítás . . . . .	9
--------------------------------	---

# Előszó

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

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

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

## Hogyan forgasd

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

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

## Milyen nyelven nyomjuk?

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

## Hogyan nyomjuk?

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



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

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml  ←
--noout
output.xml validates
rm -f output.xml
dlatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
=====
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dlatex' 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**

## **Második felvonás**

**Bátf41 Haxor Stream**

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

---

DRAFT

# 1. fejezet

## Helló, Berners-Lee!

### 1.1. 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 Java és a C++ magasszintű, objektumorientált programozási nyelv. Nagyon sok hasonlóság van a két nyelvben, hiszen a Java nagyon sok dolgot vett át a C++ tól. A Java viszont újabb ezért a fejlesztők sokat dolgoztak azon, hogy a Java kódok megbízhatóbbak, biztonságosabb és emelett platformfüggetlenek legyenek. Mint említettem, a két nyelv objektumorientált, de a C++ nem kötelez arra hogy osztályokba dolgozzunk, ezzel szembe a Javában csak objektumorientáltan lehet dolgozni.

A Java és a C++ szintaxisa nagyon hasonló, hiszen a Java a C és a C++ szintaxisán alapszik. Természetesen vannak apróbb különbségek. Később kifejtem...

A két nyelv mint említettem nagyon hasonlít azonban van egy nagy különbség. A memóriakezelés a C++-ban manuális míg Javában a memóriakezelést a Garbage collection "szemét gyűjtő" végzi ezt a munkát. A Javában a rendszer önmaga kezeli a memóriát és ha rendszer úgy érzi, hogy betelik a memória, akkor felszabadít memóriát. A C++-ban a programozónak kell a memóriát kezelnie. A C++-ban destruktorokat alkalmazunk ahhoz, hogy memóriát szabadítsunk fel. Természetesen mindkettőnek megvannak előnyei, hátrányai. Javában az előny az, hogy nem kell nekünk kezelni ezeket a memória felszabadításokat, amely egy nagyobb projektnél egy nagyon kényelmes funkció, azonban mi ezt nem tudjuk irányítani, ezért lehetséges az, hogy feleslegesen tud nagyobb méretű memóriát foglalni a rendszer egy kisebb programhoz, mint amennyire aktuálisan szükségünk van rá.

### 1.2. Olvasónapló: Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. I--II.II.

Guido van Rossum 1990-ben a Pythont, amely egy általános célú magasszintű programozási nyelv. Ez a nyelv a fejlesztők számára rengeteg pozitív tulajdonsággal rendelkezik ezek közül pl:dinamikus,objektumorientált és a platformfüggetlenség. Python tulajdonképpen egy szkriptnyelv, azonban rendkívül sok csomagot és beépített eljárását tartalmaz, ezért összetett alkalmazások készítésére is alkalmas.

Nagyon sok platformokon érhető el pl: Win, Mac, Unix, S60, UIQ, Iphone. A Python egy köztes nyelv, nincs szükség fordításra és linkelésre. Az értelmező (interpreter) interaktívan is használható amelynek segítségével például egyszerűen kezelhetünk (throw-away) programokat is bizonyos funkciók kipróbálására.

A nyelv legfőbb jellemzője, hogy behúzásalapú a szintaxisa. A programban szereplő állításokat az azonos szintű behúzásokkal tudjuk csoportokba szervezni, nincs szükség kapcsos zárójelre vagy explicit kulcsszavakra(pl.begin,end). Fontos, hogy a behúzásokat egységesen kezeljük, tehát vagy mindenhol tabot, vagy egységesen szóközt használjuk.

A nyelv további sajátossága, hogy a sor végéig tart egy utasítás, nincs szükség a megszokott ';' használatára. Ha egy utasítás több sorban fér csak el, akkor ezt a sor végére '\n' (backslash) jellel lehet jelezni. Amennyiben nem zártunk be minden nyitott zárójelet akkor az utasítás folytatósorának veszi a következő sort. Az értelmező minden egyes sort úgynevezett tokenekre bont, amelyek közt tetszőleges üres whitespace karakter lehet. Tokenek fajtái:azonosító kulcsszó,operátor,literál. Kulcsszavakat itt is is alkalmazunk pl : lambda,return,while,try stb.

Pythonban minden adatot objektumok reprezentálnak. Pythonban nincs szükség a változók típusainak explicit megadására,a rendszer futási időben,automatikusan „kitalálja” a változók típusát. Az adattípusok a következők lehetnek: számok, sztringek,ennek (tuples,n-es), listák, szótárak. A számok lehetnek egészek, lebegőpontosak és komplex számok. Az egész számok lehetnek decimálisak,oktálisak.

## 2. fejezet

# Helló, Liskov!

### 2.1. Liskov helyettesítés sértése

Ebben a feladatban a Liskov helyettesítés sértésére kellett programot írunk. A híres S.O.L.I.D programozásnak (Tiszta kódolás) az L betűje a Liskov-elv amely arról szól, hogy minden osztályt lehet helyettesíteni a leszármazott osztályával, amellyel a program alapvető működése nem változik. Ebben a feladatban segítségül kaptuk az Udprog repójában lévő forrást. Ez alapján csináltam egy saját programot. Íme :

```
class Jarmuvek{
public:
    virtual void gurul(){};
};

class Program {
public:

    void fgv ( Jarmuvek &jarmuvek ) {
        jarmuvek.gurul();
    }
};

class Car : public Jarmuvek
{};

class Hajo : public Jarmuvek
{};

int main ( int argc, char **argv )
{
    Program program;
    Jarmuvek jarmuvek;
    program.fgv ( jarmuvek);

    Car car;
    program.fgv ( car );
}
```

```
Hajo hajo;  
program.fgv ( hajo );  
  
}
```

Ebben a kódban a Járművek osztály az űsosztály és ebben a gyermekosztályok az autók és a hajók. Itt van egy gurul függvény, amely a problémát okozza, itt minden egyes gyerekosztályra utal azaz a Carra és a hajóra. A Carral semmi gond azonban, a hajó nem tud gurulni, viszont ez így nem igaz ezért a Liskov-elv sérül.

## 2.2. Szülő-gyerek

Ebben a feladatban a szülő és gyerek osztály kapcsolatáról foglalkozunk. Ez a feladat a polimorfizmussal és leginkább az öröklődéssel foglalkozik.

Az objektumorientált programozás lényegét az adatabsztrakció, öröklődés és a polimorfizmus szavakkal foglalkozhatjuk össze. Az öröklődés legegyszerűbb esete amikor egy osztályt egy már létező osztály kiterjesztésével definiálunk.

C++ és Javában kellett egy egyszerű programot írunk, ahol azt kellett demonstrálni, hogy a szülő osztályon csak a szülő üzeneteit lehet küldeni, a gyerek osztálynak az elemeit pedig nem.

Itt látható a kód C++-ban :

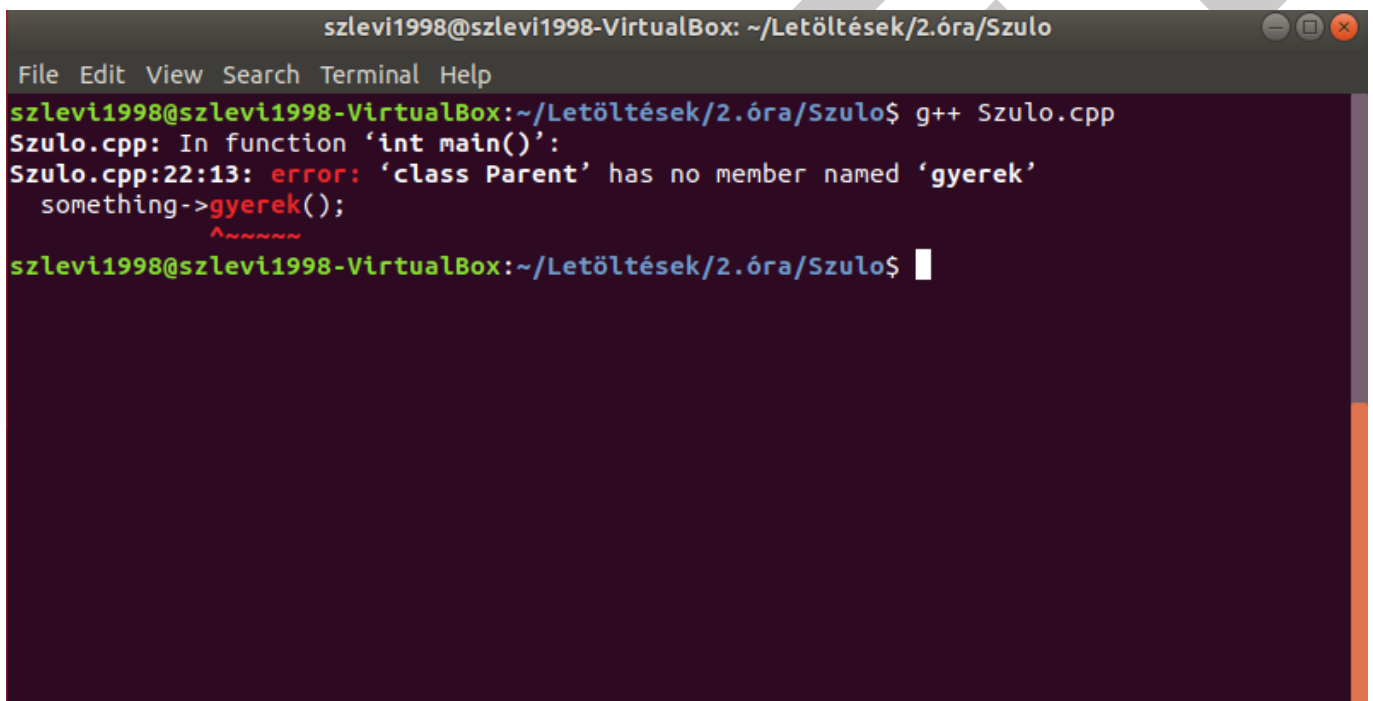
```
    ]  
#include <iostream>  
using namespace std;  
  
class Parent {  
public :  
    void parent() {  
        std::cout << " Ez a szülő osztály " << std::endl;  
    }  
};  
  
class Gyerek : public Parent {  
public :  
    void gyerek() {  
        std::cout << " Ez a gyerek osztály " << std::endl;  
    }  
};  
  
int main(){  
    Parent* something = new Gyerek();  
    something->parent();  
    something->gyerek();  
}
```

Amint látható a kódon van két osztályunk a Parent és a Gyerek osztály.

```
class Gyerek : public Parent {
```

Ebben a sorban látható az, hogy a Gyerek osztálynak átadjuk a Szülő osztályfunkcióit. A c++-ban így lehet átadni egy gyerek osztálynak a szülő osztály funkcióit.

Ezután a mainben példányosítunk és meghívjuk ezeket a függvényeket. A gyerek tudja használni a saját és a szülő osztály funkciót, azonban a szülő csak a saját funkcióit tudja alkalmazni, a gyereket viszont nem. Amint látható a program emiatt a hiba miatt nem is tud lefordulni.

A screenshot of a terminal window titled 'szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo'. The terminal shows the command 'g++ Szulo.cpp' being executed. The output indicates an error in 'Szulo.cpp' at line 22, column 13: 'error: 'class Parent' has no member named 'gyerek''. The error message points to the line 'something->gyerek();'. The terminal window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo$ g++ Szulo.cpp
Szulo.cpp: In function 'int main()':
Szulo.cpp:22:13: error: 'class Parent' has no member named 'gyerek'
  something->gyerek();
             ^~~~~~
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo$
```

Itt van a kód Javában is :

```
class Parent {

    void parent() {
        System.out.println("Ez a szülő osztály");
    }
}

class Gyerek extends Parent {
    void gyerek() {
        System.out.println("Ez a gyerek osztály");
    }
}
```

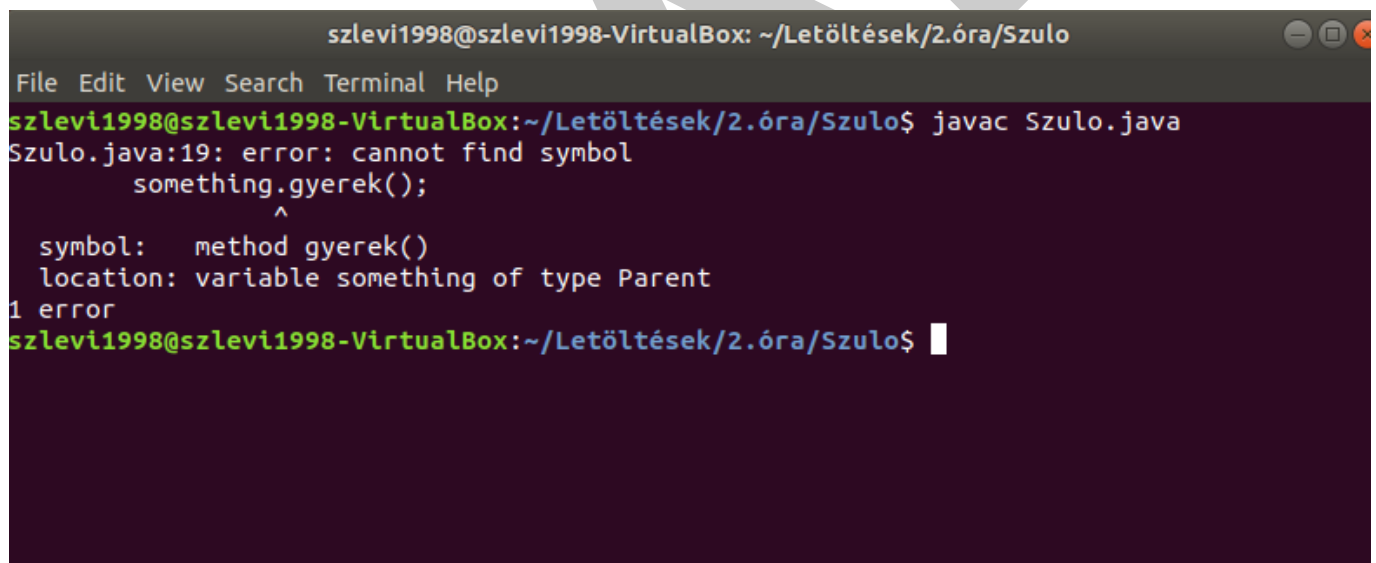


```
class Szulo {  
    public static void main(String[] args){  
        Parent something = new Gyerek();  
  
        something.parent();  
        something.gyerek();  
    }  
}
```

Amint látható egy nagyobb különbség található meg a C++ és a Java közötti programban. A két nyelv szintaktikája alapvetően nagyon hasonlít egymásra. Azonban az öröklődés egy kicsit különbözik.

```
class Gyerek extends Parent
```

Amint látható az öröklődést itt az extends kulcsszóval lehet alkalmazni. Amint látható, nincs eltérés, hiszen a gyerek képes alkalmazni a saját és a szülő osztály funkciót, és itt is a szülő csak a saját funkcióit tudja elérni, a gyereket nem.

A screenshot of a terminal window titled 'szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo'. The terminal shows the command 'javac Szulo.java' being executed. The output is an error message: 'Szulo.java:19: error: cannot find symbol something.gyerek();' with an arrow pointing to the 'gyerek()' part. Below this, it says 'symbol: method gyerek()' and 'location: variable something of type Parent'. At the bottom, it says '1 error'.

```
szlevi1998@szlevi1998-VirtualBox: ~/Letöltések/2.óra/Szulo  
File Edit View Search Terminal Help  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/2.óra/Szulo$ javac Szulo.java  
Szulo.java:19: error: cannot find symbol  
    something.gyerek();  
                ^  
symbol:   method gyerek()  
location: variable something of type Parent  
1 error  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/2.óra/Szulo$
```

## 2.3. Anti OO

Ebben a feladatban a BBP algoritmus futási időit hasonlítgatjuk össze. Sajnos az erősebb gépen a Linux nagyon sokat crashel ezért nem tudok teljes eredményt mutatni, ezért csak virtualboxos adatot tudok felmutatni. Az alábbi eredményeket kaptam:

Amint látható a Java a leggyorsabb nyelv míg a C a leglassabb. A C volt a leglassabb, ami annyira nem is meglepő, hiszen ez a nyelv a legelavultabb és mivel ez a legrégebbi, ezért nem meglepetés, az sem, hogy a legkevésbé optimalizáltabb. A Java nyert, hiszen ennek a legjobb a memória menedzselése. Érdekes volt az látni, hogy ahogy növekedett a hatvány értéke, úgy növekedett az űr a teljesítmények között.

	C	C#	Java
10 <sup>6</sup>	4.168	3.788	3.614
10 <sup>7</sup>	43.736	44.873	41.073
10 <sup>8</sup>	509.181	487.927	456.563

## 2.1. táblázat. Összehasonlítás

## 2.4. Ciklomatikus komplexitás

Ciklomatikus komplexitás egy olyan szoftveres "mértékegység" amellyel a programunknak a komplexitását tudunk számmal leírni. Ez a mérés a gráfelmélet alapján történik. Ezt a fogalmat McCabe komplexitásnak is nevezzük. Itt a képlete:

$$M = E - N + 2P$$

M az a szám amely a komplexitást adja meg. E az a szám amely az éleknek a számát adja abból kivonjuk a gráfnak csúcsai és hozzáadjuk az összes komponens dupláját. Én a ennek a dián([https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2\\_2.pdf](https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf)) 78. oldalán lévő programkódcsipetnek a komplexitását mértem. A [www.lizard.ws](http://www.lizard.ws) oldalon néztem meg az eredményét és a következő látható :

Try Lizard in Your Browser

.java
Analyse

```

@Override
protected void jatekbanVezertes(){

    if (latomAKozepet && distaceKozep < 10.0) {
        sarkonFordul();
    } else if (latomASajatGolvonalat && distanceSajatGolvonai < 0.5) {
        sarkonFordul();
    } else if (latomASzelet && distanceSzele < 5.5) {
        palyanMaradni();
    } else if (latomAFocit){
        if (distanceFoci < 0.7) {

```

Code analyzed successfully.

File Type .java
Token Count 190
NLOC 30

Function Name	NLOC	Complexity	Token #	Parameter #
jatekbanVezeries	29	15	186	

Amint láthatjuk a komplexitás ennek a kód részletnek 15. Egy jól megírt, struktúrált kód körülbelül 1 és 10 között van. 15 már egy komplex kódnak számítható, azonban nem olyan komplikált. Ez a kód még jól tesztelhető, azonban ha 20 fölötti az értékünk akkor már egy nagyon komplikált kódról beszélhetünk. 40 fölötti érték már tesztelhetlen és ilyenkor át kell gondolnunk újra a programot hiszen a kódunk túlságosan komplex.



A Visual paradigm szépen lerajzolja, ezt a binfát UML verzióban. Amint látható jól elkülöníthető részeket láthatunk. Vannak Globális függvényeink, de ezek nem sok vizet zavarnak, hiszen nincsenek ágazásaiak.

Két osztályunk van a Csomópont és a LZWBInfa ezeknek vannak tagjai. Amint láthatjuk vannak jeleink (-+ #) ezek a láthatóságot jelöli. A - jel, azt jelenti, hogy a tagunk private a + a public és a # amely a protected jelzi. Csomópont osztályban van egy olyan tag, amely önmagával kapcsolja össze magát. Ezeket az összekapcsolásokat amely osztályok között vannak asszociációnak nevezzük.

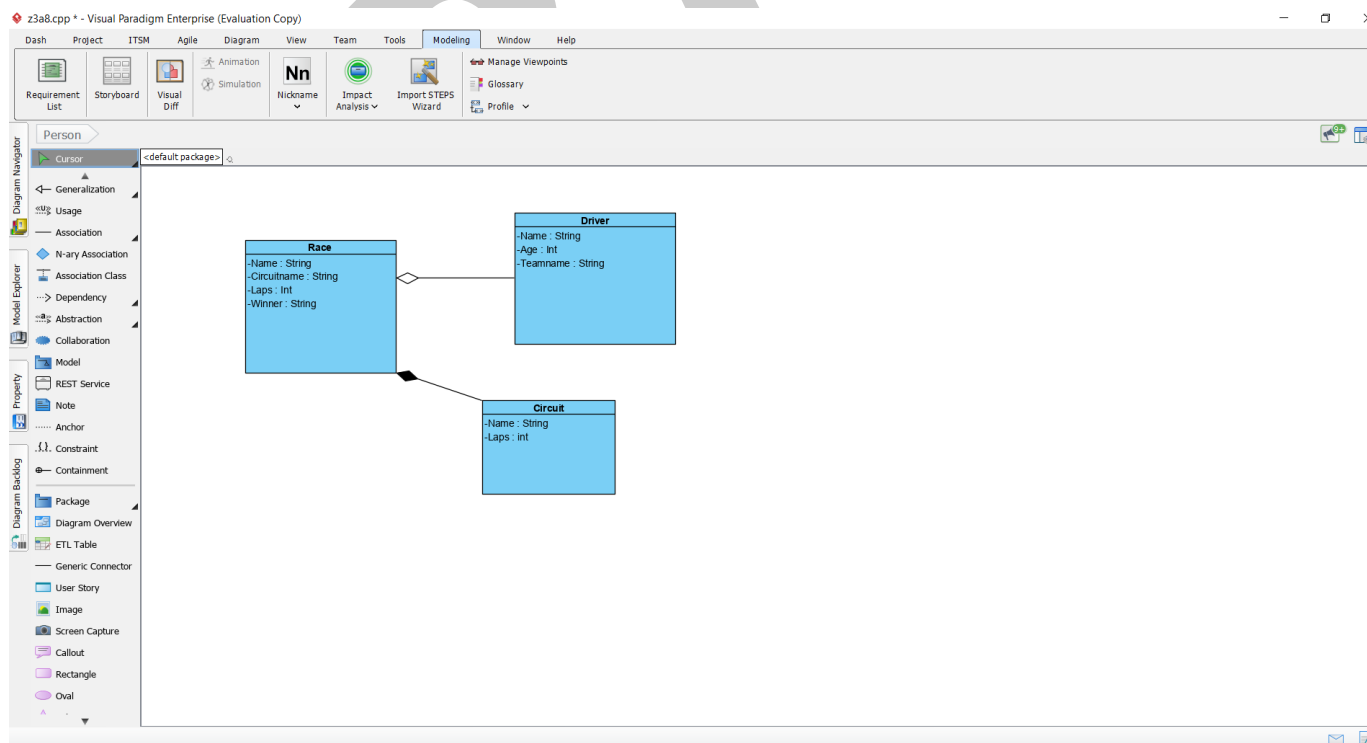
Az asszociációban az osztályok között az osztályok léte független, azonban a osztályok legtöbb esetben legalább egy osztály ismeri a másikat. Az asszociációnak két fajtája van az erős és gyenge aggregáció. A gyenge aggregáció, olyan amikor egy objektum létezhez magában is, de lehet másnak is része. Ennek a jele az UML-ben az üres rombusz.

Az erős aggregációt kompozíciónak is nevezzük, ekkor a részek élettartalma szigorúan megegyezik az egészével. A tartalmazó nem létezhet a tartalmazott nélkül. Ennek jele az UML-ben a jele a telített rombusz.

Bár az én visual paradigmmon ezeket a jelöléseket nem mutatja, mint az erős aggregációt (kompozíciót), viszont ha az egerünket rámutatjuk a kapcsolatra akkor kiírja, hogy asszociáció van a két osztály között. Láthatunk szaggatott vonalas nyilat, amely azt jelenti, hogy dependency (függőség) van az LZWBInfa és a Csomópont osztályok között. Ez azt jelenti, hogy ha valami változás lesz a Csomópont osztályban akkor kihatással lesz az LZWBInfa osztályra is.

## 3.2. Forward engineering UML osztálydiagram

Ebben a feladatban az előző feladatnak az ellentétjét kellett csinálni, itt UML osztályokból kellett programot készíteni. Forward engineering az a lényege, hogy egy program kódot struktúráltan, átláthatóan építsünk fel. Ennek az az előnye, hogyha egy komplikált kódot írunk, akkor gyorsan orvosolhatjuk a problémákat.



Ezen az ábrán van három osztályom, és ebből tudtam kódot generálni a Visual paradigmmeel. Ezzel a programmal és természetesen az UML-s módszerrel nagyon egyszerűen, lehet illusztrálni, hogy hol van

asszociáció, öröklődés a programunkban. Ez azért nagyon praktikus, mert ezzel sokkal gyorsabban lehet változtatni a teljes projekten. Gyakorlatilag egy nyíl "áthúzásával" nagyon sok sornyi programkódot lehet módosítani, átírni. A kisebb projekteken, mint az enyémen, annyira nem hatásos, mert nagyon kevés adatot tartalmaz, de itt is hasznos, mert ellenőrzésnek tökéletesen megfelel.

### 3.3. BPMN

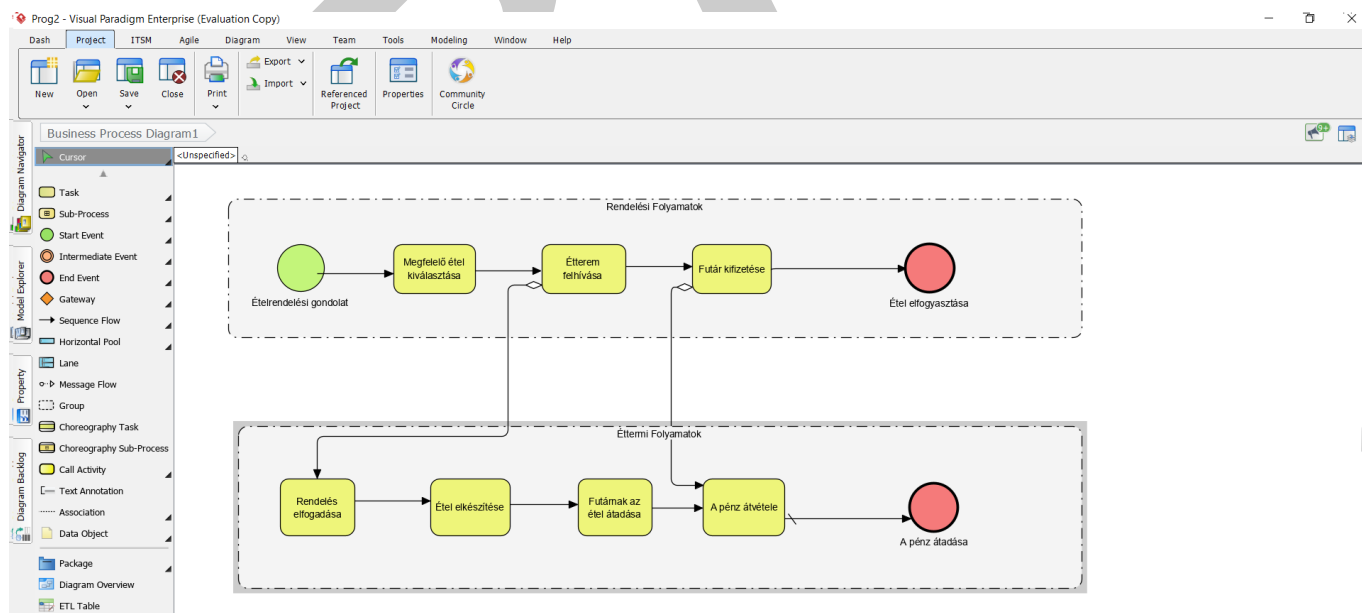
A BPMN (Business Process Modeling Notation) egy olyan folyamatábra, amely az üzleti folyamatok grafikus modellezését szolgálja. Üzleti elemzőknek és technológiai fejlesztőknek szóló grafikus jelölőnyelv.

Az üzleti folyamatok ábrázolására, bemutatására, az EPC és a BPMN módszerek a legelterjedtebbek. Az egyszerű folyamatok ábrázolásától a komplex vállalati folyamatrendszerekig képesek mindent vizualizálni. Ahhoz, hogy a valóságról kapjunk egy képet, amely alapján tudjuk elemezni majd fejleszteni a folyamatainkat elengedhetetlen, hogy azokat grafikailag ábrázoljuk.

Több fajtája is van:

- BPML (XML alapú)
- BPEL (XML alapú)
- XDPL (XML alapú)
- BPMN (Grafikus jelölésre alkalmas)

Ebbe a feladatban rajzolnunk kell egy saját folyamatábrát. Nem igazán alkalmaztam ezelőtt UML-t és ezért volt egy kisebb dilemmám, hogy milyen szoftvert alkalmazzak. Végül a Visual Paradigm-ot választottam, annak is a 30 napos próba verzióját. Egészen egyszerűen lehetett alkalmazni ezt a programot, és elkészítettem a saját folyamatomat. Íme :



Amint a képen látható egy étteremből való rendelését vezettem le. Látható egy a zöld karika, ez a kezdőpontunk. Ez a tevékenység, amely létrehozza, ezt a teljes folyamatot. Láthatunk két nagyobb halmazt, ez a két halmaz, amely a teljes folyamatot képezi. Ez a két halmaz kapcsolatba van egy mással. Láthatjuk, hogy többször is összekapcsoltam a két halmazt. Ez azért van, mert egyes tevékenységet egy adott halmaz nem tehet meg a másik "engedélye nélkül."

Láthatunk a képen sárga téglalapokat, ezek a részfolyamatok. Emelett van két piros karikánk. Ez a két piros karika amely jelzi a folyamatunknak a végét. Mivel mindkét halmaznak van egy adott teljes folyamata, ezért mindkét halmazt le kell zárni.

Természetesen ez a folyamat ábra nagyon egyszerű, a képen látható, hogy lehet belerakni gatewayeket, amelyek arra képesek hogy egy folyamatnak több elágazása legyen. Bár én nem illusztráltam a képemen de lehet olyat is csinálni, hogy beszúrok egy gatewayt az étterem felhívásnál és ha ott adok egy olyan opciót, hogy az étterem zárva, akkor a teljes folyamatnak adhatok egy végpontot.

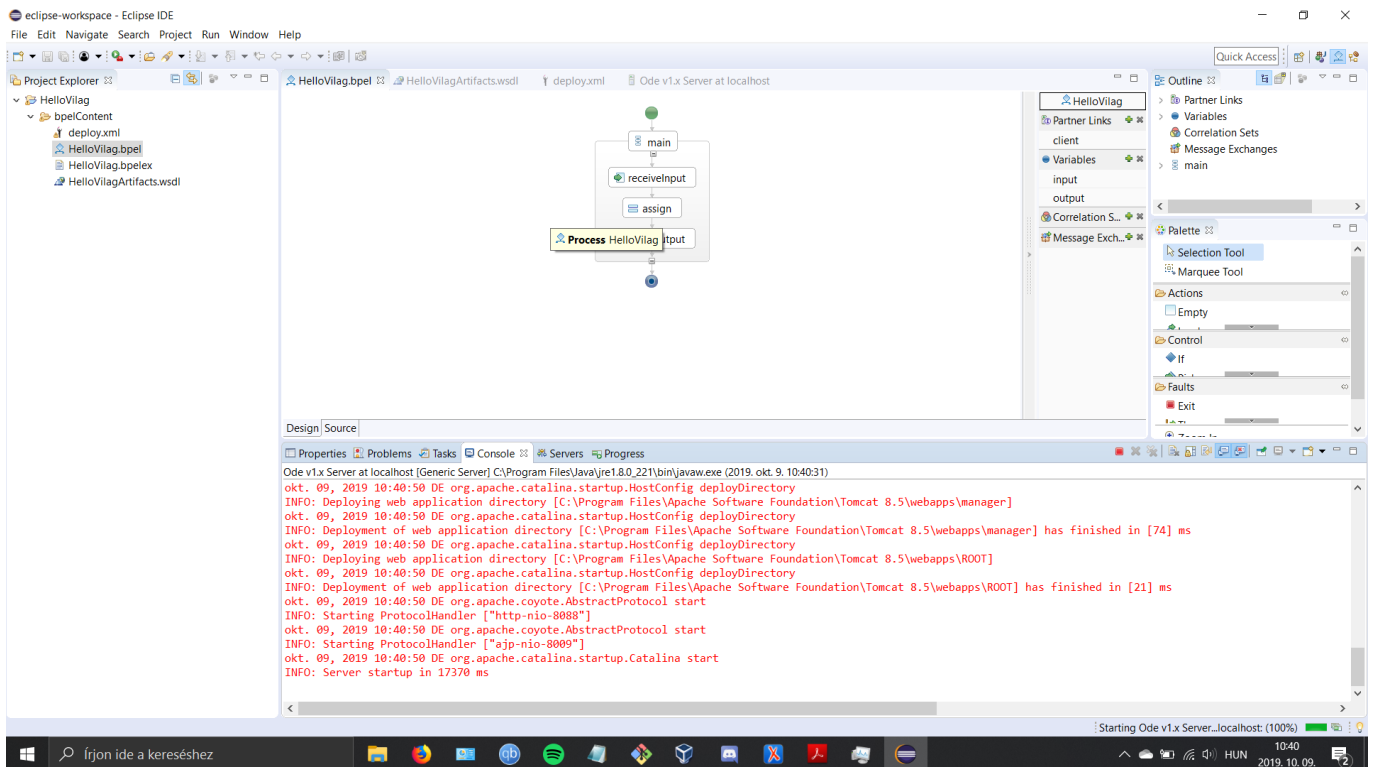
### 3.4. BPEL Helló, Világ! egy visszhang folyamat

BPEL (Business Process Execution Language) üzleti folyamatok modellezésének végrehajtó nyelve. XML alapú folyamatleíró nyílt szabványt alkalmaz. Elsősorban üzleti folyamatok leírására használatos, de egysegessége és elterjedtsége miatt sokszor használják munkafolyamatok leírását igénylő feladatokban is.

Minden egyes BPEL aktivitás egy külső, kiegészítő nyelven elkészített parancs meghívásával jár. A kiegészítő nyelv leggyakrabban Java, de lehet más, magas szintű script-nyelv is. A BPEL nyelvet a Microsoft és az IBM fejlesztette, illetve a korábbi BPML Üzleti folyamatokat modellező nyelvet használták fel.

Ebben a feladatban egy olyan webszervert kellett csinálni amelyben, ha egy string inputot adunk akkor a szervernek az output ugyan az a string legyen. A feladatot a Youtube-os link alapján csináltam, amely a pdf-ben található. Mivel ez a feladat "zöldes" (deprecated) eléggé nehéz volt a megtalálni a megfelelő szoftvereket találni. A legnagyobb problémám azzal adódott, hogy valamiért az Eclipse a szerverindításnál a portjaimat mindig foglaltak titálta. Végül sikerült a szervert elindítani és ezután már csak ki kellett próbálni, hogy működnek a funkciók.

Itt látható ezen a képen, hogy a szerver működik.



Valamilyen bug miatt az Eclipse-nél nem dobja fel a webservices opciót amivel tudnám a szimulálni a programomat. Ezt a hibát szeretném kijavítani később.

## 4. fejezet

# Helló Chomsky!

### 4.1. Encoding

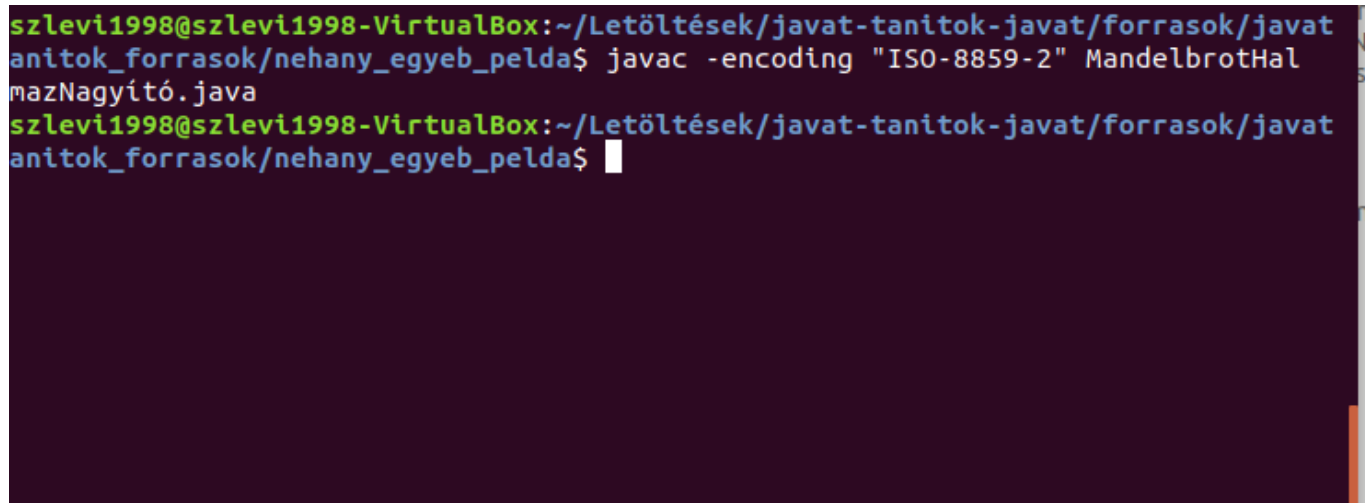
Ezt a feladatot az órán elkezdtek, azzal a problémával találkozunk, hogy a mandelbrotnagyító nem fordul le, az ékezetes betűkkel. Először is a problémánk az, hogy ne essünk abba a hibába mint én. Ami a PDF linkjében linkelt feladat önmagában sose fordul le, hiszen ez egy teljes projekt, nem csak egy programfájl. Amint fordítjuk a programunk még több hibával fog fordulni, de ez nem baj hiszen a hibák most már egyértelműek, ugyanis itt a java nem képes a magyar ABC szavait kezelni.

```
^
MandelbrothHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
UTF-8
    // vizsgáljuk egy adott pont iterációt:
    ^
MandelbrothHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
UTF-8
    // vizsgáljuk egy adott pont iterációt:
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
UTF-8
    // Az egórmutató pozíciója
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
UTF-8
    // Az egórmutató pozíciója
    ^
MandelbrothHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
UTF-8
    // Az egórmutató pozíciója
    ^
100 errors
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/javat-tanitok-javat/forrasok/javat
anitok_forrasok/nehany_egyeb_pelda$
```

Amint láthatjuk a képen, hogy az ékezetes betűk nem mappelhető karakterek. Ezek után utána kellett nézni, hogy milyen karakterkódolást kell alkalmaznunk, ahhoz hogy ezek a karakterek alkalmazhatóak legyenek. Én gyors keresés után a Java doksiknál megtaláltam azt, hogy magyar karakterek alkalmazásához encoding

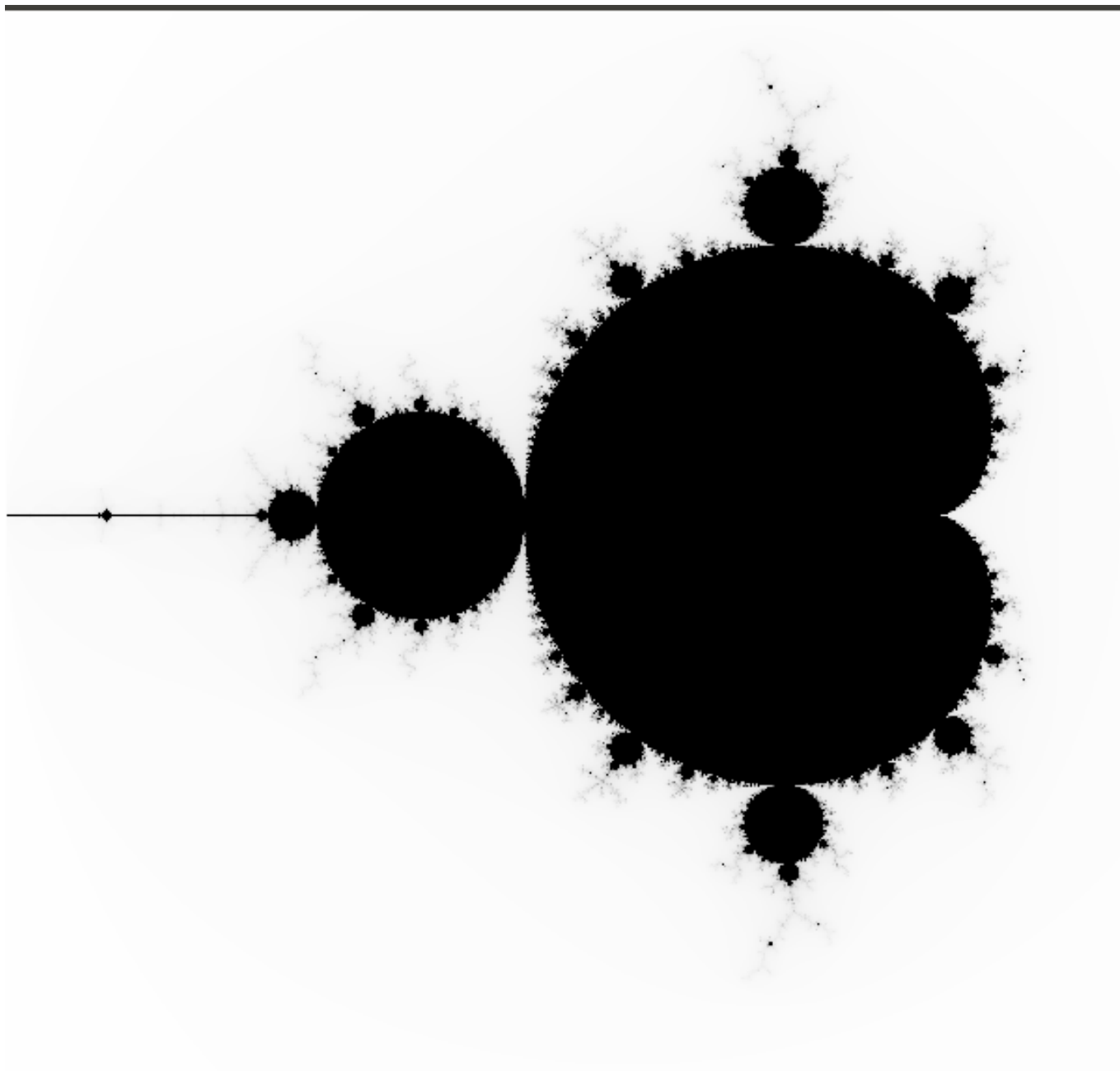


kapcsolót kell alkalmaznunk. A magyar karakterekhez a Latin 2-t kellett alkalmazni és ahhoz, hogy ezt működtessük, a fordításnál ezt a kapcsolót kell alkalmaznunk, az alábbi kép alapján.

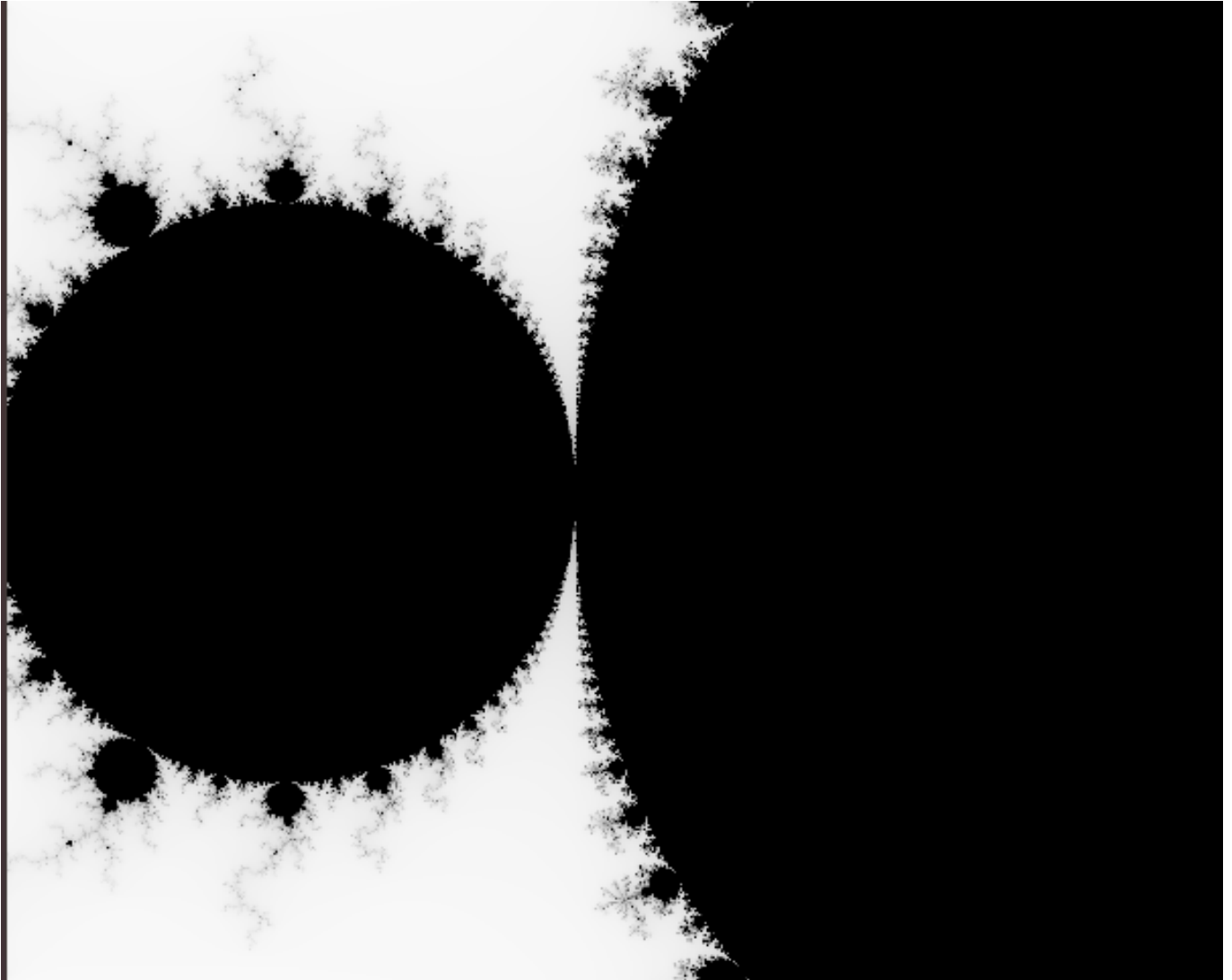


```
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/javat-tanitok-javat/forrasok/javat  
anitok_forrasok/nehany_egyeb_pelda$ javac -encoding "ISO-8859-2" MandelbrothHal  
mazNagyító.java  
szlevi1998@szlevi1998-VirtualBox:~/Letöltések/javat-tanitok-javat/forrasok/javat  
anitok_forrasok/nehany_egyeb_pelda$
```

Itt a kép a fordításról.



Itt a kép a programról.



Itt a kép a futásról és a nagyításról.

Amint láthatjuk, a program ezek után gond nélkül fut és fordul. A képen a kedvünkre tudunk nagyítani, ott ahol akarunk.

## 4.2. Full screen

Ebben a feladatban az volt a lényeg, hogy egy olyan Java kódot írjunk amelyben, teljes képernyőt alkalmazunk.

```
import javax.swing.*;  
  
import java.awt.*;  
import java.awt.event.MouseAdapter;  
import java.awt.event.MouseEvent;
```

```
public class Fullscreen {

    public static void main(String[] args) {

        JButton clickme = new JButton("Click me!");
        clickme.setBounds(180, 100, 200, 50);
        clickme.setFont(new Font("Times New Roman", Font.PLAIN, 11));

        JButton click = new JButton("Don't click me!");
        click.setFont(new Font("Times New Roman", Font.BOLD, 15));
        click.setBounds(400, 100, 250, 50);

        click.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                System.exit(0);
            }
        });

        JPanel panel = new JPanel();
        panel.setBackground(new Color(0, 191, 255));
        panel.setBounds(200, 200, 900, 300);

        panel.add(click);
        panel.add(clickme);
        panel.setLayout(null);

        JFrame frame = new JFrame("Fullscreen");

        frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setUndecorated(false);
        frame.setVisible(true);
        frame.getContentPane().setLayout(null);
        frame.getContentPane().setBackground(new Color(255, 255, 0));
        frame.getContentPane().add(panel);

    }

}
```

Lássuk is a kódunkat.

```
        JButton clickme = new JButton("Click me!");
        clickme.setBounds(180, 100, 200, 50);
```

```
clickme.setFont(new Font("Times New Roman", Font.PLAIN, 11));
```

Ebben a részben létrehozok egy Java Butont amely clickmenek nevezek el. "" részben adtam meg, hogy a gombra mi legyen ráírva. Aztán a setBoundssal megadom a gombnak a helyzetét (x,y tengelyen) illetve a magasságát és a szélességét. Emellett változtatok a gomb betű típusán a méretét és a stílusát itt például dőltre.

```
JButton click = new JButton("Don't click me!");
click.setFont(new Font("Times New Roman", Font.BOLD, 15));
click.setBounds(400, 100, 250, 50);

click.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.exit(0);
    }
});
```

Itt létrehozok még egy gombot ez nagyon hasonlít a másik gombra,annyi különbséggel, hogy mássabb a betűstílus mérete és természetesen amit kiírok. MouseListener az egy olyan listener amely arra figyel, hogy ha rákattintok akkor az alkalmazást bezárja.

```
JPanel panel = new JPanel();
panel.setBackground(new Color(0, 191, 255));
panel.setBounds(200, 200, 900, 300);

panel.add(click);
panel.add(clickme);
panel.setLayout(null);
```

Itt létrehozok egy panelt amelynek a setboundssal adok méretet és pozíciót. Illetve a setBackgrounddal ennek a tálcának adok egy háttérszínt. A tálcához hozzáadom a gombokat a panel.add metódussal.

```
JFrame frame = new JFrame("Fullscreen");

frame.setExtendedState(JFrame.MAXIMIZED_BOTH);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setUndecorated(false);
frame.setVisible(true);
frame.getContentPane().setLayout(null);
frame.getContentPane().setBackground(new Color(255, 255, 0));
frame.getContentPane().add(panel);
```

```
}  
  
}
```

Ablakot itt hozzuk létre, a JFrame-mel és "Fullscreen" a cím viszont ezt nem láthatjuk, ugyanis a `frame.setUndecorated()` zároljuk, és ezzel nem látjuk a címsort. Azonban ezzel a módszerrel, még nem változik teljes képernyőssé a programunk. Ezt `frame.setExtendedState(JFrame.MAXIMIZED_BOTH)`; ezzel a módszerrel maximalizáljuk a szélességét és a magasságát az ablaknak.

Emelet még ahhoz, hogy ablak látható legyen a `frame.setVisible(true)`-t alkalmazzuk. Abban az esetben, ha `false` az értékünk, akkor nem láthatjuk az ablakunkat. Állítottam egy háttérszínt az ablaknak `frame.getContentPane().setBackground(new Color(255, 255, 0))`; -al és ezek után legutoljára, hozzáadtam a ablakhoz a tálcánkat, így válik teljessé a programunk.



### 4.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Ebben a feladatban egy OpenGL-es projektben kellett kisebb változtatásokat alkalmazni. Először is rengeteg dolgot felrakni, ahhoz hogy a programkódunk fusson és forduljon. Szükségünk volt a libboost-ra, az OpenGL-re és emelet számos update-ra. Ezek a szükséges szoftverek, könyvtárak után a fordítás és a futás sikeresen működött. Szokatlan volt az, hogy az irányítás ellentett volt. Először is rákerestem hogy, hol van a feladatban az irányítás része. Az adott programcsipetben látható a forgatásrésze.

```
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();
}
```

Az eredeti kódban az értékek előtti műveleti jelek megvoltak cserélve, ezért ezeket átírtam az ellentetjére és ezek után kézre álló volt. Természetesen, ha az értékeken változtatunk akkor a forgatás mértéke is növekszik vagy csökken.

Emellett, az is feladatunk volt, hogy a változtassunk egy kicsit a színvilágon. Nem volt nehéz dolgunk hiszen a programunk nagyon sokszor alkalmazza az OpenGL-ben használt glColor3f függvényt. Ebben a függvényben 3 darab argumentum van és ezek az argumentumok adják meg a színünket. Az értékek az rgb-hez színvilágnak felel meg, hiszen a 3 érték a pirosnak a zöldnek és a kéknek felel meg. A számítása a következő : a választott értékeinket osztani kell 255-el és ezt a törtszámot kell nekünk megadni pl: aranyárgának az értékei :

```
glColor3f ( .960f, .772f, .0031f );
```

Ebbe a kis programcsipetben található a "sima" fehérén hagyott négyzeteket.

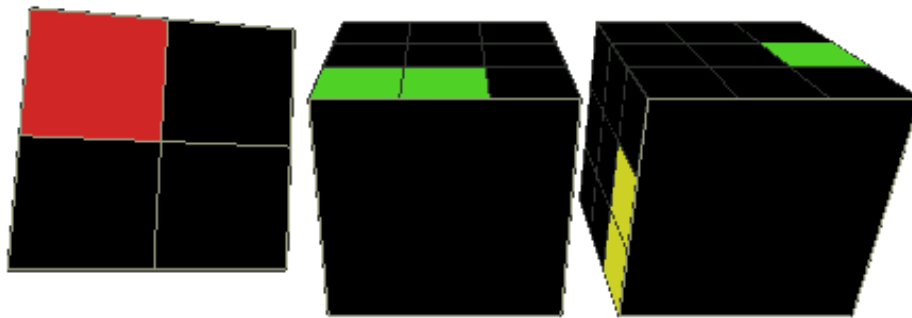
```
void drawPaRaCube ( int idx )
{
    glPushMatrix();

    int d = cubeLetters.size() /2 ;
    glTranslatef ( ( idx-d ) *2.5f, 0.0f, 0.0f );

    glRotatef ( cubeLetters[idx].rotx, 1.0f, 0.0f, 0.0f );
    glRotatef ( cubeLetters[idx].roty, 0.0f, 1.0f, 0.0f );
    glRotatef ( cubeLetters[idx].rotz, 0.0f, 0.0f, 1.0f );
}
```

```
glBegin ( GL_QUADS );  
  
glColor3f ( .0001f, .0001f, .0001f );
```

Itt látható hogy a glColorba átállítottam, 0-ra és ezek után az eredetileg fehér négyzetek feketére változtak. A változások így szembetűnőek. Íme a kép:





## 5. fejezet

# Helló, Stroustrup

### 5.1. JDK osztályok

Ebben a feladatban,azzal kellett foglalkoznunk, hogy egy olyan c++-os kódot kellett írunk,hogy a Java JDK osztályainak a fájlait kellett kiírtani. Ahhoz ,hogy ezt elérjük, először is szükségünk van egy Java JDK-ra amelyből ki tudjuk listázni a fájlokat. Ezután szükségünk van a libboostos könyvtárakra is.

```
sudo apt-get install libboost
```

Illetve a JDK osztályt lehetett volna tölteni az Oracle oldalára,de én nem akartam Oracle fiókot létrehozni, ezért innen töltöttem le : <https://bell-sw.com/pages/java-8u232/>

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

using namespace std;
using namespace boost::filesystem;

int main(int argc, char *argv[])
{
    path p ("src");

    if(!exists(p) || !is_directory(p)) {
        cout << p << "is not a path" << endl;

        return 1;
    }

    int fajlok = 0;

    recursive_directory_iterator begin(p), end;
```

```
vector <directory_entry> v(begin, end);
for (auto& f:v){
    if(path(f).has_extension()){
        cout << "File: " << path(f).filename() << endl;
        fajlok++;
    } else {
        cout << f << endl;
    }

}
cout << " A fájlok száma: " << fajlok << endl;
}
```

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

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

Lássuk is a kódot. Amint látható szükségem volt, a libboost osztályokra ezért includeolni kellett.

```
int main(int argc, char *argv[])
{

    path p ("src");

    if(!exists(p) || !is_directory(p)) {
        cout << p << " is wrong folder " << endl;

        return 1;

    }

    int fajlok = 0;
```

Ebben a részben a (mainben) először is megadjuk, hogy melyik mappát vizsgáljuk. Az ifben pedig azt vizsgáljuk, hogy megfelelő mappát vizsgáljuk. Abban az esetben ha nem jó mappát vizsgáljuk akkor, mint a kódban is látható kiadja a program, hogy nem jó mappát vizsgálunk.

A terminal window with a dark background and light-colored text. The window title is 'szlevi1998@Lenovo-Y520: ~/Letöltések'. The prompt is 'szlevi1998@Lenovo-Y520:~/Letöltések\$'. The first command is 'g++ lib.cpp -o lib -std=c++11 -lboost\_system -lboost\_filesystem'. The second command is './lib'. The output is '"src" is wrong folder'. The prompt is now 'szlevi1998@Lenovo-Y520:~/Letöltések\$' with a cursor.

```
szlevi1998@Lenovo-Y520: ~/Letöltések
szlevi1998@Lenovo-Y520:~/Letöltések$ g++ lib.cpp -o lib -std=c++11 -lboost_system -lboost_filesystem
szlevi1998@Lenovo-Y520:~/Letöltések$ ./lib
"src" is wrong folder
szlevi1998@Lenovo-Y520:~/Letöltések$
```

A képen is látható, hogy ha nem megfelelő path-t adok neki. Valamint adtam egy fájlok nevű változót amely arra szolgál, hogy majd később a fájlok számát ebbe a változóba tudjuk majd tárolni.

```
        recursive_directory_iterator begin(p), end;
vector <directory_entry> v(begin, end);
for (auto& f:v){
    if(path(f).has_extension()){
        cout << "File: " << path(f).filename() << endl;
        fajlok++;
    } else {
        cout << f << endl;
    }
}
cout << " A fájlok száma: " << fajlok << endl;
}
```

Ez a programnak az "agya". Ebben a részben használjuk azokat a libboost-os osztályokat amelyek miatt kellett includeolni a programunkba. A directory iterátorral végigjárjuk az összes directory entryt és amint. Ezeket a bejárásokat egy vectorba rögzítjük. Majd a for ciklusban az íffel megvizsgáljuk, hogy az adott fajlnak van-e utótagja, ha van akkor "File :" előtagot adok neki ezért megtudjuk különböztetni a fájlokat a mappáktól. Addig amíg a for ciklus fut, azaz addig amíg a bejárások megtörténnek addig növeljük a fajlok értékét. A fajlok növelése addig fut ameddig le nem fut a bejárás.

Ezután kiíratjuk, hogy hány darab fájlt találtunk.

```
szlevi1998@Lenovo-Y520: ~/Letöltések/5.óra
szlevi1998@Lenovo-Y520:~/Letöltések/5.óra$ g++ lib.cpp -o lib -std=c++11 -lboost
_system -lboost_filesystem
szlevi1998@Lenovo-Y520:~/Letöltések/5.óra$
```

```
szlevi1998@Lenovo-Y520: ~/Letöltések/5.óra
File: "launcher_ko.java"
File: "lib.cpp"
"src/launcher"
File: "jli_util.h"
File: "java.c"
File: "java_md.h"
File: "wildcard.h"
File: "version_comp.h"
File: "emessages.h"
File: "jli_util.c"
File: "manifest_info.h"
File: "defines.h"
File: "java_md_solinux.c"
File: "main.c"
File: "parse_manifest.c"
File: "java_md_common.c"
File: "java.h"
File: "splashscreen.h"
File: "splashscreen_stubs.c"
File: "version_comp.c"
File: "java_md_solinux.h"
File: "wildcard.c"
A fájlok száma: 17427
szlevi1998@Lenovo-Y520:~/Letöltések/5.óra$
```

## 5.2. Másoló-mozgató szemantika

Ebben a feladatban példákat kellett mutatni, olyan másoló-mozgató szemantikákra, amelyek a c++11-ben lettek elérhetőek. A Z3a9.cpp (ami tudatom szerint a legfrissebb binfa amit lehet találni).

Másoló és mozgató szemantika között elsődlegesen azon van a hangsúly, hogy jobbérték referencia kerül-e átadásra vagy balérték, mert jobbérték esetén a mozgató szemantika érvényesül, minden más esetben pedig a másoló konstruktor hívódik meg.

A másoló szemantika úgy inicializál egy objektumot, hogy egy másik objektumot használ fel a vele azonos osztályból. A dinamikusan lefoglalt változóknál nem elég a mutatókat másolni, új területet kell foglalni, és átmásolni a változó értékét. Az ilyen másolat készítését hívják mély másolásnak (deep copy), ellentétben azzal, amikor csak a mutatókat másoljuk.

```
LZWBinFa ( const LZWBinFa & regi ) {  
    std::cout << "LZWBinFa copy ctor" << std::endl;  
  
    gyoker.ujEgyesGyermeke ( masol ( regi.gyoker.egyesGyermeke ( ), regi ←  
        .fa ) );  
    gyoker.ujNullasGyermeke ( masol ( regi.gyoker.nullasGyermeke ( ), ←  
        regi.fas ) );  
  
    if ( regi.fas == & ( regi.gyoker ) )  
        fas = &gyoker;
```

Mint ahogy az outputban is látható, ez a binfa másoló konstruktor. Itt az történik, hogy az új gyökérnek az egyes és nullás gyermekének átadjuk a régi gyökeret.

```
LZWBinFa ( LZWBinFa && regi ) {  
    std::cout << "LZWBinFa move ctor" << std::endl;  
  
    gyoker.ujEgyesGyermeke ( regi.gyoker.egyesGyermeke ( ) );  
    gyoker.ujNullasGyermeke ( regi.gyoker.nullasGyermeke ( ) );  
  
    regi.gyoker.ujEgyesGyermeke ( nullptr );  
    regi.gyoker.ujNullasGyermeke ( nullptr );  
  
}
```

Ez pedig a mozgatókonstruktor a binfának. Itt az a lényeg, hogy a binfának a régi fáját mozgadjuk egy másik memóriacímre míg a régit töröljük. A régi gyökérnek az egyes és nullás gyerekeit is nullpointerrel tesszük egyenlővé, azaz lenullázzuk.

## 5.3. Változó argumentum ctor

Ebben a feladatban az volt a lényeg, hogy a prog 1-es példában is használt Perceptron osztály ne egy értéket adjon vissza, hanem egy ugyanakkora méretű képet kell visszatértenie.

Mielőtt a feladatra térnénk, fontos kiemelni, hogy a programhoz szükséges több fontos könyvtár ami lehetséges hogy nincs meg egyből. Ilyen például a libpng amely feltétlenül szükséges és fontos, bár nem tudom, hogy csak nálam okozott problémát, de szükséges a linkgrammar is.

Ennek a projektben 2 fontos fájlunk van és mind a két fájlban, kellett változtatásokat eszközölni. Először is foglalkozunk a main fájlunkkal.

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

using namespace std;

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, 1);

    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* kep = (*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].red = kep[i*png_image.get_width()+j];

    png_image.write("output.png");

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

Ez a main fájlunk. Ebben kellett egy kisebb változást alkalmazni.

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

Ebben a részben hozzáadjuk a fájlhoz a szükséges könyvtárakat. A png++ könyvtár teszi lehetővé, hogy képállománnyal tudjuk dolgozni.

```
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, 1);

    double* image = new double[size];
```

Itt van a main függvényünk. Az első sorban meghatározzuk, hogy a képállományunk az parancsori argumentum legyen. Ezután létrehozuk a size int típusú változót. Ez a változó lesz, amellyel kiszámoltatjuk a képnek a méretét és ebbe fogjuk tárolni az adott méretet. Ezután példánosítjuk a Perceptron osztályt. Majd létrehozunk egy double mutatót.

```
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;
```

Itt ezután létrehozunk két for ciklust amelynek, a funkciója az, hogy a 2 ciklus egyenként átmennek a kép magasságán, illetve a kép szélességén és ezt majd később az imagebe fogja tárolni.

```
double* kep = (*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].red = kep[i*png_image.get_width()+j];

png_image.write("output.png");

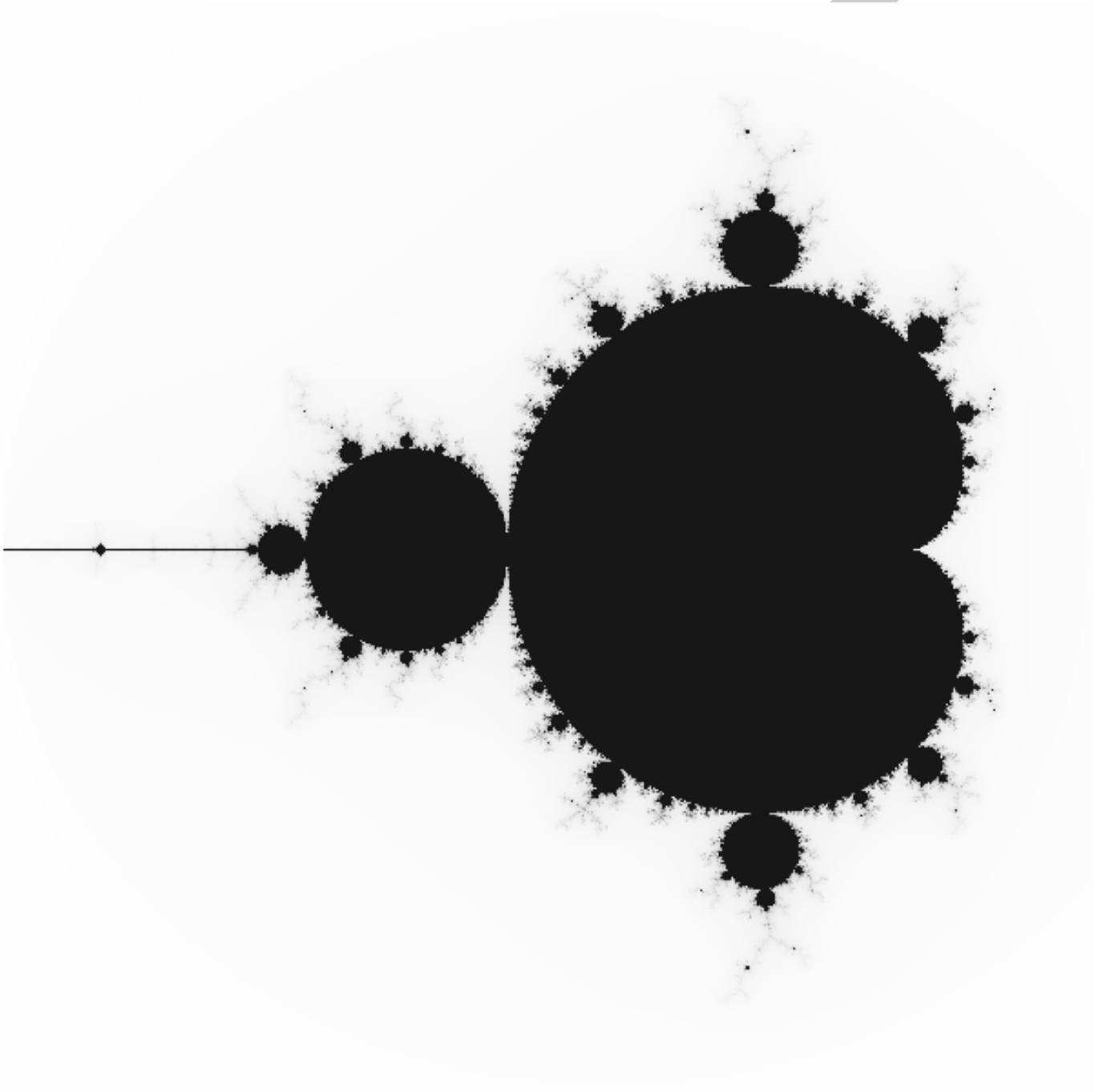
delete p;
delete [] image;
}
```

Itt változtatunk igazán a programon, hiszen az image-al már nem értéket fog visszaadni, hanem most már egy képet. Itt is double csillag típusra van szükség. Itt ismét 2 for ciklus kell amely átmegy az eredeti képnek szélességén és magasságán és ezeket az értékeket majd a kep be tároljuk. Ezután alkalmazzuk a write függvényt és ezzel a tárolt adatok alapján egy új output.png-t létrehozunk. Ezzel még nem vagyunk

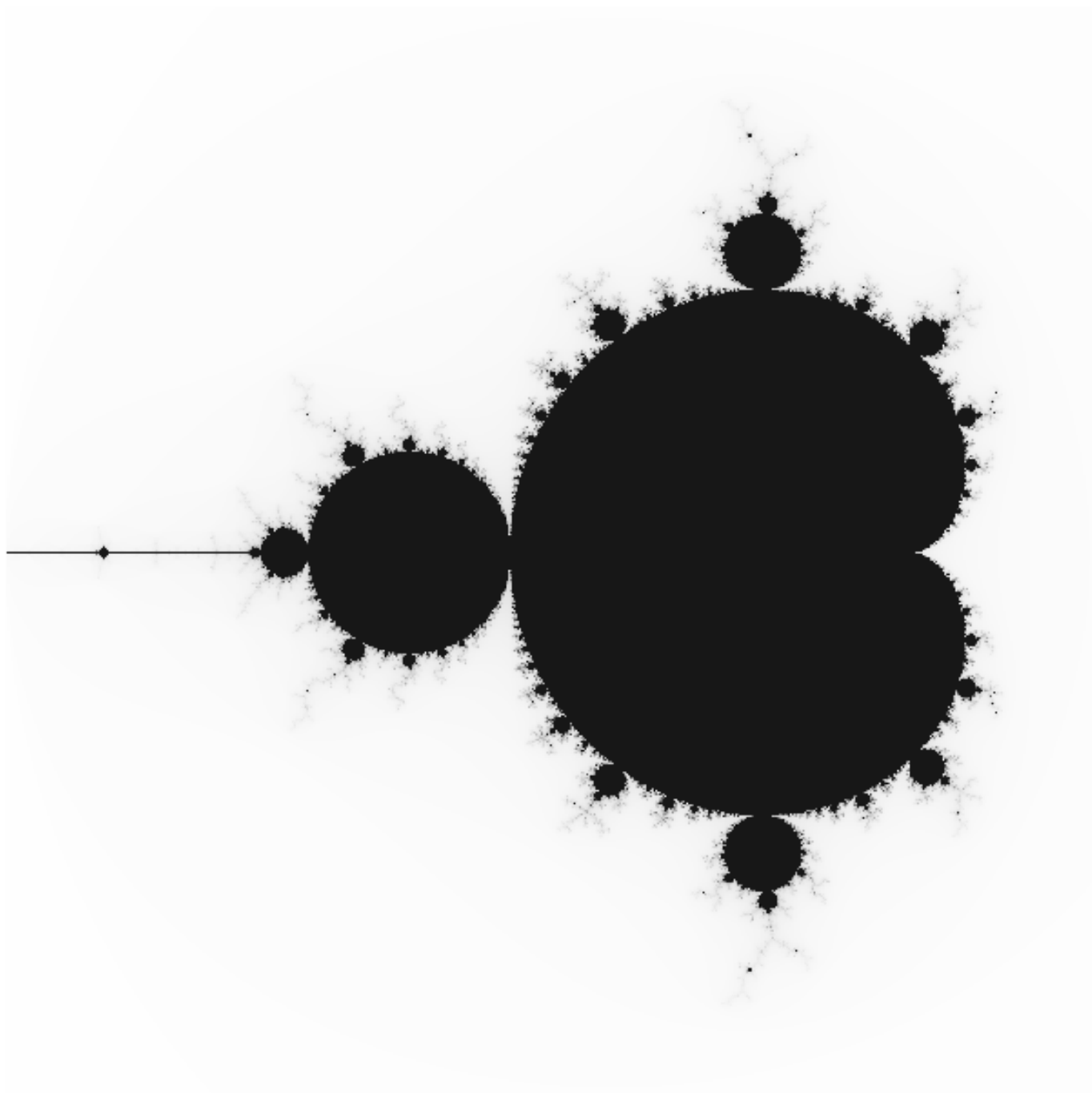
készek, hiszen az mlp fájlban át kell írni az operator függvénynek a visszatérítési értékét, ugyan is már nem double-t kell visszatérítenie, hanem egy double mutatót kell.

```
// double operator() ( double image [] )  
double* operator() ( double image [] )
```

Így kellett változtani az mlp.hpp-n és ezek után már gond nélkül működik a programunk. Íme a kép a programunkról:







Amint látható a másolás tökéletesen működött, a mandel és az output teljesen megegyezik.

## **II. rész**

### **Irodalomjegyzék**

## 5.4. Általános

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

## 5.5. C

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

## 5.6. C++

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

## 5.7. Lisp

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

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

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