Univerzális programozás

Írd meg a saját programozás tankönyvedet!

DRAFT

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

PDF BY DBLATEX

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

DRAFT

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert Ã©s Tóth, László Szilárd	2020. december 6.	

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

DRAFT

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]

DRAFT

Tartalomjegyzék

I. Bevezetés **1**

1. Vízió **2**
1.1. Mi a programozás? . . . 2
1.2. Milyen doksikat olvassak el? . . . 2
1.3. Milyen filmeket nézzek meg? . . . 2

II. Tematikus feladatok **3**

2. Helló, Turing! **5**
2.1. Végtelen ciklus . . . 5
2.2. Lefagyott, nem fagyott, akkor most mi van? . . . 5
2.3. Változók értékének felcserélése . . . 7
2.4. Labdapattogás . . . 7
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 7
2.6. Helló, Google! . . . 7
2.7. 100 éves a Brun tétel . . . 8
2.8. A Monty Hall probléma . . . 8

3. Helló, Chomsky! **9**
3.1. Decimálisból unárisba átváltó Turing gép . . . 9
3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 9
3.3. Hivatkozási nyelv . . . 9
3.4. Saját lexikális elemző . . . 10
3.5. l33t.l . . . 10
3.6. A források olvasása . . . 10
3.7. Logikus . . . 11
3.8. Deklaráció . . . 11

DRAFT

4. Helló, Caesar! 13
4.1. int *** háromszögmátrix . . . 13
4.2. C EXOR titkosító . . . 13
4.3. Java EXOR titkosító . . . 13
4.4. C EXOR törő . . . 13
4.5. Neurális OR, AND és EXOR kapu . . . 14
4.6. Hiba-visszaterjesztéses perceptron . . . 14

5. Helló, Mandelbrot! 15
5.1. A Mandelbrot halmaz . . . 15
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 15
5.3. Biomorfok . . . 15
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 15
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 15
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 16

6. Helló, Welch! 17
6.1. Első osztályom . . . 17
6.2. LZW . . . 17
6.3. Fabejárás . . . 17
6.4. Tag a gyökér . . . 17
6.5. Mutató a gyökér . . . 18
6.6. Mozgató szemantika . . . 18

7. Helló, Conway! 19
7.1. Hangyaszimulációk . . . 19
7.2. Java életjáték . . . 19
7.3. Qt C++ életjáték . . . 19
7.4. BrainB Benchmark . . . 20

8. Helló, Schwarzenegger! 21
8.1. Szoftmax Py MNIST . . . 21
8.2. Szoftmax R MNIST . . . 21
8.3. Mély MNIST . . . 21
8.4. Deep dream . . . 21
8.5. Robotpszichológia . . . 22

DRAFT

9. Helló, Chaitin! 23
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 23
9.2. Weizenbaum Eliza programja . . . 23
9.3. Gimp Scheme Script-fu: króm effekt . . . 23
9.4. Gimp Scheme Script-fu: név mandala . . . 23
9.5. Lambda . . . 24
9.6. Omega . . . 24

III. Második felvonás 25

10. Helló, Arroway! 27
10.1. OO szemlélet . . . 27
10.2. „Gagyi” . . . 27
10.3. Yoda . . . 28
10.4. Homokózó . . . 28

11. Helló, Liskov! 29
11.1. Anti OO . . . 29
11.2. Szülő-gyerek . . . 30
11.3. Liskov helyettesítés sértése . . . 30
11.4. EPAM: Interfész evolúció Java ban . . . 31

12. Helló, Mandelbrot! 33
12.1. Reverse engineering UML osztálydiagram . . . 33
12.2. Forward engineering UML osztálydiagram . . . 34
12.3. Neptun tantárgyfelvétel modellezése UML ben . . . 38
12.4. Neptun tantárgyfelvétel UML diagram implementálása . . . 39

13. Helló, Chomsky! 40
13.1. Encoding . . . 40
13.2. l334d1c4 . . . 40
13.3. Perceptron osztály . . . 41
13.4. Full screen . . . 42

DRAFT

14. Helló, Stroustrup! **43**
14.1. JDK osztályok . . . 43
14.2. Hibásan implementált R SA törése . . . 45
14.3. Változó argumentumszámú ctor és Összefoglaló . . . 48

15. Helló, Gödel! **52**
15.1. Gengszterek . . . 52
15.2. STL map érték szerinti rendezése . . . 53
15.3. Alternatív Tabella rendezése . . . 55
15.4. GIMP Scheme hack . . . 56

16. Helló, ! **61**
16.1. FUTURE tevékenység editor . . . 61
16.2. OOCWC Boost ASIO hálózatkezelése . . . 62
16.3. OOCWC Boost ASIO hálózatkezelése . . . 63
16.4. SamuCam . . . 64
16.5. BrainB . . . 67

17. Helló, Lauda! **72**
17.1. Junit teszt . . . 72
17.2. Android Játék . . . 73
17.3. Port scan . . . 77
17.4. EPAM: Kivételkezelés . . . 77

18. Helló, Calvin! **79**
18.1. MNIST . . . 79
18.2. CIFAR-10 -deprecated . . . 82
18.3. EPAM: Back To The Future . . . 85

IV. Irodalomjegyzék **89**
18.4. Általános . . . 90
18.5. C . . . 90
18.6. C++ . . . 90
18.7. Lisp . . . 90

DRAFT

Ábrák jegyzéke

16.1. Az átalakított program 62

17.1. JUnit teszt 73

DRAFT

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

DRAFT

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

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

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

A DocBook XML 5.1 új neked?
Ez esetben forgasd a https://tdg.docbook.org/tdg/5.1/ könyvet, a végén találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

DRAFT

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

DRAFT

II. rész

Tematikus feladatok

DRAFT

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

Megoldás videó:

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

```
Program T100
{

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

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

DRAFT

```
  }
}
```

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

```
T100(t.c.pseudo)
true
```

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

Mit for kiírni erre a `T1000(T1000)` futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.4. Labdapattogás

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

DRAFT

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

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

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

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

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

i.
```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezelo);
```

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

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

DRAFT

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

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

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

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

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX

Megoldás videó: https://youtu.be/ZexiPy3ZxsA, https://youtu.be/AJSXOQFF_wk

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája

DRAFT

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

DRAFT

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

DRAFT

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

DRAFT

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... térj ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked!

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

DRAFT

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

DRAFT

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

DRAFT

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

DRAFT

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

DRAFT

III. rész

Második felvonás

DRAFT

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

10. fejezet

Helló, Arroway!

10.1. OO szemlélet

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Arroway!/-oo

Az algoritmus elkészítése "public" "PolárGenerátor" osztályban történik. Deklarálunk egy "nincsTárolt" változót is, amely "boolean" típusú. Deklarálunk egy "tárolt" változót is, amely "double" típusú. Deklarálunk egy "következő" "double" típusú fgv-t is. Amennyiben a "nincsTárolt" igaz, akkor egy do-while cikluson belül a "Math.random+ függvényt használva "u1" illetve "u2" egy véletlen értéket kap, addig ameddig "w" nagyobb mint 1. "v1" illetve "v2" megkapja az előző változók értékeit, ahogy "w" is. Utána kiszámoljuk az "r" értékét is, ahol már a "tárolt" változónk is értéket kap. Ha hamis, akkor a fgv a már eltárolt elemet adja vissza.

10.2. „Gagyi”

Az ismert formális2„while x <= t && x >= t && t != x);” tesztkérdéstípusraadj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Arroway!/-Gagyi

DRAFT

10.3. Yoda

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Arroway!/-Yoda

10.4. Homokózó

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Arroway!/-LZWBinFa

DRAFT

11. fejezet

Helló, Liskov!

11.1. Anti OO

A BBP algoritmussal a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10^6, 10^7, 10^8darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! https://www.tankonyvtar.hu/hu/tarta tanitok-javat/apas03.html#id561066

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Liskov!/Anti%20C

fordítás/futtatás:

C:

g++ pi_bbp_bench.c -o pi_bbp_bench -lm

./pi_bbp_bench

C++

g++ BBP_test.cpp -o bbpTest

./bbpTest

Java

javac PiBBPBench.java

java PiBBPBench

C#

mcs PiBBPBench.cs

mono PiBBPBench.exe

C:

Kapott eredmények:

C:

10^6 =2,111283

10^7 =24,401839

DRAFT

10^8 =282,93285

C++:

10^6 =2,44803

10^7 =27,7696

10^8 =313,902

Java:

10^6 =1,883

10^7 =22,144

10^8 =248,324

C#:

10^6 =1,922333

10^7 =22,595576

10^8 =256,861381

11.2. Szülő-gyerek

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Liskov!/Sz%C3%
gyerek

11.3. Liskov helyettesítés sértése

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Liskov!/Liskov%2

A Liskov-féle behelyettesítési elv, rövid nevén LSP a következőt mondja ki:

Ha S osztály a T osztály leszármazottja, akkor S szabadon behelyettesíthető a programunkban minden olyan helyre (őaraméter,változó...), ahol eredetileg T szülő típust várnánk

fordítás/futtatás:

C++

g++ Liskovsertes.cpp -o Liskovsertes

DRAFT

./Liskovsertes

Java

javac Liskovsertes.java

java Liskovsertes

Mind a kettő esetben azt kapjuk vissza, hogy az Ebr tud láncot javítani, ami nem lehetséges mert az ebr egy kerekes tank.

11.4. EPAM: Interfész evolúció Java ban

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

Megoldás videó:

A Java 7 2011-ben jelent meg.

Ez volt az első nagyobb Java-frissítés, miután az Oracle megszerezte a Sun Microsystems-t.

A Java 7 a végső Java-verzió, amely hivatalosan támogatná a Windows XP-t.

A Java 7 tartalmazott néhány olyan főbb funkciót, mint például:

Tömörített 64 bites mutatók.

Frissített class-loader architektúra.

Több kivétel kezelése.

JVM támogatás a nyelvek dinamikus támogatásához.

Frissített 1.1 sor és JDBC 4.1.

Karakterlánc objektum egy kapcsoló utasításban.

Automatikus erőforrás-kezelés try utasításban és még sok más.

Java7 vs. Java8

Lambda kifejezések

A lambda kifejezések csak törzset és paraméterlistát tartalmaznak.

A Lambda Expressions használatának előnyei,

Továbbfejlesztett iteratív szintaxis.

Olvashatóság.

Kód újrafelhasználása.

JAR fájlméret csökkentése.

Egyszerűsített változó hatóköre.

A funkcionális programozás ösztönzése.

Null Referencia Sablon,Nevezhetjük Java Opcionális Osztálynak is.

DRAFT

A Java Optional osztály használatának előnyei:

Nincs semmiféle kivétel a futás közben.

Tiszta API-k fejleszthetők.

Nincs szükség semmilyen ellenőrzésre.

Új dátum és idő API

A java.util.Date osztályban az évek 1900-tól kezdődnek, a hónapok pedig 0-tól kezdődnek. Ezért fontos tudni, mivel amikor egy évet és egy hónapot szeretnénk kijelölni, annak a következőnek kell lennie: (ÉÉÉÉ-1900) és (HH) -1). Ezt a Java 8 legyőzte a java.time csomagban található osztályok használatával.

Az egész számtól vagy a karaktersorozattól eltérõen a Dátum osztály változtatható. Ami azt jelenti, hogy a dátum megváltoztatható a tulajdonos osztály tudta nélkül. A Java 8 rendelkezik erre megoldással. A Java 8-ban bevezetett LocalDate, LocalTime, LocalDateTime stb. Változhatatlanok (az eredeti objektum nem változik. Valahányszor változtatás történik, egy új objektum kerül visszaadásra).

Unsigned Integer Arithmetic

Nem világos, hogy a Java 8 mit próbált elérni az előjel nélküli egész számtani előrehaladással. De általában egy előjel nélküli egész szám károsnak tekinthető a Java-ra. Bonyolultabbá teszi a típusátalakító eszközöket, a típusrendszert és a könyvtár API-kat.

Stream API

A Stream API használatának legfőbb előnye, hogy egyértelműen növeli a program sebességét és hatékonyságát. Javában erőműnek számít.

Párhuzamos rendezés

A párhuzamos rendezés több szálat használ a művelethez. Gyorsan és hatékonyan teszi a programot. Ez nem lesz annyira látható egy kis adathalmaz mellett. De ha nagy adatkészletet használunk, a hatékonyság és a teljesítmény növekedése nagyon észrevehető lesz.

Új JavaScript motor

A Java 8-mal bevezetett JavaScript-motor neve Nashorn volt.

DRAFT

12. fejezet

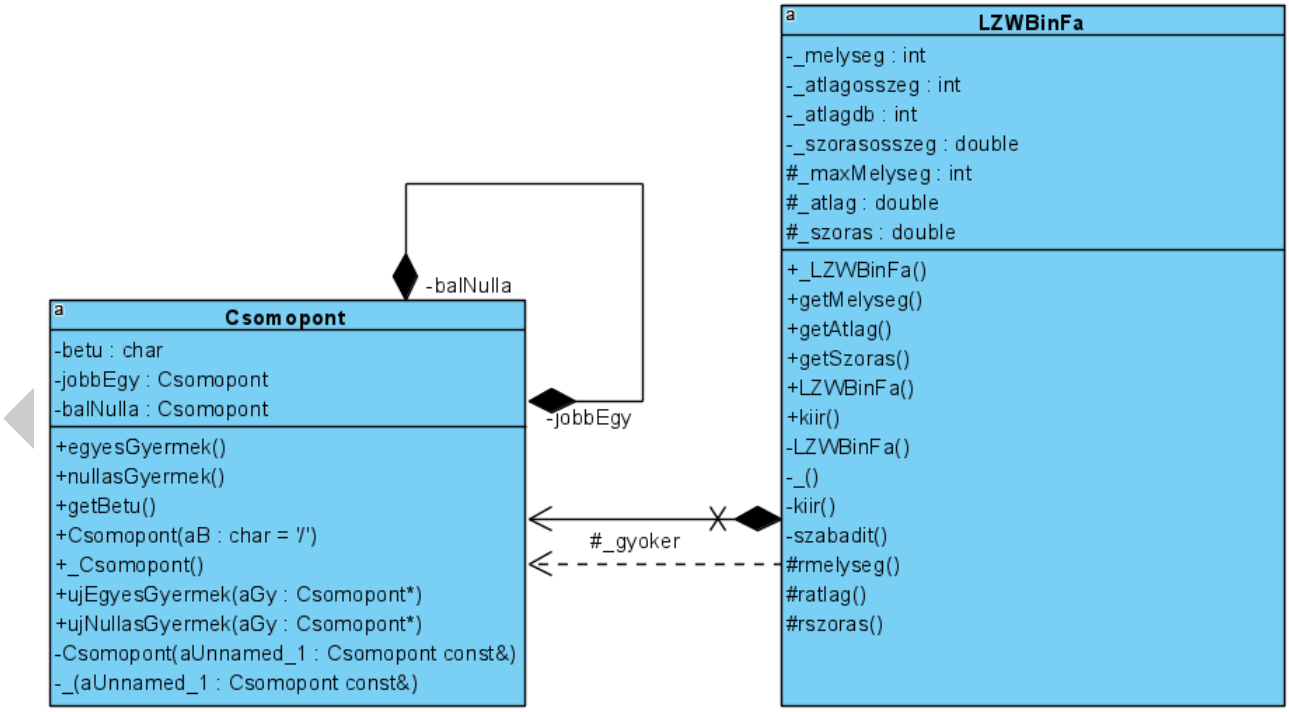
Helló, Mandelbrot!

12.1. Reverse engineering UML osztálydiagram

UML osztálydiagram raj zolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nlERlEOs Lásd fóliák!

Megoldás videó:

Megoldás forrása:



Az osztálydiagram az osztályok közötti kapcsolatot képviseli a programban.

A két osztály két kék színű téglalap.

Az téglalap felső része az osztály attribútumait az alsó a metódusait tartalmazza. A láthatóságukat az elötte lévő jelek is megmutatják.

+ azaz public

- azaz private

azaz protected

A feladat külön kéri, hogy térjünk ki a kompozíció és aggregáció kapcsolatára.

Aggregációról akkor beszélhetünk, amikor az egyik objektum részben (vagy akár egészben) is tartalmazza a másikat. Két típusú van:

erős

gyenge

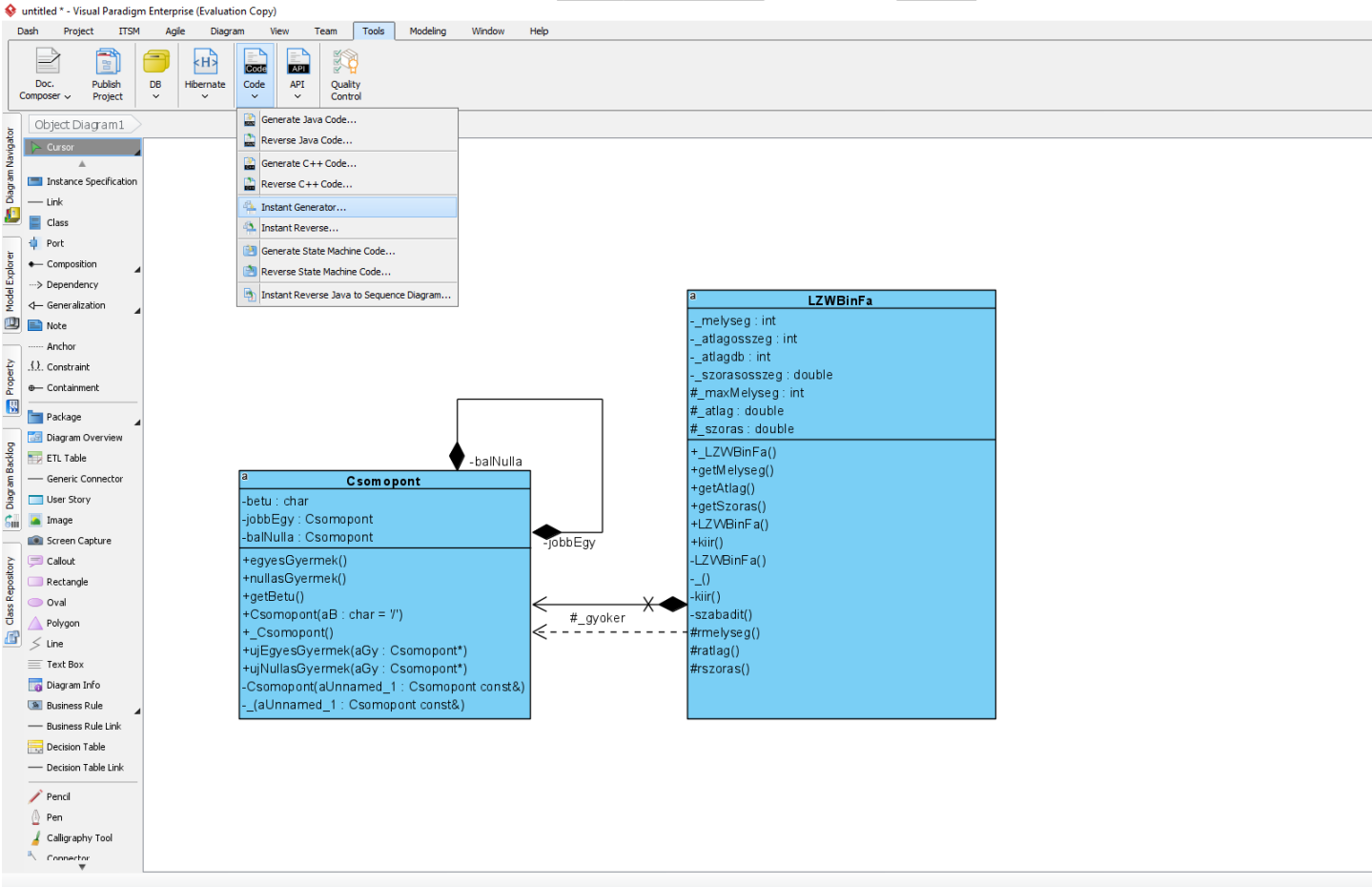
A gyenge aggregációt nevezzük kompozíciónak, azaz a tartamazott objektum illetve a tartalmazó objektum függnek egymástól. Egymás nélkül nem funkcionálnak.

12.2. Forward engineering UML osztálydiagram

UML ben tervezzünk osztályokat és generáljunk belőle forrást!

Megoldás videó:

Megoldás forrása: C++



A "Tools" menün belül a "Code", majd a "Generate C++" lehetőséget választjuk, hogy legenerálja nekünk a forrást C++ nyelven. Ha javaban szeretnénk megkapni a forrást, akkor a "Generate java" opciót válasszuk.

```
#include "Csomopont.h"

void Csomopont::egyesGyermek() {
  // TODO - implement Csomopont::egyesGyermek
  throw "Not yet implemented";
}

void Csomopont::nullasGyermek() {
  // TODO - implement Csomopont::nullasGyermek
  throw "Not yet implemented";
}

void Csomopont::getBetu() {
  // TODO - implement Csomopont::getBetu
  throw "Not yet implemented";
}

Csomopont::Csomopont(char aB) {
  // TODO - implement Csomopont::Csomopont
  throw "Not yet implemented";
}

void Csomopont::_Csomopont() {
  // TODO - implement Csomopont::_Csomopont
  throw "Not yet implemented";
}

void Csomopont::ujEgyesGyermek(Csomopont* aGy) {
  // TODO - implement Csomopont::ujEgyesGyermek
  throw "Not yet implemented";
}

void Csomopont::ujNullasGyermek(Csomopont* aGy) {
  // TODO - implement Csomopont::ujNullasGyermek
  throw "Not yet implemented";
}

Csomopont::Csomopont(Csomopont const& aUnnamed_1) {
  // TODO - implement Csomopont::Csomopont
  throw "Not yet implemented";
}

void Csomopont::_(Csomopont const& aUnnamed_1) {
  // TODO - implement Csomopont::_
  throw "Not yet implemented";
}
```

```
#ifndef CSOMOPONT_H
#define CSOMOPONT_H

class Csomopont {

private:
  char betu;
  Csomopont jobbEgy;
  Csomopont balNulla;

public:
  void egyesGyermek();

  void nullasGyermek();

  void getBetu();

  Csomopont(char aB = '/');

  void _Csomopont();

  void ujEgyesGyermek(Csomopont* aGy);

  void ujNullasGyermek(Csomopont* aGy);

private:
  Csomopont(Csomopont const& aUnnamed_1);

  void _(Csomopont const& aUnnamed_1);
};

#endif
```

```
#include "LZWBinFa.h"

void LZWBinFa::_LZWBinFa() {
  // TODO - implement LZWBinFa::_LZWBinFa
  throw "Not yet implemented";
}

void LZWBinFa::getMelyseg() {
  // TODO - implement LZWBinFa::getMelyseg
  throw "Not yet implemented";
```

```
}

void LZWBinFa::getAtlag() {
  // TODO - implement LZWBinFa::getAtlag
  throw "Not yet implemented";
}

void LZWBinFa::getSzoras() {
  // TODO - implement LZWBinFa::getSzoras
  throw "Not yet implemented";
}

void LZWBinFa::_() {
  // TODO - implement LZWBinFa::_
  throw "Not yet implemented";
}

void LZWBinFa::szabadit() {
  // TODO - implement LZWBinFa::szabadit
  throw "Not yet implemented";
}

void LZWBinFa::rmelyseg() {
  // TODO - implement LZWBinFa::rmelyseg
  throw "Not yet implemented";
}

void LZWBinFa::ratlag() {
  // TODO - implement LZWBinFa::ratlag
  throw "Not yet implemented";
}

void LZWBinFa::rszoras() {
  // TODO - implement LZWBinFa::rszoras
  throw "Not yet implemented";
}
```

```
#ifndef LZWBINFA_H
#define LZWBINFA_H

class LZWBinFa {

protected:
  Csomopont _gyoker;
```

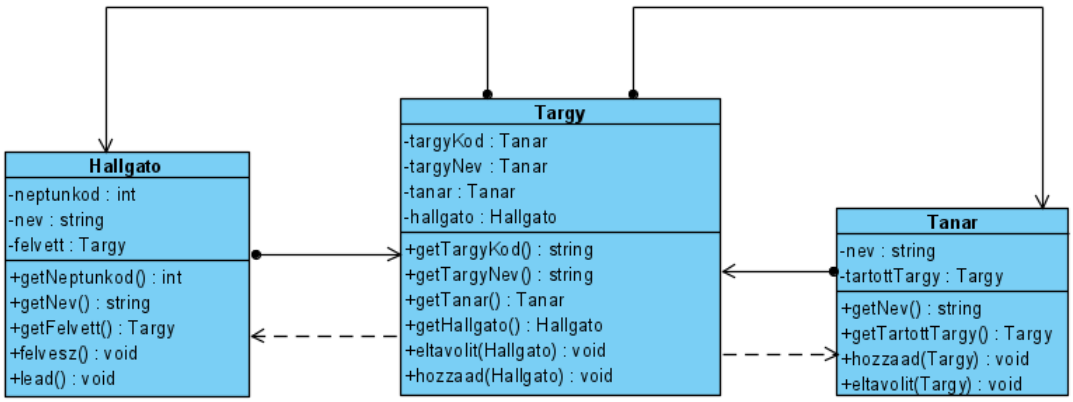
```
private:
  int _melyseg;
  int _atlagosszeg;
  int _atlagdb;
  double _szorasosszeg;
protected:
  int _maxMelyseg;
  double _atlag;
  double _szoras;

public:
  void _LZWBinFa();

  void getMelyseg();

  void getAtlag();

  void getSzoras();

  LZWBinFa();

  void kiir();

private:
  void _();

  void szabadit();

protected:
  void rmelyseg();

  void ratlag();

  void rszoras();
};

#endif
```

12.3. Neptun tantárgyfelvétel modellezése UML ben

Modellezd le a Neptun rendszer tárgyfelvételéhez szükséges objektumokat UML diagramm segítségével.

Megoldás videó:

Megoldás forrása:



12.4. Neptun tantárgyfelvétel UML diagram implementálása

Implementáld le az előző feladatban létrehozott diagrammot egy tetszőleges nyelven.

Megoldás videó:

Megoldás forrása:

C++

java

DRAFT

13. fejezet

Helló, Chomsky!

13.1. Encoding

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Chomsky!/-Encoding

A futtassuk során egy olyan hibaüzenet jelent meg. Olyan karakterek szerepelnek a kódban, amiket nem tud értelmezni a fordító, mivel nem UTF-8as kódolás.

8859_2 ISO Latin-2 -es kódolással orvosolhatjuk is a próblémát.Így a fordításuk a következő:

javac -encoding ISO8859_2 MandelbrotHalmazNagyító.java

javac -encoding ISO8859_2 MandelbrotIterációk.java

13.2. l334d1c4

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Chomsky!/-l334d1c4

A leetspeak a kommunikáció során a karaktereket(betűket, számokat) hozzájuk hasonlóra cseréli.A 80as években alakult ki.

"Maga a leet szó leetspeakben írva így fest: 1337. A szó egyébként az angol elite szóalak elvonásával jött létre követve az élőszó hangzását."

DRAFT

Először minden Java könyvtárat importálok. Majd a "LeetChyper" osztályban létrehozok egy stringet az általunk megadott szövegnek amellyet átalakítani szeretnénk, illetve egy másik stringet a kész, azaz átalakított szövegnek. Az eredményt print-el írom ki. A következő lépés az "atalakit" fgv volt, amelybe feltöltöttem a "character" nevű MAP-et a megfelelő párokkal. Utána egy for ciklus következik. A for ciklussal végigmegyek az általunk megadott szó betűin ahol a leetspeak-es megfelelőjét szóközökkel elválasztva tárolom egy változóban. Utána létrehoztam a konstruktort is, ahol meghívtam a leet átalakításhoz szükséges metódust. Itt kapott helyet a kiírás is. A main-en belül egy Stringbuildert hozok létre, amivel a már átalakított stringet építem fel egy for ciklussal, ami végigmegy a parancssori argumentumként megadott szón. Végezetül a "LeetChyper" objektumot hozom létre.

13.3. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! Lásd https://youtu.be/XpBnR31BRJY

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Chomsky!/-Perceptron

Érdemes lenne először a fogalmat megmagyarázni. A perceptron nem más mint ezen neuron mesterséges intelligenciában használt változata. Tanulásra képes, a bemenő 0-k és 1-esek sorozatából mintákat tanul meg és súlyozott összegzést végez. A következő feladat során egy ilyen perceptront fogunk elkészíteni, aminek esetünkben a feladata az lesz, hogy a mandelbrot.cpp programunk által létrehozott Mandelbrot-halmazt ábrázoló PNG kép egyik színkódját vegye és az a színkód legyen a többrétegű neurális háló inputja. Lássuk a programot!

Include-oljuk az iostream, az mlp és a png++/png könyvtárakat. Utóbbi kettőt azért, mert többrétegű perceptront akarunk majd létrehozni (Multi Layer Perceptron), így muszáj ezt a könyvtárat include-olnunk a programunkba. Utolsó könyvtárunk ahogy a neve is sugallja, a PNG képállományokkal való munkát teszi lehetővé. Előzőleg telepíteni szükséges, ha nem megtalálható a gépünkön.

Kezdődik a main függvényünk. Első sorában megmondjuk, hogy az 1-es parancssori argumentum alapján kerül beolvasásra a képállomány. Ettől kezdve dolgozni tudunk A kép méretét a get_width és a get_height szorzatából kapjuk, ezt el is tároljuk egy változóban. A következő sorban létrehozásra példányosítunk egy perceptront a new operátor segítségével, amely paraméterei balról jobbra haladva: a rétegek száma, 1. réteg neuronjai az inputrétegben, 2. réteg neuronjai az inputrétegben, az eredmény (jelen esetben 1 szám)

Létrehozásra kerül egy double típusú mutató. Jönnek a for ciklusok. Az egyik for ciklus végigmegy a kép szélességét alkotó pontokon, a másik pedig a magasságán. Miután végigmentünk a képpontokon, az image tárolni fogja a képállomány vörös színkomponensét. Tehát a beolvasásra került képállomány Univerzális programozás 231 / 303 piros vörös komponensét a lefoglalt tárba másoljuk bele. A value értéke a Perceptron image-re történő meghívása adja majd. Így a perceptronban tárolásra kerül a vörös színkomponens. A value változó egy double típusú értéket tárol. Ez kiiratásra kerül és töröljük tovább nem használatos elemeket, hiszen a számukra eddig fenntartott memóriaterületet érdemes felszabadítanunk, így a lefoglalt memóriamennyiség újra használható. Programunk ezzel véget is ért. Fordítjuk és futtatjuk a képen látható módon:

g++ mlp.hpp main.cpp -o perc -lpng -std=c++11

./perc elso.png

DRAFT

13.4. Full screen

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

Megoldás videó:

Megoldás forrása: hhttps://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Chomsky!/-Full%20screen

Importáljuk a grafikus ablakban való megjelenítéshez és annak beállításához szükséges osztályokat.

Létrehozásra kerül a fullscreen osztály ami az előbb importált JPanel osztály leszármazottja lesz. Egy stringben megadom a kiiratásra szánt szöveget majd a paint metódusban attribútumként megadott objektumpéldány megjelenítését módosítom a függvénytörzsben. A setFont a szöveg karaktereinek megjelenését hivatott beállítani. A paraméterek a következők: betűtípus, típus(esetünkben félkövér) és betűméret. A setColor értelemszerűen a kiiratandó szöveg színét változtatja meg. A drawString függvénnyel pedig egy adott stringet adott koordinátákon jeleníthetünk meg.

A main fgv-en belül példányosítok egy JFrame példányt, aminek paramétere az a string lesz, mely a program ablakának "neve" lesz. Hozzáadom a frame-hez a fullscreen osztály egy példányát majd beállításra kerül a frame mérete pixelben a setSize függvénnyel, a láthatósága a setVisible függvénnyel, a viselkedés amit tanusít, amikor bezárjuk az adott ablakot a setDefaultCloseOperation-nal - ami ebben az esetben nem jelent mást, mint a programból való kilépést , illetve az ablak átmérezhetőségét a setResizable függvénnyel.

Fordítás, futtatás:

javac fullscreen.java

java fullscreen

DRAFT

14. fejezet

Helló, Stroustrup!

14.1. JDK osztályok

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Stroustrup!/-JDK%20oszt%C3%A1lyok

A feladat megkezdését természetesen a leírás által is tanácsolt fénykard program megkeresésével kezdtem. Bátfai Tanár Úr FUTURE projektjének programkódjai között sikerült is rátalálnom: https://github.com/-nbatfai/future/blob/master/cs/F7/fenykard.cpp

Ami nekünk ebből a kódból érdekes lesz, az a következő, "read_acts" függvény:

```
        void read_acts(boost::filesystem::path path, std::map <std::string, ↩
            int> &acts)
        {

            if (is_regular_file(path)) {

                std::string ext(".props");
                if (!ext.compare(boost::filesystem::extension(path))) {

                    std::string actpropspath = path.string();
                    std::size_t end = actpropspath.find_last_of("/");
                    std::string act = actpropspath.substr(0, end);

                    acts[act] = get_points(path);

                    std::cout << std::setw(4) << acts[act] << "    " << act ↩
                        << std::endl;
                }
```

DRAFT

```
        } else if (is_directory(path))
            for (boost::filesystem::directory_entry & entry : boost:: ↩
               filesystem::directory_iterator(path))
                read_acts(entry.path(), acts);

    }
```

Mit csinál ez a függvény? Végigmegy egy állományszerkezeten és egy adott kiterjesztésű (esetben .props) fájlokat keres, majd a bennük található információkat kiolvassa és eltárolja.

Ha jobban belegondolunk, lényegében most is valami hasonlót kell csinálnunk. Hiszen egy állományszerkezetet kell bejárnunk és szintén egy adott kiterjesztésű (.java) fájlokat keresünk.

Lássuk, a fénykardos példából kiindulva hogyan is sikerülne ezt megvalósítani!

```
#include <iostream>
#include <string>
#include <fstream>
#include <iomanip>
#include <vector>

#include <boost/filesystem.hpp>

using namespace std;

int counter=0;
```

Inkludáljuk a megfelelő osztályokat és deklarálunk egy változót az osztályok számának tárolására.

```
void read_classes (boost::filesystem::path path, vector<string>  ↩
   acts)
{
    if(is_regular_file(path))
    {
    string ext(".java");
    if(!ext.compare(boost::filesystem::extension (path)))
    {
        cout<<path.string()<<'\n';
        string actjavaspath=path.string();
        size_t end = actjavaspath.find_last_of("/");
        string act = actjavaspath.substr(0,end);
        acts.push_back(act);
        counter++;
    }
}
  else if(is_directory(path))
  for(boost::filesystem::directory_entry & entry :
      boost::filesystem::directory_iterator (path))
  read_classes(entry.path(),acts);
}
```

Az eljárás (aminek egyébként két paramétere van: az elérési hely - vagyis ahol keresnie kell - illetve a keresett elemeket tároló vector) a következőképpen működik:

Alapvetően egy feltételvizsgálat következik, aminek két ága van. Az if-fel bevezetett feltételvizsgálatban, azon belül pedig az "is_regular_file()" függvénnyel megvizsgáljuk, hogy a fájlhiearchiában jelenleg "hol állunk"..Tehát hogy ha ún. "regular fájlok" vagyis hagyományos fájlok, nem könyvtárak állnak rendelkezésre vizsgálatra, akkor megvizsgáljuk, mely fájlok kiterjesztése .java. Ezeket az állományokat a vectorba gyűjtjük és növeljük a számlálót. Eltároljuk a keresett fájlok elérési útvonalát is két segédváltozó "end" és "actjavaspath" segítségével.

Ha nem fájlokkal vagyunk körülvéve ott, ahol jelenleg állunk, meghívásra kerül a "is_directory" függvény, tehát ez az a helyzet, mikor még a könyvtárak szintjén állunk a "kutakodásban".. Ekkor egy for ciklus indul és rekurzívan meghívjuk a "read_classes" függvényt. Így előbb-utóbb eljutunk a fájlok szintjére. (Már ha nem vagyunk ott jelenleg..)

```
int main( int argc, char *argv[])
{
    vector<string> acts;
    read_classes("src", acts);
    cout << "Ennyi Java osztalyt talaltunk: "<< counter << "\n";
}
```

A "main" függvényben járunk már. Létrehozásra kerül az osztályokat tároló vectorunk, meghívásra kerül a "read_classes" függvény és kiiratjuk, hány osztály is volt megtalálható az src mappában.

14.2. Hibásan implementált R SA törése

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Stroustrup!/-Hib%C3%A1san%20implement%C3%A1lt%20RSA%20t%C3%B6r%C3%A9se

Bevezető forrásai: https://hu.wikipedia.org/wiki/RSA-eljárás, https://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_informatikai_biztonsag_es_kriptografia/ch08s07.html

Először pár szóban az RSA-ról: napjaink egyik legszélesebb körben használt titkosító eljárása. Az eljárás neve lényegében egy mozaikszó, az algoritmus alkotói: Ron Rivest, Adi Shamir és Len Adleman kezdőbetűiből áll össze, 1978-ban publikálták. Működése matematikai alapokon, pontosabban a Fermat-tételen nyugszik. Érdekes jellemzője az algoritmusnak, hogy két kulccsal, egy nyílttal vagy nyilvánossal és egy titkos kulcssal dolgozik. Mindkét kulcs egy-egy számpár. Ez azért érdekes, mert ezzel teljesen ketté lesz választva a titkosítás és a törés folyamata. Mivel a kódolás és a dekódolás paraméterei nem lesznek ugyanazok, így az egyik meghatározása a másikból sem lesz megoldható.

Az RSA titkosításról minden további tudnivaló itt

Fontos említést tennünk a BigInteger osztályról is, amit a feladat során természetesen használni is fogunk. A BigInteger osztály már egész régóta a JDK részét képezi, jelenleg a java.math csomagban találhatjuk

DRAFT

meg. Gyakran használják kriptográfiai kódolók-törők elkészítéséhez, sokak számára használatuk összeolvadt ezzel a fajta felhasználással. A típust egy egész értékből és 32-bites egészből úgynevezett skálázó faktor alkotja.

Lássuk a programunkat!

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map.Entry;
```

Először is importálunk minden olyan könyvtárat, melyekre a feladat megoldása során szükségünk lesz. Itt található a bevezetőben már említett "BigInteger " osztály is.

```
  public class rsa_cipher {
  public static void main(String[] args) {
    int bitlength = 2100;

    SecureRandom random = new SecureRandom();

    BigInteger p = BigInteger.probablePrime(bitlength/2, random);
    BigInteger q = BigInteger.probablePrime(bitlength/2, random);

    BigInteger publicKey = new BigInteger("65537");
    BigInteger modulus = p.multiply(q);

    String str = "this is a perfect string".toUpperCase();
    System.out.println("Eredeti: " + str);
```

Haladjunk logikai sorrendben. A "main "ünket tartalmazó, "rsa_cipher " osztályban járunk. Itt történik a bithossz beállítása, itt kerülnek létrehozásra a "BigInteger " számaink. Ilyen érték lesz többek között a nyilvános kulcsunk és a modulus is. Megadjuk eredeti, átalakításra váró szövegünket, amit a "str " nevű, String típusú változóban el is tárolunk.

```
    byte[] out = new byte[str.length()];
    for (int i = 0; i < str.length(); i++) {
      char c = str.charAt(i);
      if (c == ' ')
        out[i] = (byte)c;
      else
        out[i] = new BigInteger(new byte[] {(byte)c}).modPow(publicKey,  ↩
           modulus).byteValue();
    }
    String encoded = new String(out);
    System.out.println("Kodolt:" + encoded);

    Decode de = new Decode(encoded);
    System.out.println("Visszafejtett: " + de.getDecoded());
```

```
  }
}
```

A "main "ből kiemelt pár sor végzi a titkosítást, méghozzá a stringünk minden egyes karakterén végighaladva, egyenként teszi meg azt. Egy byte elemeket tartalmazó tömbben kerülnek tárolásra a titkosított karakterek, melyekből stringet készítünk, neve "encoded " lesz melyet ki is iratunk. A következő sorban létrehozunk egy új "Decode " objektumot, ami a már gyakoriság alapon visszafejtett stringünket tartalmazza. Kérdés az, hogyan is működik ez a fajta visszafejtés. Vessünk egy pillantást a következő függvény(ek)re!

```
  private void loadFreqList() {
  BufferedReader reader;
  try {
    reader = new BufferedReader(new FileReader("gyakorisag.txt"));
    String line;
    while((line = reader.readLine()) != null) {
      String[] args = line.split("\t");
      char c = args[0].charAt(0);
      int num = Integer.parseInt(args[1]);
      this.charRank.put(c, num);
    }
  } catch (Exception e) {
    System.out.println("Error when loading list -> " + e.getMessage());
  }
}
```

A függvényen belül egy try-catch szerkezetben történik a lényeg. Ugyanis beolvasásra kerül egy gyakoriság lista. Karakterenként végigmegy a lista elemein és abban az esetben, ha az aktuális betű már megtalálható benne, a gyakorisága növelésre kerül. Ha új betűvel találkozik, gyakorisága 1-re állítódik be. Mindez a try-on belül volt megtalálható. Catch-csel hibaüzenetet dobunk ha nem sikerült a listát betöltenünk.

```
private char nextFreq() {
  char c = 0;
  int nowFreq = 0;
  for(Entry<Character, Integer> e : this.charRank.entrySet()) {
    if (e.getValue() > nowFreq) {
      nowFreq = e.getValue();
      c = e.getKey();
    }
  }
  if (this.charRank.containsKey(c))
    this.charRank.remove(c);
  return c;
}
```

Lássuk, mit tesz a "nextFreq " függvényünk. Lényegében rendezéssel, pontosabban maximum-kiválasztásossal a gyakorisági listában lévő betűk behelyettesítése történik. Látható, hogy alapvetően a gyakorisági listánk az, ami befolyásolja majd a kapott eredményünket, hiszen a nagyobb gyakorisági értékkel rendelkező karaktereket helyezi előtérbe majd az algoritmus.

```
    public Decode(String str) {
    this.charRank = new HashMap<Character, Integer>();
    this.decoded = str;

    this.loadFreqList();

    HashMap<Character, Integer> frequency = new HashMap<Character, Integer ↩
       >();
    for (int i = 0; i < str.length(); i++) {
      char c = str.charAt(i);
      if (c != ' ')
        if(frequency.containsKey(c))
          frequency.put(c, frequency.get(c) + 1);
        else
          frequency.put(c, 1);
    }

    while (frequency.size() > 0) {
      int mi = 0;
      char c = 0;
      for (Entry<Character, Integer> e : frequency.entrySet()) {
        if (mi < e.getValue()) {
          mi = e.getValue();
          c = e.getKey();
        }
      }
      this.decoded = this.decoded.replace(c, this.nextFreq());
      frequency.remove(c);
    }
  }
```

Már csak a "main "ben meghívott "Decode " függvény tárgyalása maradt hátra. Nem túltárgyalva, ez az a függvény, amely az előzőleg tárgyalt függvényeket és gyakorisági listát felhasználva végzi el a dekódolást. Nyilván a while függvényből láthatjuk, hogy mindezt addig teszi, amíg karaktereink el nem fogytak.

14.3. Változó argumentumszámú ctor és Összefoglaló

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Stroustrup!/-V%C3%A1ltoz%C3%B3%20argumentumsz%C3%A1m%C3%BA%20ctor

Tanulságok, tapasztalatok, magyarázat...

Források:

http://www.cs.ubbcluj.ro/~csatol/mestint/pdfs/neur_halo_alap.pdf

https://www.tankonyvtar.hu/hu/tartalom/tamop425/0026_neuralis_4_4/ch04.html

https://gyires.inf.unideb.hu/GyBITT/19/ch03s02.html

Érdemes lenne először a fogalmakat megmagyarázni.

Először is, hogyan lehet egy konstruktor változó argumentumszámú és hogyan jelenik meg ez a mi példánkban?

```
class Perceptron
{
public:
  Perceptron ( int nof, ... )
  {
```

Változó argumentumszámú konstruktorról akkor beszélhetünk, mikor nincsen egyértelműen definiálva, egy konstruktornak hány paraméterrel kell rendelkeznie. A mi programunk mlp.hpp header-fájljában is ezzel találkozhatunk. Ebből annyit olvashatunk le, hogy egy argumentumot fixen tartalmaznia kell (ez esetünkben az Integer típusú "nof " lesz), a változó argumentumszámot pedig a "..." karaktersorozat jelzi számunkra.

A perceptron nem más mint ezen neuron mesterséges intelligenciában használt változata. Szintén tanulásra képes, a bemenő 0-k és 1-esek sorozatából mintákat tanul meg és súlyozott összegzést végez. Ennek több változata is létezik, mi ezen feladat során a többrétegűekkel fogunk foglalkozni, ez a Multi Layer Perceptron (MLP), amely az egyik, ha nem a leggyakrabban használt hálózat-architektúra. Ez a fajta neurális hálózat 3 rétegből épül fel, melyek a következők: bemeneti réteg: azok a neuronok találhatók itt, amelyek a bemeneti jel továbbítását végzik a hálózat felé. A legtöbb esetben nem jelöljük őket külön; rejtett réteg: a tulajdonképpeni feldolgozást végző neuronok tartoznak ide. Egy hálózaton belül több rejtett réteg is lehet; kimeneti réteg: az itt található neuronok a külvilág felé továbbítják az információt. A feladatuk ugyanaz, mint a rejtett rétegbeli neuronoké.

Amint már említettem, a többrétegű perceptron minden egyes rétege neuronokból áll. Biztosítani kell azonban, hogy a neurális hálózatunk kimenete a súlyok folytonos, differenciálható függvénye legyen.

Fontos az is, hogyan történik a neurális hálónk tanítása. A tanítási folyamat egy ún. ellenőrzött tanítás, ahol a hálózat kimenetén értelmezett hiba felhasználásával határozhatjuk meg a kritériumfüggvény vagy kockázat paraméterfüggését. A tanítás leggyakrabban a hiba-visszaterjesztéses módszerrel valósul meg. A tanítás fő lépései: a kezdeti súlyok megadása; a bemeneti jelet (azaz a tanító pontot) végigáramoltatjuk a hálózaton, de a súlyokat nem változtatjuk meg; Az így kapott kimeneti jelet összevetjük a tényleges kimeneti jellel; A hibát visszaáramoltatjuk a hálózaton, súlyokat pedig megváltoztatjuk a hiba csökkentése érdekében.

A következő feladat során egy ilyen perceptront fogunk elkészíteni, aminek alapvetően a feladata az, hogy a "mandelbrot.cpp " programunk által létrehozott Mandelbrot-halmazt ábrázoló (elsomandel.png) PNG képet felhasználva generáltassunk egy vele megegyező méretű képet. Lássuk a programot!

Kezdjük a feladat értelmezésével. A feladat szinte ugyanazt kéri, mint az előző Perceptronos feladatainkál egy kis csavarral. Ugyanis az eddigi példáink során a Perceptronunk egy képállományra egy értéket adott vissza.

A kód eleje megegyezik az előző fejezetben már tárgyaltéval ezért az elemzés egy része onnan került átemelésre, innen a hasonlóság.

DRAFT

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

Include-oljuk az iostream, az "mlp " és a "png++/png " könyvtárakat. Utóbbi kettőt azért, mert többrétegű perceptront akarunk majd létrehozni (Multi Layer Perceptron), így muszáj ezt a könyvtárat include-olnunk a programunkba. Utolsó könyvtárunk ahogy a neve is sugallja, a PNG képállományokkal való munkát teszi lehetővé. Előzőleg telepíteni szükséges, ha nem megtalálható a gépünkön.

```
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, size);
```

Kezdődik a main függvényünk. Első sorában megmondjuk, hogy az 1-es parancssori argumentum alapján kerül beolvasásra a képállomány. Ettől kezdve dolgozni tudunk A kép méretét a "get_width " és a "get_height " szorzatából kapjuk, ezt el is tároljuk egy változóban. A következő sorban létrehozásra példányosítunk egy perceptront a `new` operátor segítségével, amely paraméterei balról jobbra haladva: a rétegek száma, 1. réteg neuronjai az inputrétegben, 2. réteg neuronjai az inputrétegben, az eredmény (jelen esetben egy, a már megszokott szám helyett egy képállomány)

```
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;
```

Létrehozásra kerül egy `double` típusú mutató. Jönnek a for ciklusok. Az egyik for ciklus végigmegy a kép szélességét alkotó pontokon, a másik pedig a magasságán. Miután végigmentünk a képpontokon, az `image` tárolni fogja a képállomány vörös színkomponensét. Tehát a beolvasásra került képállomány piros vörös komponensét a lefoglalt tárba másoljuk bele.

```
double* newPicture = (*p) (image); // eddig: double value = (*p)( ↩
   image);

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

png_image.write("kimeneti.png");

delete p;
delete [] image;
```

Itt válik érdekessé a dolog, hiszen már nem "csak" egy értéket akarunk kiiratni, hanem egy képet generáltatni, ahhoz azonban az egy double csillag típusra van már szükségünk. Ezt követően két for ciklussal végigmegyünk az eredeti kép szélességén és magasságán és az új képünk megkapja a színadatokat. Ezt követően a "write " függvénnyel létrehozásra kerül az új képállományunk kimeneti néven, png formátumban.

A végén törlésre kerülnek az eddig, de tovább már nem használatos elemeket, hiszen a számukra eddig fenntartott memóriaterületet érdemes felszabadítanunk, így a lefoglalt memóriamennyiség újra használható. Programunk ezzel véget is ért.

Természetesen ahhoz, hogy megfelelően működjön a programunk, többek között bele kellett nyúlni egy kicsit az "mlp.hpp " fájlba is.

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

Többek között az () operátor működésébe is, ami mostantól nem egy double értéket, hanem egy double pointert ad vissza.

Fordítsuk és futtassuk a programunkat!

És az eredmény:

DRAFT

15. fejezet

Helló, Gödel!

15.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban https://youtu.be/DL6iQwPx1Yw (8:05-től)

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20G%C3%B6del!/-Gengszterek

https://github.com/nbatfai/robocar-emulator

A Robocar World Championship (OOCWC = rObOCar World Championship) egy olyan projekt/platform mely lehetőséget ad/adott a robotautó-kutatásra. Az egész projekt középpontjában a Robocar City Emulator áll.

A lambda-kifejezések a C++ 11-es verziójában jelentek meg. Ezek a kifejezések lehetővé teszik számunkra az úgynevezett in-line function-ök írását, vagyis egy- vagy kevés soros függvényekét. Fontos, hogy ezek olyan függvények, melyekre tipikusan egyszer van szükségünk, többször nem akarjuk felhasználni őket. Lényegében "egyszer használatosak", ezért még nevet SEM adunk nekik.

Egy egyszerű példa lambda-kifejezésre:

```
[](int x, int y) -> { return x + y; }
```

A szögletes zárójel jelzi számunkra, hogy lambda-kifejezés kezdődik. Ezután a zárójelekben a paraméterek találhatók, majd a nyilacskát követően a kapcsos zárójelek között a függvény maga található az elvégzendő művelettel. Itt dől el az is, mi lesz a visszatérési érték típusa.

Forrás: https://en.cppreference.com/w/cpp/language/lambda

Lássunk a mi példánkat!

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

DRAFT

```
        void sort (RandomAccessIterator first, RandomAccessIterator last,  ↩
           Compare comp);
```

A "gangsters " vektor kerül rendezésre, ezt a "sort " függvény első és második paramétere teszi számunkra világossá (gangsters.begin() és gangsters.end()). A harmadik paraméter nem lesz más, mint maga a lambda-kifejezésünk. Ezen lambda-kifejezés paramétere két Gangster lesz. A kifejezés egy boolean értéket, pontosabban true-t ad vissza, ha az x gengszter és a rendőr távolsága kisebb, mint az y gengszter és a rendőr távolsága.

Tehát ha értelmezzük az előző pár sort, elmondhatjuk: ez esetben a vektorunk a gengszterek rendőrököz való távolsága alapján lesz rendezve.

A feladat megoldása során egyébként az utolsó sorban található "sort " függvényt használtuk, melynek harmadik paramétere lehetővé teszi, hogy kézzel adjuk meg az összehasonlítási szemponto(ka)t. Ez esetünkben a lambda-kifejezés.

15.2. STL map érték szerinti rendezése

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20G%C3%B6del!/-STL%20map%20%C3%A9rt%C3%A9k%20szerinti%20rendez%C3%A9se

Tanulságok, tapasztalatok, magyarázat...

Az STL (Standard Template Library) olyan C++ sablon- vagy mintaosztályok összessége, amelyek implementálnak -ezzel lehetőséget adva használatukra- számos, széles körben előszeretettel használt algoritmus és adatszerkezet használatát. Három fő dolgot tartalmaz: tárolókat, algoritmusokat és iterátorokat.

Esetünkben a tárolók lesznek érdekesek, sőt, abból is egy különleges fajta, a **map**. Mit érdemes tudni róla? Asszociatív tárolók, vagyis a kulcsuk által azonosíthatók az elemek. Ezzel le is lőttem a poént, kulcs-érték párokat tartalmaz. A benne található elemek minden esetben rendezettek, azok kulcsuk alapján.

Bevezető forrása: link, link

Ez elő is vetíti számunkra a feladatot...Hiszen most nem kulcsuk, hanem értékük szerint kellene rendeznünk a map-et. Nézzük a megoldást!

```
        #include <map>
        #include <iostream>
        #include <algorithm>
        #include <vector>

        std::vector<std::pair<std::string, int>> sort_map ( std::map <std:: ↩
           string, int> &rank )
        {
                std::vector<std::pair<std::string, int>> ordered;

                for ( auto & i : rank ) {
```

DRAFT

```
                        if ( i.second ) {
                                std::pair<std::string, int> p {i.first, i. ↩
                                   second};
                                ordered.push_back ( p );
                        }
                }

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

                return ordered;
        }
```

A megoldás a fénykard programból lett átemelve. A "sort_map " függvényünk fogja megvalósítani az érték szerinti rendezést. A függvény visszatérési értéke vektorpárok lesznek. Mi történik a függvénytörzsön belül? Először is létrehozzuk a vektort, ami a visszatérési értékünk lesz "ordered " névvel. Majd egy for végigmegyünk a "rank " map-en. Azt nézi, hogy vannak-e a vektorban érték párosok. Ha vannak, akkor egy "pair "be kerülnek bele. Ezeket a "pair "eket pedig az "ordered " vektorba nyomjuk.

Az értékpárokkal feltöltött vektor rendezéshez segítségül a már ismert az "std::sort " függvényt hívjuk meg ami harmadik paramétereként egy, szögletes zárójellel bevezetett lambda kifejezést alkalmazva rendez úgy, hogy végigmegy a vektorunkon és igazzal tér vissza abban az esetben, ha az első visgált paraméter nagyobb, mint a második. A függvényünk visszatérési értéke a "ordered ", már rendezett vektor lesz.

```
        int main()
        {
            std::map<std::string, int> map;
            map["a"]=33;
            map["b"]=45;
            map["c"]=7;

            std::vector<std::pair<std::string, int>> sorted = sort_map(map) ↩
               ;

            for(auto & i : sorted)
            {
                std::cout <<i.first << " " <<i.second << std::endl;
            }
        }
```

A "main "ben létrehozunk egy map-et, azt feltöltjük elemekkel, majd az előbb tárgyalt "sort_map " függvénnyel rendezzük azt. Ezt, a rendezés utáni állapotot egy új, "sorted " vektorban tároljuk el. Majd végül egy for ciklussal végigmegyünk új vektorunkon és kiiratjuk a rendezés utáni állapotot.

Fordítás és futtatás után a kis mintaprogramunk a következő eredménnyel tér vissza:

15.3. Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang Interface Comparable<T> szerepét!

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20G%C3%B6del!/-Alternat%C3%ADv%20Tabella%20rendez%C3%A9se

Tanulságok, tapasztalatok, magyarázat...

Talán érdemes lenne azzal kezdeni, mi is az az alternatív tabella? Az alternatív tabella nem más, mint az egyes futballbajnokságokhoz készített olyan sorrend, ami nem a megszokott módon (amikor is a győzelem 3 , a döntetlen 1, a vereség pedig 0 pontot ér...) számolja a csapatok sorrendjét, hanem megpróbálja figyelembe venni azt, hogy egy adott csapat éppen melyik másik csapattal szemben érte el az éppen adott eredményét. Hiszen ha belegondolunk, a valóságban is "értékesebbnek" tartjuk a csapatunk által szerzett azt a 3 pontot, amelyet valamelyik éllovas ellen szerzett, mintsem azt, amelyet az éppen aktuális kiesőjelöltek ellen sikerült begyűjteni.

Az alternatív tabellák elkészítésének egyik módja, hogy a Google PageRank algoritmusát használjuk fel ehhez. Tehát ez esetben az alternatív sorrend úgy alakul ki, hogy az a csapat kerül rajta előrébb, amely előkelőbb helyen lévő csapatoktól szerez pontot.

A bevezető forrása: https://hu.wikipedia.org/wiki/Alternativ_tabella

Az eredeti- és az alternatív tabella összehasonlítása. A kép forrása: link

Most nézzük meg a feladat által kért interface-t!

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

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

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

Íme a java.lang package Comparable interface melyet a "Csapat " osztály implementál. Két objektum kerül összehasonlításra a "compareTo " függvénnyel, az aktuális, a hívottal. Ha a hívó objektum értéke kisebb, mint a paraméterként átadott objektumé, akkor - 1-t ad vissza, fordított esetben 1-et, egyenlőség esetén 0-t.

Miért is jó nekünk ez? A programunk következő sorai miatt:

```
            java.util.List<Csapat> rendezettCsapatok = java.util.Arrays. ↩
               asList(csapatok);
         java.util.Collections.sort(rendezettCsapatok);
```

Egy kis magyarázat ehhez a kódcsipethez..Létrejön a "Csapat " típusú, "rendezettCsapatok " névre hallgató listánk, amelyet a következő sorban a "sort " metódussal rendezünk. Kis utánajárást követően megtaláltam, miért cselekedtünk az előzőekben úgy, ahogy. A "sort " metódus nem mindenfajta listán működik, hanem csakis olyanokon melynek tagjai implementálják a tárgyalt "Comparable " interface-t.

Ennek "bizonyítéka", mely megtalálható a jdk-ban:

```
      "Lists (and arrays) of objects that implement this interface can be  ↩
         sorted
      automatically by {@link Collections#sort(List) Collections.sort} (and
      {@link Arrays#sort(Object[]) Arrays.sort}).  Objects that implement  ↩
         this
      interface can be used as keys in a {@linkplain SortedMap sorted map}  ↩
         or as
      elements in a {@linkplain SortedSet sorted set}, without the need to
      specify a {@linkplain Comparator comparator}."
```

Úgy gondolom, így már mindenképp érthetőbb a megoldásunk.

Azt érdemes megemlíteni, hogy a szükségünk lesz a "Wiki2Matrix.java " állományra melyet ha lefuttatunk, egy linkmátrixszal leszünk gazdagabbak. A linkmátrix tartalmát az "AlternativTabella.java " kódunkba szükséges illesztenünk, az "L " kétdimenziós tömbbe. Csak ezután fordíthatjuk és futtathatjuk utóbb említett fájlunkat.

Fordítás és futtatás után:

15.4. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témát (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20G%C3%B6del!/GIMP%20Scheme%

Tanulságok, tapasztalatok, magyarázat...

A most következő feladat során a Lisp programnyelvek egyik képviselőjével fogunk dolgozni, a Scheme nyelvvel. A Scheme programnyelv is a Lisp programnyelvek családjába tartozik. Az 1970-es évek közepén jelent meg de egyszerűsége miatt mind a mai találkozhatunk Scheme-kódokkal.

A most következő feladat során a Scheme egyik dialektusát, a Script-Fu-t fogjuk használni arra, hogy az előző feladat során már megismert GIMP képszerkesztő programhoz egy olyan scriptet írjunk, ami lehetővé teszi a bemenetként megadott szöveg "króm effektezését". Ezen script megírásához használhatjuk a már előző feladat során látott Script-Fu-konzolt is.

Lássuk, hogy néz ez ki (.scm kiterjesztésű) kód formájában!

DRAFT

```
(define (color-curve)
(let* (
    (tomb (cons-array 8 'byte))
)
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
      tomb)
  )
```

Definiáljuk a "color-curve " függvényünket amivel egy tömböt töltünk fel 8 különböző értékkel. Ez a függvény még visszaköszön majd a feladat során a 9. lépésnél.

```
(define (elem x lista)

(if (= x 1) (car lista) (elem (- x 1) ( cdr lista ) ) )

)

(define (text-wh text font fontsize)
(let*
    (
        (text-width 1)
        (text-height 1)
    )

    (set! text-width (car (gimp-text-get-extents-fontname text ↩
       fontsize PIXELS font)))
    (set! text-height (elem 2  (gimp-text-get-extents-fontname ↩
       text fontsize PIXELS font)))

    (list text-width text-height)
    )
)
```

Ebben a részben két függvény is definiálásra kerül. Az első, "elem " függvény egy paraméterként megadott listából szintén paraméterként megadott sorszámú elem értékét adja vissza. Ez a függvény felhasználásra is kerül a következő "text-wh " függvény során is, amely függvénynek a visszatérési értéke egy lista melyben a font szélessége és magassága található meg.

```
(define (script-fu-bhax-chrome text font fontsize width height ↩
   color gradient)
    (let*
    (
        (image (car (gimp-image-new width height 0)))
```

```
                (layer (car (gimp-layer-new image width height RGB-IMAGE  ↩
                   "bg" 100 LAYER-MODE-NORMAL-LEGACY)))
                (textfs)
                (text-width (car (text-wh text font fontsize)))
                (text-height (elem 2 (text-wh text font fontsize)))
                (layer2)
            )
```

Az első sorban definiálásra kerül a scriptünk, ahol az látható, hogy a script nevét szóközzel elválasztva követik a paraméterek.

Ezt követi a "let* " függvény ami két részből áll, ezekből az elsőt tárgyaljuk ki ebben a bekezdésben. Ez az ún. "varlist" rész, itt azok a változók kerülnek deklarálásra, amelyeket a későbbiekben sokat fogunk használni. A több helyen is használt "car " függvény a GIMP eljárások és függvények által visszaadott listájának első elemét adja vissza. Természetesen bármilyen megkapott lista első elemét is visszaadja, nem csupán a GIMP-eseket...

```
        ;step 1
        (gimp-image-insert-layer image layer 0 0)
        (gimp-context-set-foreground '(0 0 0))
        (gimp-drawable-fill layer  FILL-FOREGROUND )
        (gimp-context-set-foreground '(255 255 255))
```

Első lépésként az effektelni kívánt szöveget fehér betűszínnel fekete háttérre kell írnunk.

A megoldáshoz a GIMP eljárásböngészőjét hívjuk segítségül, ahol a keresőbe bepötyögve a megfelelő függvény/eljárás nevét, minden információt megkapunk róla.

A "gimp-image-insert-layer " egy új réteget ad hozzá képünkhoz, első paramétere a kép maga, második a réteg, harmadik a szülő-réteg, ami jelen esetünkben nincs (ezért 0), az utolsó paraméter a pozíciója a rétegnek, ami jelen esetben 0, hiszen a legfelső rétegnek szeretnénk.

A "gimp-context-set-foreground " eljárással az előtérszínt RGB(0 0 0)-ra, vagyis feketére állítjuk. A "gimp-drawable-fill layer " eljárássalrétegünket kitöltjük az előtérszínnel. Újra a "...set-foreground " eljárást használjuk, az előtérszínt most fehérre állítjuk.

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

        (set! layer (car(gimp-image-merge-down image textfs CLIP-TO- ↩
           BOTTOM-LAYER)))
```

A "set! "-tel a "gimp-text-layer-new " függvény által a megfelelő paraméterekkel elkészített, majd visszaadott réteget beállítjuk "textfs " névre. Ezt a réteget a következő sorban az "..image-insert-layer " eljárással beszúrjuk.

A "gimp-layer-set-offsets " eljárással a rétegünket középre pozícionáljuk. Az utolsó sorban a "gimp-image-merge-down " függvénnyel lefelé haladva összefésüljük a rétegeinket és ezt a "layer " változóban tároljuk.

Az első lépés után ez a képünk:

A második lépés: Gauss-elmosás alkalmazása

```
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)
```

Az eljárás paraméterei a következők: futási mód (esetünkben ez RUN-INTERACTIVE), a kép maga, "rajzolható input"(nekünk a réteg), az elmosódás pixelben mért sugara, elmosódás vízszintesen, elmosódás függőlegesen.

A lépés után ez a képünk:

A harmadik lépés: játék a színszintekkel

```
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 ↩
   TRUE)
```

Az első paraméter, hogy min végezze el a műveletet a függvény, nekünk természetesen a rétegünkön kell majd, hogy elvégezve legyen. Minden további paraméter az egyes színértékekkel kapcsolatos, az Eljárásböngészőből kikeresgélhetőek.

A lépéssel a képünk:

A negyedik lépés: Gauss-elmosás, újra

```
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)
```

A paraméterek megegyeznek a nemrég nézett Gauss-elmosást létrehozó eljárásával, egyedül az elmosás sugara kisebb most, mint az előbb volt, így ez most egy enyhébb elmosást eredményez, szinte nem is látható a különbség az előző képhez képest:

Az ötödik lépés: A fekete rész szín szerinti kijelölése majd a kijelölés invertálása

```
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)
```

A "gimp-image-select-color " eljárással történik a kijelölés. Paraméterei: melyik képen kerül majd alkalmazásra, a kijelöléshez használatos művelet, "rajzolható input" és a kijelölendő szín.

A "gimp-selection-invert " pedig az egyetlen paraméterként kapott képen megfordítja a kijelölést.

A hatodik lépés: "Lebegő átlátszó kijelölés létrehozása"

```
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE ↩
   "2" 100 LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)
```

Az első sorban a "set! "-tel "layer2-t " egy új rétegként állítjuk be, mely réteget a "gimp-layer-new " függvény állított elő a képen látható paraméterekkel. Ezek közül a 6. a legérdekesebb ami nem más, mint az opacitás (áttetszőség) amit 100-ra állítottunk be. A következő réteget beillesztjük a már előzőek során is látott "gimp-image-insert-layer " eljárással.

A hetedik lépés: "Az áttetszősség kitöltése átmenettel

```
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ↩
   GRADIENT-LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ ↩
       height 3)))
```

DRAFT

Az első eljárás beállítja az elsődleges átmenetet, a "gimp-edit-blend " eljárás pedig egy kezdő- és végpont között egy "keverést" hoz létre a paraméterek segítségével, melyekről bővebben az Eljárásböngészőben lehet olvasni.

Így állunk jelenleg:

A nyolcadik lépés: Buckaleképezés alkalmazása

```
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 ↩
   5 5 0 0 TRUE FALSE 2)
```

A "plug-in-bump-map "-pel buckaleképezést hozunk létre a layer2-n, bemenetként felhasználva másik rétegünket.

A kilencedik lépés: Színgörbékkel való játék

```
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))
```

A fémesebb hatás érdekében "gimp-curves-spline "-nal ügyeskedünk még és voilá, kész is vagyunk:

```
(script-fu-register "script-fu-bhax-chrome"
    "Chrome3"
    "Creates a chrome effect on a given text."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 19, 2019"
    ""
    SF-STRING       "Text"      "Bátf41 Haxor"
    SF-FONT         "Font"      "Sans"
    SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
    SF-VALUE        "Width"     "1000"
    SF-VALUE        "Height"    "1000"
    SF-COLOR        "Color"     '(255 0 0)
    SF-GRADIENT     "Gradient"  "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
    "<Image>/File/Create/BHAX"
)
```

A scriptet ha nem a script-konzolon keresztül szeretnénk beadni, hanem inkludálni szeretnénk, hogy a GIMPen belül könnyebben használható legyen, függvényt kell használnunk hozzá. Ezt teszi a "script-fu-register " függvény, amely definiálása során alapértelmezett értékeket is megadunk.

A kódunk zárásaként megírjuk az ún. "Menübe regisztráló függvényt". Ahogy az elnevezés is sugallja, ezen függvény hatására tudjuk majd kiválasztani grafikusan is, a menüből a scriptünket.

16. fejezet

Helló, !

16.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! https://github.com/nbatfai/future/tree/master/-cs/F6 Itt láthatjuk működésben az alapot: https://github.com/nbatfai/future/tree/master/cs/F6

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hello!/F6

Ehhez a feladathoz létre kell hoznunk egy ".css" végződésű stylesheet fájlt, ahol a webfejlesztéshez hasonlóan szerkeszthetjük a külső megjelenést. `ActivityEditor.java` fájlban hozzá kell adnunk egy sort, mellyel a stylesheet fájlokat adhatunk hozzá:

```
scene.getStylesheets().add("style.css");
```

A `style.css` fájl tartalma az alábbi kódcsipetben látható:

```
.root {
  -fx-accent: #70BC1D;
  -fx-base: #262525;
  -fx-font-size: 12pt;
}

.label{
    -fx-text-fill: #ffffff;
    -fx-background-color: #313131;
}

.separator *.line {
   -fx-background-color: #313131;
   -fx-text-fill: #ffffff;
}

.tree-cell {
   -fx-text-fill: #ffffff;
}
```

DRAFT

Ahol a programban módosítani kívánt részek ponttal kezdődnek, majd a programrész neve követi. A kapcsos zárójelek közé kerülnek a módosítandó adatok. A JavaFX-ben előre meg vannak adva, hogy milyen kulcszavakat kell előhívni bizonyos módosításokhoz. Erről részletesebben a https://docs.oracle.com/-javafx/2/api/javafx/scene/doc-files/cssref.html weboldalon lehet olvasni. A fenti kódcsipetben az -fx-accent segítségével adjuk meg, hogy milyen színű kiemelést kapjon a ráklikkelt elem. Az -fx-base pedig az alapszínt adja meg. Az -fx-font-size értelemszerűen a szöveg méretét jelenti, jelen esetben 12pt lesz és a szöveg színének beállításához a -fx-text-fill kell használnunk.

16.1. ábra. Az átalakított program

Az alap program ablaka az elindítás után nálam kicsiben és képernyő alsó sarkában jelent meg, ezért ahhoz, hogy megadjunk egy alap ablak méretet és hogy a program elindítása után a kijelző közepén jelenjen meg, az alébbi kódcsipetre kell módosítani a programunkat a `ActivityEditor.java` fájl végén:

```
javafx.geometry.Rectangle2D primaryScreenBounds = javafx.stage.Screen. ↩
   getPrimary().getVisualBounds();
stage.setX(primaryScreenBounds.getMinX() + primaryScreenBounds.getWidth());
stage.setY(primaryScreenBounds.getMinY() + primaryScreenBounds.getHeight()) ↩
   ;

stage.setWidth(600);
stage.setHeight(800);

stage.centerOnScreen();

stage.show();
```

A stage.setX() és a stage.setY() kódsorokkal kapjuk meg az aktuális kijelzőnk méretét. Az ablakunk méretét a stage.setWidth() és a stage.setHeight() függvények segítségével módosíthatjuk, jelen esetünkben most a szélesség 600 és a magasság pedig 800 lesz. Ahhoz, hogy az ablakunk középen jelenjen meg, a stage.centerOnScreen() automatikusan végzi el majd végül a stage.show() jeleníti meg számunkra a programot.

16.2. OOCWC Boost ASIO hálózatkezelése

carlexerl.ll

Mutassunk rá a scanf szerepére és használatára! carlexerl.ll

A scanf kulcsszó már ismerős lehet számunkra, mert az előző fejezetben találkozhattunk vele, most ennek működését mutatom be. Először is lássuk a kódcsipetet:

```
            while ( std::sscanf ( data+nn, "<OK %d %u %u %u>%n", &idd, &f,  ↩
               &t, &s, &n ) == 4 )
```

```
                {
                    nn += n;
                    gangsters.push_back ( Gangster {idd, f, t, s} );
                }
```

A kód while ciklussal kezdődik, azaz addig fog lefutni, amíg be nem olvastuk az összes adatot. A "std::sscanf()" függvény első paramétere a beolvasni kívánt adatot tartalmazza, a második pedig azt, hogy milyen adatot keresünk. Ebben az esetben "OK"-val kezdődjön, majd decimális egész számot várunk, továbbá 3 darab előjel nélküli egészet (%u). Az %n a legutóbb beolvasott bájtok számát jelenti, ezt minden egyes alkalommal hozzáadjuk és eltároljuk az "nn" változóban. Ha mind a négy adat megegyezik, akkor létrehozunk egy új gangster objektumot és eltároljuk a gangsters vektorban.

Tehát a sscanf() függvény sztringből formázott adatot olvas, lényegében megegyezik a scanf() függvénnyel, azzal a különbséggel, hogy az előbbi a bemenetét a buffer által megadott memóriaterületről veszi. Visszatérési értéke a sikeresen beolvasott és tárolt mezők száma.

16.3. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! https://github.com/nbatfai/robocar- emulator/blob/master/-justine/rcemu/src/carlexer.ll

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hello!/OOCWC

https://github.com/nbatfai/robocar-emulator

Tanulságok, tapasztalatok, magyarázat...

A Robocar World Championship (OOCWC = rObOCar World Championship) egy olyan projekt/platform mely lehetőséget ad/adott a robotautó-kutatásra. Az egész projekt középpontjában a Robocar City Emulator áll. A következőkben a "carlexer.ll"-ben található "scanf" függvény jelentőségével és használatával fogunk foglalkozni.

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

Forrás: http://www.cplusplus.com/reference/cstdio/sscanf/

A "sscanf" a sima "scanf" függvénnyel ellentétben formázott stringből olvas be adatokat. Alapvetően két részből állnak: egy úgynevezett "buffer"ből és egy "format"ból. A "buffer" egy pointer lesz egy karakterstringre, amiből kiolvasásra kerül majd az adat. A "format" határozza meg, az adatok hogyan kerüljenek olvasásra. Úgynevezett format-specifikátorokból áll, ezek rendre százalékjellel kezdődnek.

Lássuk a mi esetünkben ez hogy néz ki! Azt azonban fontos megemlíteni, hogy forrásunkban a "sscanf" függvénynek több előfordulása van, nem csak az, amit a következőkben meg fogunk tekinteni, azonban lényegében mind egy kaptafára épülnek, ezért úgy gondolom, elég egyet bemutatni közülük.

DRAFT

Példánkban ilyennel találkozhatunk: <%d %u %u %u>%n. A "d" az integerekre illeszkedik, míg a "u" az unsigned integerre, vagyis az előjel nélküli integerekre. Az utolsó, "n" pedig arra fog szolgálni, hogy számon tartsa a már beolvasott karakterek számát. Az előbb mintáknak megfelelő adatok rendre a <& jellel bevezetett változókban kerülnek tárolásra.

Az beolvasandó adatok addig kerülnek feldolgozásra, míg nincs meg mind a 4 várt argumentum. (ezt láthatjuk a while ciklusfej végén- == 4).

Abban az esetben ha mind a 4 várt argumentumot sikeresen be tudtuk olvasni, az az számunkra azt jelenti, a gengsztereket jellemző mind a 4 tulajdonság meg lett adva, így ez esetben egy új Gangster kerül létrehozásra a megfelelő argumentumokkal és a "gangsters" vektorba tároljuk az "elkészült" gengszterünket.. Az "nn" változóban pedig tárolásra kerül az összesen beolvasott karakterek száma.

Látható, hogy az "nn" változó a "sscanf" függvény buffer részében is megtalálható. Amint mondtam, az nem más, mint egy pointer. A "data" értékét azért növeljük minden egyes alkalommal "nn" értékével, hogy onnan olvassunk ki további adatokat, ahonnan még nem tettük azt, ezzel a még kiolvasatlan rész elejére ugrunk mindig.

16.4. SamuCam

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

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hello!/SamuCam

Tanulságok, tapasztalatok, magyarázat...

Mivel a feladat kifejezetten a webcam kezelését kéri tőlünk, egyértelmű, hogy a "SamuCam.cpp" állományunk lesz az, amivel foglalkoznunk kell. Kezdjük is el, elemezzük ki a kódot!

```
#include "SamuCam.h"

SamuCam::SamuCam ( std::string videoStream, int width = 176, int  ↩
   height = 144 )
  : videoStream ( videoStream ), width ( width ), height ( height )
{
  openVideoStream();
}

SamuCam::~SamuCam ()
{
}

void SamuCam::openVideoStream()
{
  videoCapture.open ( 0 );

  videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
  videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
  videoCapture.set ( CV_CAP_PROP_FPS, 10 );
```

DRAFT

```
}
```

Mindenekelőtt inkludáljuk a header-fájlunkat, enélkül nehézen tudna kódunk normálisan működni. Megjelenik a konstruktorunk mely 3 paraméterrel rendelkezik. Ezek a következők: videoStream, width és height. A destruktorral is itt találkozhatunk. Az "openVideoStream" függvényen belül már kezdődnek az érdekességek. Fontos beszélni a "videoCapture.open" függvényről és paraméteréről is. Kérdés, hogy miért 0-t találunk ott.. Az eszköz / device indexet találhatjuk meg ott. Ezek az indexek 0-tól kezdődnek, a 0 az alapértelmezett kamera-eszközünk indexe. De akár IP cím is állhatna ott paraméterként.

Ugyanitt történik az előbb említett objektum finomhangolása, tehát a kamera képének szélessége és magassága valamint az FPS szám megadása.

```
void SamuCam::run()
{

  cv::CascadeClassifier faceClassifier;

  std::string faceXML = "lbpcascade_frontalface.xml"; // https:// ↩
     github.com/Itseez/opencv/tree/master/data/lbpcascades

  if ( !faceClassifier.load ( faceXML ) )
    {
      qDebug() << "error: cannot found" << faceXML.c_str();
      return;
    }

  cv::Mat frame;
```

Elérkeztünk a "run" függvényhez. Szükségünk van egy CascadeClassifier-re, amit alapesetben egy adott tárgy "detektálására" szoktak használni az adott videóstreamben, OpenCV-s programokban. Esetünkben ez a "faceClassifier" lesz, aminek a feladata az arc felismerése lesz. Ahhoz viszont, hogy ez működjön, a megjegyzésben és a github repóban is található linkre kattintva le kell töltenünk egy xml fájlt, aminek tartalmát a "faceXML" stringben tárolunk majd el. Ez tartalmazza majd ténylegesen az arc felismeréséhez használatos információkat. A "load" függvénnyel működésre bírjuk az xml-t, természetesen a fájl hiányában hibaüzenettel térünk vissza.

```
while ( videoCapture.isOpened() )
{

  QThread::msleep ( 50 );
  while ( videoCapture.read ( frame ) )
    {

      if ( !frame.empty() )
        {

          cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0,  ↩
             cv::INTER_CUBIC );

          std::vector<cv::Rect> faces;
```

```
                cv::Mat grayFrame;

                cv::cvtColor ( frame, grayFrame, cv::COLOR_BGR2GRAY );
                cv::equalizeHist ( grayFrame, grayFrame );

                faceClassifier.detectMultiScale ( grayFrame, faces, 1.1,  ↩
                   3, 0, cv::Size ( 60, 60 ) );

                if ( faces.size() > 0 )
                  {

                    cv::Mat onlyFace = frame ( faces[0] ).clone();

                    QImage* face = new QImage ( onlyFace.data,
                                                onlyFace.cols,
                                                onlyFace.rows,
                                                onlyFace.step,
                                                QImage::Format_RGB888 );

                    cv::Point x ( faces[0].x-1, faces[0].y-1 );
                    cv::Point y ( faces[0].x + faces[0].width+2, faces ↩
                       [0].y + faces[0].height+2 );
                    cv::rectangle ( frame, x, y, cv::Scalar ( 240, 230,  ↩
                       200 ) );


                    emit  faceChanged ( face );
                  }

                QImage*  webcam = new QImage ( frame.data,
                                               frame.cols,
                                               frame.rows,
                                               frame.step,
                                               QImage::Format_RGB888 );

                emit  webcamChanged ( webcam );

              }

            QThread::msleep ( 80 );

          }
```

Egy "while" függvényen belül történik a "csoda"..50 ms-onként ellenőrzi, megvan-e még nyitva kameránk. Ha meg, abban az esetben, ha a vannak képkockáink, azok a frame-ről beolvasásra majd tárolásra is kerülnek.

A "resize" függvény segítségével átméretezésre kerül a kép. Létrehozunk egy "faces" vektort, majd a képet a "cvtColor" segítségével szürkévé alakítjuk. Ezen képpontok számára is létrehozunk egy tárolót "grayFrame" néven.

Ezt követően a "detectMultiScale" függvényt felhasználva arcokat keresünk, a megtalált arcok pedig egy "rectangle" (téglalap) kerülnek tárolásra. Az arcból "QImage" objektum készül, majd egy "webcam" névre hallgató QImage objektum is készül, mindkettő 5 paraméterrel.

Mindkét esetben találkozhatunk az "emit" függvénnyel, ami nem más, mint egy makró. A következő feladatban foglalkozunk majd részletesebben a Qt slot-signal mechanizmusával, egyelőre annyit elég tudni az "emit"ről, hogy slotok és signalok összeegyeztetésénél van szerepe.

Az előbb olvasható blokkban található tartalom pedig rendre 80 ms-enként ismétlődik majd.

```
        if ( ! videoCapture.isOpened() )
        {
          openVideoStream();
        }
```

Megvizsgáljuk azt is, megvan-e nyitva kameránk. Ellenkező esetben megteszi azt.

Fordítjuk és futtatjuk a kódot, és voilá! Látható, hogy sikeres volt az arcfelismerés!

16.5. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: https://github.com/nbatfai/esport-talent-search

Megoldás videó:

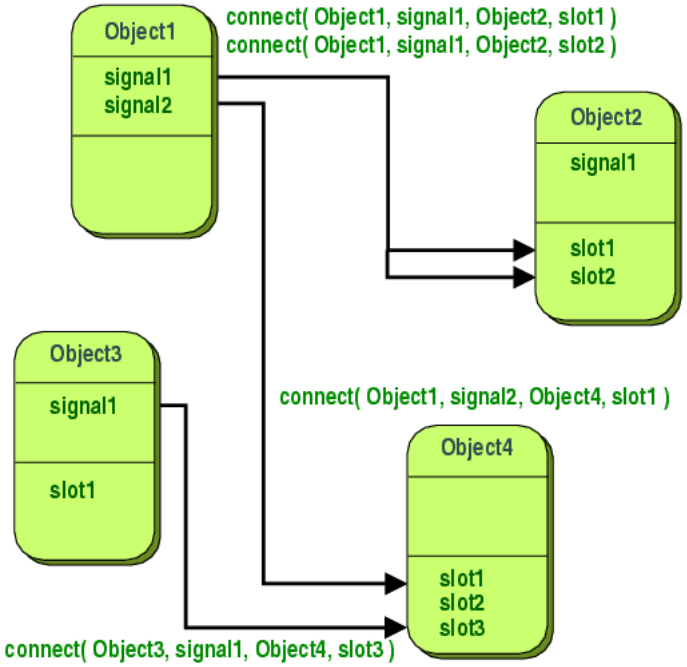
Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hello!/BrainB

Tanulságok, tapasztalatok, magyarázat...

A BrainB projekttel már előző félévben, Magas szintű programozási nyelvek 1 tárgyunk során volt szerencsém találkozni. A program célja nem más, mint hogy segítse a jövő kiemelkedő esportolóinak megtalálását. A program ebben úgy segít, hogy jövő kiemelkedő esportolóinak fellelése. A program az agy kognitív képességét méri.

A játékos feladata a Samu Entropy nevű karakteren tartani az egérmutatót ami az idő teltével természetesen egyre nehezebb hiszen újabb és újabb Entropy karakterek jelennek meg a képernyőn. A program érzékeli, ha Samu Entropyt elvesztettük, ekkor csökkenti a többi Entropy számát. A kódok elemzése előző féléves Prog. 1-es tanulmányaink során már megtörtént ezért most arra külön nem térnék ki. Viszont akkor még nem tértünk ki a Qt slot-signal mechanizmusára, most pótoljuk be ezt és nézzük, mi is az!

DRAFT



A kép forrása: https://doc.qt.io/qt-5/signalsandslots.html

Mit látunk az ábrán? Signal-okat és slot-okat. A feladat megértéséhez tisztáznunk kell a fogalmakat. A signal-ok az objektumok által kerülnek "kisugározva", lényegében jelként kiküldve többek között akkor ha az adott objektum belső állapota valami oknál fogva megváltozott.

Ezekhez a signal-okhoz vannak kapcsolva a slot-ok, amik akkor kerülnek meghívásra ha a hozzájuk tartozó signal kiküldésre kerül. Ezek a slot-ok egyébként egyszerű C++ függvények, hívásuk viszont nem úgy történik, mint a hétköznapi függvényeknek. Az "emit" kulcsszóval bocsáthatók ki. Erre láthatunk példát akár a BrainBThread.cpp forrásunkban is:

```
        emit endAndStats ( endTime );
```

A slot-okat a signal-okkal minden esetben egy "connect" függvénnyel kapcsolhatjuk egymáshoz. Erre programunkban két ízben láthatunk példát a "BrainBWin.cpp" fájlban.

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ),
          this, SLOT ( updateHeroes ( QImage, int, int ) ) );

connect ( brainBThread, SIGNAL ( endAndStats ( int ) ),
          this, SLOT ( endAndStats ( int ) ) );
```

Hogyan is épül fel ez a "connect" függvény, mik az egyes paraméterek? Formailag a következőképp néz ki: "connect(első_objektum, signal, második_objektum, slot)". Az első objektum küldi magát a signalt, míg a második feladata kezelni azt.

Jogosan merül fel a kérdés, megtörténhet-e az, hogy egy signal-hoz több slot is tartozzon? Természetesen ez is lehetséges, ekkor a slot-ok egymás után sorban kerülnek végrehajtásra. Sőt, akár fordítva is történhet a dolog, több signal is kapcsolható akár egyetlen slothoz. Ha nagyon el akarunk rugaszkodni a tipikus felhasználási módtól, akár két signal egymáshoz kapcsolása is megoldható.

Fontos megemlíteni, hogy a mechanizmus típus biztos, vagyis a signal és a slot szignatúrájának meg kell egyeznie.

Vessünk egy pillantást a mi példánkra ismét!

DRAFT

```
connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ),
          this, SLOT ( updateHeroes ( QImage, int, int ) ) );

connect ( brainBThread, SIGNAL ( endAndStats ( int ) ),
          this, SLOT ( endAndStats ( int ) ) );
```

Láthatjuk, hogy a slot-ok és a signal-ok paramétereinek száma és típusa vagyis szignatúrája megegyezik.

Az első connect értelmezése: ha a "brainBThread" objektumban a "heroesChanged" signál emittálódik, akkor az "updateHeroes" slotnak kell meghívódnia.

A hívott függvény:

```
        void BrainBWin::updateHeroes ( const QImage &image, const int &x,
           const int &y )
{

        if ( start && !brainBThread->get_paused() ) {

                int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) +
                   ( this->mouse_y - y ) * ( this->mouse_y - y );

                if ( dist > 121 ) {
                        ++nofLost;
                        nofFound = 0;
                        if ( nofLost > 12 ) {

                                if ( state == found && firstLost ) {
                                        found2lost.push_back ( brainBThread
                                           ->get_bps() );
                                }

                                firstLost = true;

                                state = lost;
                                nofLost = 0;
                                //qDebug() << "LOST";
                                //double mean = brainBThread->meanLost();
                                //qDebug() << mean;

                                brainBThread->decComp();
                        }
                } else {
                        ++nofFound;
                        nofLost = 0;
                        if ( nofFound > 12 ) {

                                if ( state == lost && firstLost ) {
                                        lost2found.push_back ( brainBThread
                                           ->get_bps() );
                                }

                                state = found;
                                nofFound = 0;
```

```
                                    //qDebug() << "FOUND";
                                    //double mean = brainBThread->meanFound();
                                    //qDebug() << mean;

                                    brainBThread->incComp();
                            }

                    }

            }
            pixmap = QPixmap::fromImage ( image );
            update();
}
```

Ez azt eredményezi, hogy a hőseink helye megváltozik a képernyőn.

A második connect értelmezése: ha a "brainBThread" objektumban az "endAndStats" signál emittálódik, akkor az ugyanilyen nevű "endAndStats" slotnak kell meghívódnia.

Meghívásra került:

```
        void BrainBWin::endAndStats ( const int &t )
        {

                qDebug()  << "\n\n\n";
                qDebug()  << "Thank you for using " + appName;
                qDebug()  << "The result can be found in the directory " + ↩
                   statDir;
                qDebug()  << "\n\n\n";

                save ( t );
                close();
        }
```

Az "endAndStats" függvény lényegében "búcsúüzenetet" dob a felhasználónak, elmenti eredményeinket egy fájlba és kilép.

Fordítás és futtatás után már használható is:


Press and hold the mouse button on the center of Samu Entropy
0:4/10:0 6660 bps
Samu Entropy
Norbi Entropy
Nandi Entropy
Matyi Entropy

17. fejezet

Helló, Lauda!

17.1. Junit teszt

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hallo%2C%20Lauda!/junit

A progprater poszt kézzel számított mélységét és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

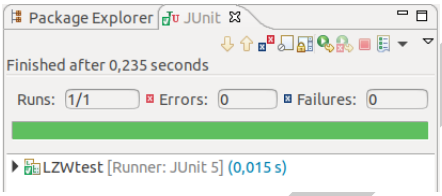
A Junit egy egységteszt keretrendszer Java nyelvhez. Ezzel kódot tesztelő osztályokat írhatunk és futtatásával leellenőrízhetjük, hogy a programunk működése a várt módon hajtódik-e végre. Ehhez előbb létre kell hoznunk egy új projektet az Eclipse szerkesztőben, majd a néhány fejezettel ezelőtti LZWBinFa.java programunkat kell bemásolni. Ha ezzel megvagyunk, akkor létrehozhatunk egy JUnit tesztelőt úgy, hogy a projektunk gyökérkönyvtárra a jobb egérrel klikkelünk, majd "new > JUnit Test Case" menüpontot válasszuk ki. Egy osztálynév megadása után klikkeljünk a finishre, majd gépekjük be a következő kódot:

```
package Binfa;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
class LZWtest {
  LZWBinFa testBinfa = new LZWBinFa();

    @Test
    public void testOperator() {
      String input = "01111001001001000111";

      for(char b: input.toCharArray()){
            testBinfa.operator(b);
        }

        assertEquals(4, testBinfa.getMelyseg());
        assertEquals(2.75, testBinfa.getAtlag(), 0.001);
        assertEquals(0.957427, testBinfa.getSzoras(), 0.000001);
    }
}
```

DRAFT

Ebből nekünk a "@Test" az ami újdonság. Ezzel jelezzük a programunk számára, hogy tesztelő függvény következik. Megadunk egy input-ot, jelen esetben a feladatban található számsort, amit toCharArray() segítségével egyenként egy tömbbe esszük az egyeseket és a nullákat. Ezután elvégzi a számításokat az LWWBinFa.java programunkban ismert módon. Végül a kapott eredményeket az assertEquals() függvénnyel ellenőrízzük le. Ennek elsőparamétere a számunkra várt eredmény, második paramétere a kapott eredmény és végül a harmadik paraméterként azt kell megadni, hogy lehet-e deltányi eltérés.

Ha ezzel megvagyunk, akkor elindíthatjuk a tesztelő programunkat úgy, hogy a jobb egérrel a következő menüpontokra klikkelünk: Debug as > JUnit Test. Ha zöld csík jelenik meg, akkor mindent jól csináltunk és azt az eredményt kaptuk, amit kézzel is kiszámoltunk:



17.1. ábra. JUnit teszt

17.2. Android Játék

Írjunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Helló, Android!” feladatára!

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hallo%2C%20Lauda!/Android

Ebben a feladatban egy Tic Tac Toe nevű játékot fogunk felépíteni, amely egy olyan kétszemélyes stratégiai játék, ahol egy 3x3 mezőből álló táblára "X" vagy "O" jeleket teszünk. Az nyer, akinek sikerül egy vonalban 3 jelet elhelyeznie, vízszintes, függőleges vagy átlós irányba.

Mivel ez egy Androidos játk lesz, ezért célszerű letölteni az Android Studio-t. Kezdésként hozzunk létre egy új prpjektet, aminek adjunk egy nevet. Én azAndroid 4.4-es verziót választottam ki, mivel úgy tűnik jelenleg ezzel a verzióban létrehozott appok az eszközök több, mint 90%-ában el fog futni. Ezután az "Empty Activity"-t kell majd kiválasztanunk. Egy kis idő elteltével az azblakon 2 fület fogunk látni: `activity_main.xml` és a `MainActivity.java`. Ezeket fogjuk majd szerkesztgetni. Először az alkalmazásunk külsejét fogjuk kódolni, amit az `activity_main.xml` fájlban tehetünk meg:

```
...
    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:id="@+id/text_view_p1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Player 1: 0"
            android:textSize="30sp"
            android:freezesText="true"/>
        <TextView
            android:id="@+id/text_view_p2"
```

DRAFT

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_below="@+id/text_view_p1"
            android:text="Player 2: 0"
            android:textSize="30sp"
            android:freezesText="true"/>
        <Button
            android:id="@+id/button_reset"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentEnd="true"
            android:layout_centerVertical="true"
            android:layout_marginEnd="33dp"
            android:text="reset" />
    </RelativeLayout>
...
```

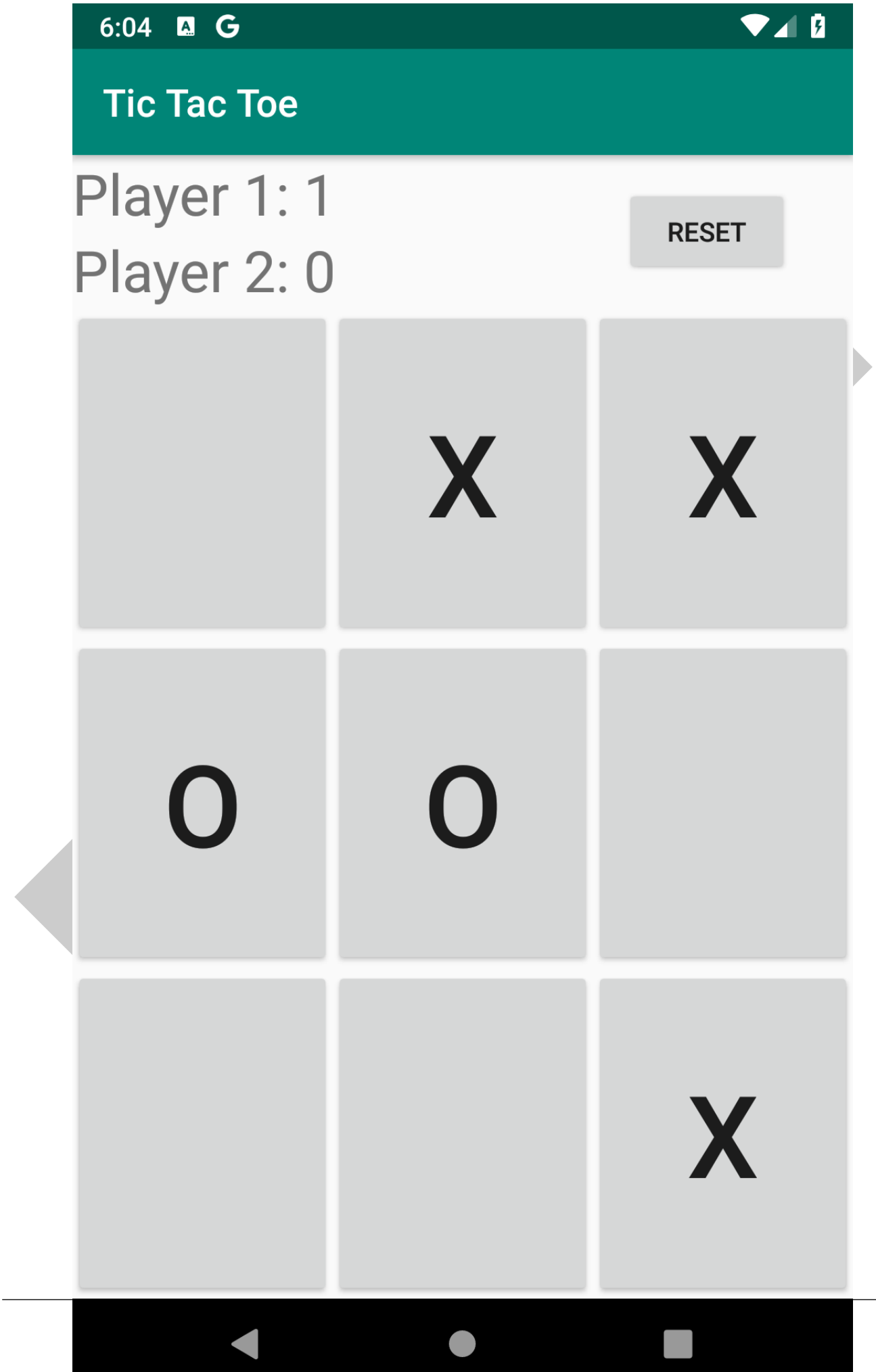
A fenti kódcsipetben a "Player 1: 0" és a "Player 2: 0" szövegeket fogjuk megjeleníteni, továbbá lesz egy reset gombunk is. Az egyes elemek szélességét és magasságát a android:layout_width és a android:layout_height segítségével tehetünk meg. Például a szövegünk, ami a TextView tag-ekben találhatóak, a szélessége meg fog egyezni az eszközünk méretével (match_parent) és a magasság pedig automatikus lesz (wrap_content). Mivel megadtunk egy szövegméretet is, ezért ezzel megegyező lesz a magasságunk: android:textSize="30sp".

```
...
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1">
        <Button
            android:id="@+id/button_00"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:textSize="60sp"
            android:freezesText="true"/>
        <Button
            android:id="@+id/button_01"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:textSize="60sp"
            android:freezesText="true"/>
        <Button
            android:id="@+id/button_02"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:textSize="60sp"
```

```
            android:freezesText="true"/>
    </LinearLayout>
...
```

LinearLayout tag-ben adjuk meg a mezőnket. Mivel 3x3-as mátrixot csinálunk, ezért a fájlban három LinearLayout-ot fogunk létrehozni, amikben 3-3 Button lesz. Az első LinearLayout egy sorból és 3 darab teljes magasságú oszlopkból fog állni. Ha hozzáadunk még egy ilyen LinearLayout-ot, akkor viszont már 2 sor és 3 oszlop lesz látható és így tovább. Ezekez a gombokhoz megadunk egy id-t is, például az eső gombunk id-je: android:id="@+id/button_00". Értelemszerűen a következő gombunk button_01 lesz és ügyeljünk rá, hogy ne legyen megegyező azaonosítójuk az egyes gomboknak.

DRAFT


6:04
Tic Tac Toe
Player 1: 1
Player 2: 0
RESET
X
X
O
O
X

17.3. Port scan

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

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hallo%2C%20Lauda!/PortScan

A kivételkezelés a szerepe, hogy a program futtatása során hiba esetén megszakítja a programot, majd kidob egy hibaüzenetet. Ha egy metódus eldob egy kivételt, a futtató környezetmegpróbál a kezelésére találni valamit. 3 blokkból állhat: try, catch és finally blokkokból. A try a kivételkezelő első lépésének eszköze. Ide kerül az a kódcsipet, ami hibát dobhat a program futtatása során. A catch blokkba kerül a hibaüzenet kiíratása, a finally blokk pedig rendet tesz a programban, tehát például bezárja a nem használt fájlokat, amikre nem lesz már szükségünk. Most pedig nézzük meg a feladatban szereplő kódcsipetet:

```
public class PortScan {

    public static void main(String[] args) {

        for(int i=0; i<1024; ++i)

            try {

                java.net.Socket socket = new java.net.Socket(args[0], i);

                System.out.println(i + " figyeli");

                socket.close();

            } catch (Exception e) {

                System.out.println(i + " nem figyeli");

            }
    }
}
```

Ez a program annyit csinál, hogy minden 1024 alatti számú gép TCP kapuival megpróbál kapcsolatot létrehozni. Minden egyes kapcsolaton végigmegy és egyesével írja ki, hogy sikerült-e vagy sem. Siker esetén például kiíratjuk, hogy az adott számút figyeli, majd bezárjuk a kapcsolatot, azonban ha nem sikerül a kapcsolat, átlépünk a catch blokkba, ahol azt íratjuk ki, hogy hanyas számú portot nem figyeli a program.

17.4. EPAM: Kivételkezelés

Ha az input float, akkor egy ChildException-t dob, amit az első catch elkap, hisz a ChildException mindig a ChildException egy instance-ja lesz. Azért nem kapja el a parent-exceptiont, mert a catch block kidob a finally részig egészen, így a parentet nem tudjuk elkapni.

Ha az input string akkor egy ParentExceptiont fog dobni, amit nem más mint a második catch fog elkapni, mert a szülő osztály nem instance-ja a child osztálynak, bár ez fordítva igaz, de a a cath-ágak sorrendje miatt ez irreleváns itt.

Ha az Input NULL, akkor csak egy generikus "Runtime Exceptiont" dob, mert az a parent osztája mind a Parent és Child exceptionoknak, azért ugye az instanceof false-t returnöl, így csak a legutolsó catch kapja el.

DRAFT

18. fejezet

Helló, Calvin!

18.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel,

https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol Háttérként ezt vetítsük le:

https://prezi.com/0u8ncvvoabcr/no-programming-programming/

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Calvin!/mnist

Tanulságok, tapasztalatok, magyarázat...

Érdemes lenne azzal kezdeni, mi is az MNIST. Lényegében kézzel írott arab számjegyek adatbázisa, mely összesen 60000 darab 28x28 pixel méretű, greyscale PNG képállományt tartalmaz. Ez a 60000 állomány azonban alapvetően két részre bontható, hiszen 50000 darab tanítási és 10000 darab tesztelési képet tartalmaz. Előbbiek alapján tanulja meg a gép számjegyeket, majd a tanulásának eredményességét ellenőrzi 10 ezer képre nézve.

Lássuk a gépi tanulást megvalósító kódunkat!

```
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D,  ↩
   MaxPooling2D
from keras.models import Sequential
from keras.utils import to_categorical,np_utils
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
```

Kezdődik az egész azzal, hogy a programunk legelején importáljuk a megoldásunkhoz szükséges könyvtárakat. Számos alkalommal fordult elő, hogy ezen könyvtárak közül valamelyik nem volt megtalálható a gépemen. Ezeket mindig egy `pip install 'library_neve'` parancs segítségével orvosolni tudjuk. Példának okáért ha a matplotlib könyvtárral még nem rendelkezünk, akkor: `pip install matplotlib`.

DRAFT

Ennek említését azért tartottam fontosnak, mert nem egyszer jött velem szemben ez a probléma a feladat megoldása során.

```
(train_X,train_Y), (test_X,test_Y) = tf.keras.datasets.mnist. ↩
   load_data()
```

Az előző sor azért felel, hogy tudjunk dolgozni az adatbázisunkkal, itt töltjük be a `load.data()` függvény segítségével. Most már rendelkezésünkre áll a több tízezer képállomány, mellyel dolgozni tudunk majd. Ezek vektorokban kerülnek tárolásra.

```
train_X = train_X.reshape(-1, 28,28, 1)
test_X = test_X.reshape(-1, 28,28, 1)
```

Elkészítjük a tanításra és tesztelésre használatos számok tárolására alkalmas vektorokat, melyeket a `reshape` függvény segítségével számunkra és a feladatnak megfelelő formájúra hozunk. Ez esetünkben azt jelenti, hogy 28 db. , 28 db. elemet tartalmazó vektorra bontjuk adatainkat. Az első paraméter -1, vagyis ezt minden tagra eljátsszuk. Ha más szám állna ott, például 5, akkor nyilván csak a vektor első 5 tagja került volna ilyen formában átalakításra.

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255
```

A vektorok típusát átállítjuk, majd osztjuk a látható számértékkel azért, hogy a képeket alkotó képpontok értéke megfelelő legyen a lehető leggyorsabb tanítási folyamathoz.

```
train_Y_one_hot = to_categorical(train_Y,10)
test_Y_one_hot = to_categorical(test_Y,10)
model = Sequential()
```

Ebben a pár sorban megjelenik az úgynevezett `one hot` encoding is. Ez azért lényeges, mert a modell nem tud kategorikus adatokkal (a `to_categorical` függvénnyel kerül ilyen formájúra) dolgozni. Ezért 0-sok és 1-esek sorozatára kerül felbontásra az adott szám. Erről az encodingról több itt és itt olvasható. Ezen túl beállításra kerül a modellünk típusa is.

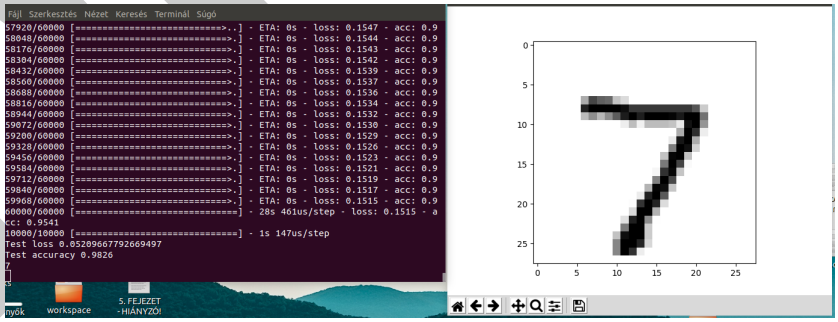
```
model.add(Conv2D(64, (3,3), input_shape=(28, 28, 1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(64))

model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, optimizer ↩
   =keras.optimizers.Adam(),metrics=['accuracy'])
```

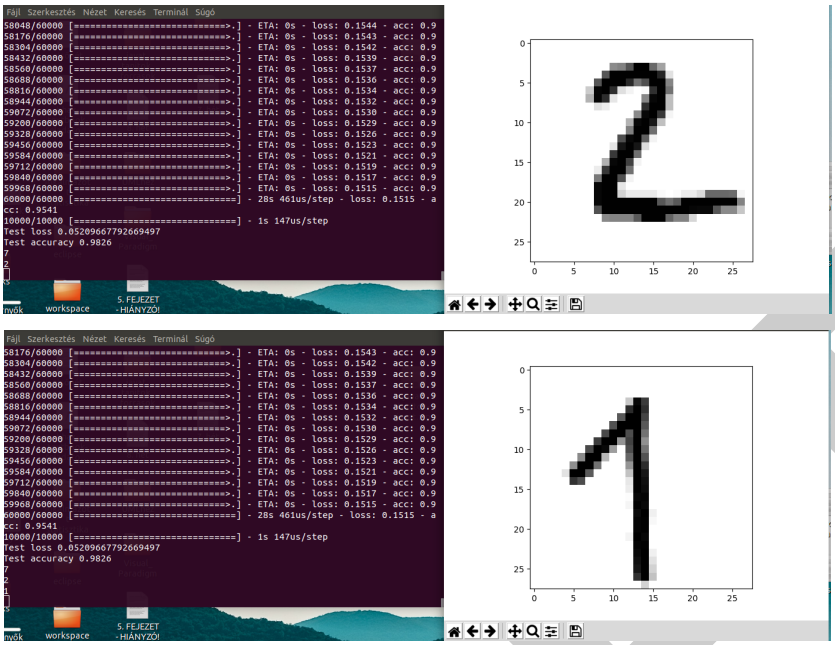
Rendre az `add` függvényt használjuk arra, hogy modellünkhöz újabb rétegeket adjunk hozzá. Láthatunk példát az első sorban arra, hogy milyen az, mikor egy konvolúciós réteget adunk hozzá a modellünkhöz. Amit erről tudni kell: első paramétere a neuronok száma, a második az úgynevezett detektor, a harmadik pedig az input shape, mely esetünkben a 28x28-as greyscale állomány lesz. Tehát egyértelműen egy úgynevezett konvolúciós neurális hálózatunk lesz (CNN angolul), amely a képanalitika területén igen népszerű modell. A következő sorban aktiválunk egy úgynevezett ReLU-t (Rectified Linear Unit), amit ha nagyon le szeretnénk fordítani magyarra, talán a "helyesbített lineáris egység" lenne a legmegfelelőbb. Az azt követő sorban a `pool_size`-zal azt adjuk meg, mennyi adat kerüljön feldolgozásra, ennek két argumentuma van, egy függőleges és egy vízszintes érték. A `compile` függvénnyel megindul a tanítási folyamat.

```
        model.fit(train_X, train_Y_one_hot, batch_size=64, epochs=1)

        test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
        print('Test loss', test_loss)
        print('Test accuracy', test_acc)

        predictions = model.predict(test_X)

        print(np.argmax(np.round(predictions[0])))
        plt.imshow(test_X[0].reshape(28, 28), cmap = plt.cm.binary)
        plt.show()
        print(np.argmax(np.round(predictions[1])))
        plt.imshow(test_X[1].reshape(28, 28), cmap = plt.cm.binary)
        plt.show()
        img = Image.open('one.png').convert("L")
        img = np.resize(img,(28,28,1))
```

A `fit` függvénnyel elkezdjük beállítani a tanítás jellemzőit. Itt inkább az utolsó két paraméter érdekes. A `batch_size` a tanulási sebesség, míg az epochs az lesz, hányszor hajtsa végig a folyamatot.

Majd kiiratjuk a tanulási folyamat során való veszteséget majd a pontossági értéket is. Eltároljuk a prediction-öket majd megjelenítjük az első, majd második feldolgozott képállományunkat a gép általi prediction-nel egyetemben. A harmadik képet mi adjuk- és nyitjuk meg az `Image.opern` függvény segítségével. Mi jelen esetben egy 1-es számmal teszteltünk, lássuk, mire sikerült jutnia a modellünknek!





Látható, hogy igen jó százalékos pontossággal sikerült neki eltalálni a számokat. A kézzel írott 1-esünket is sikerült felismernie.

18.2. CIFAR-10 -deprecated

Az alap feladat megoldása, +saját fotót is ismerjen fel,

https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tf_tutorial_peldabol

Megoldás videó:

Megoldás forrása: https://github.com/grestemayster/prog2/tree/master/Hell%C3%B3%2C%20Calvin!/Cifar-10

Tanulságok, tapasztalatok, magyarázat...

A feladat hasonlít az első, MNIST-es feladatunkhoz. Az alepvető különbség az, hogy most számjegyek helyett tárgyakat, élőlényeket (összefoglalóan mondhatjuk azt, objektumokat) kell majd felismernie. Szintén 60 ezer darab képpel fog dolgozni, ugyanúgy 50 ezer tanítási és 10 ezer tesztelési képet tartalmaz. A képállományok azonban most már nem greyscale-esek, hanem RGB-sek lesznek illetve az előző méret helyett 32x32-esek. Alapvetően 10 féle dologról találhatók meg képek az adatbázisban, ezekre néhány példa: autó, szarvas vagy éppen madár.

Azonban ahogy arra már a feladat elején utaltam, nagyrészt egyezik a megoldás logikája az első feladatunkéval. Ezzel együtt jár az is, hogy a kód sem sokban tér el tőle. Ezért ezen feladat esetében is elemzésre kerül a kód, azonban az újdonságokat az előző kódhoz képest félkövér betűvel fogom jelezni. Lássuk tehát!

```
import keras
from keras.datasets import fashion_mnist
from keras.layers import Dense, Activation, Flatten, Conv2D,  ↩
   MaxPooling2D
from keras.models import Sequential
from keras.utils import to_categorical,np_utils
from PIL import Image
```

DRAFT

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import os
```

Ismét azzal kezdünk, hogy a programunk legelején importáljuk a megoldásunkhoz szükséges könyvtárakat. A hiányzó könyvtárakat még mindig egy `pip install 'library_neve'` parancs segítségével orvosolni tudjuk.

```
(train_X,train_Y), (test_X,test_Y) = cifar10.load_data()
```

Értelemszerű, most egy másik adatbázis képeivel dolgozunk majd, ezeket töltjük be.

```
train_X = train_X.reshape(-1,32,32,3)
test_X = test_X.reshape(-1, 32,32, 3)
```

Az első paramétert leszámítva minden változik az előzőhöz képest, a lényeg viszont ugyanaz. A második és harmadik 32 lesz, hiszen 32x32-es képállományokkal lesz dolgunk, a negyedik paraméterként megadott 3-as pedig arra utal, RGB képekkel fogunk dolgozni.

```
train_X = train_X.astype('float32')
test_X = test_X.astype('float32')
train_X = train_X / 255
test_X = test_X / 255
```

A vektorok típusát átállítjuk, majd osztjuk a látható számértékkel azért, hogy a képeket alkotó képpontok értéke megfelelő legyen a lehető leggyorsabb tanítási folyamathoz.

```
train_Y_one_hot = to_categorical(train_Y,10)
test_Y_one_hot = to_categorical(test_Y,10)
model = Sequential()
```

Ebben a pár sorban megjelenik az úgynevezett `one hot` encoding is. Ez azért lényeges, mert a modell nem tud kategorikus adatokkal (a `to_categorical` függvénnyel kerül ilyen formájúra) dolgozni. Ezért 0-sok és 1-esek sorozatára kerül felbontásra az adott szám. Erről az encodingról több itt és itt olvasható. Ezen túl beállításra kerül a modellünk típusa is, ami szekvenciális lesz.

```
model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(32, ↩
   32, 3)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))

model.add(Dense(10))
model.add(Activation('softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss=keras.losses.categorical_crossentropy, optimizer ↩
   =sgd,metrics=['accuracy'])
```

```
img = Image.open('allat.png').convert("L")
img = np.resize(img,(32,32,3))
im2arr = np.array(img)
im2arr = im2arr.reshape(1,32,32,3)
```

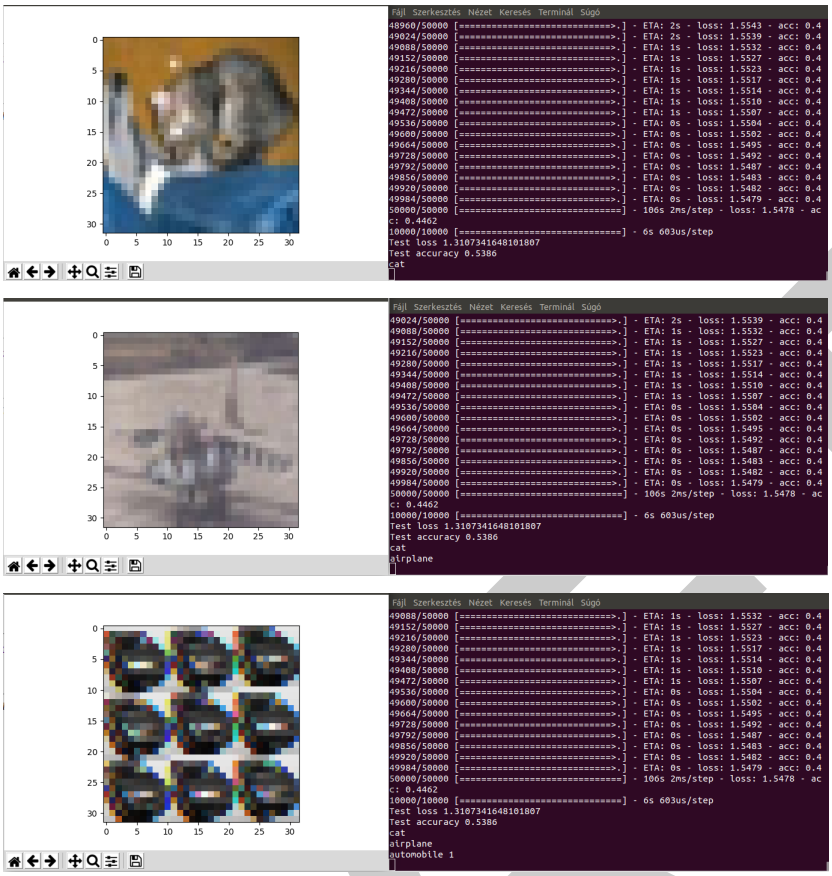
Rendre az `add` függvényt használjuk arra, hogy modellünkhöz újabb rétegeket adjunk hozzá. Láthatunk példát az első sorban arra, hogy milyen az, mikor egy konvolúciós réteget adunk hozzá a modellünkhöz. Amit erről tudni kell: első paramétere a neuronok száma, a második az úgynevezett detektor. **Mivel teljesen más jellemzőjű képállományokkal dolgozunk majd, értelemszerűen megváltoznak az `input_shape` függvény paraméterei is hiszen most már 32x32-es, ráadásul RGB-s képeket szeretnénk feldolgoztatni majd megtaníttatni. Továbbá több neuront használunk fel ez alkalommal.** A következő sorban ismételten aktiválunk egy úgynevezett ReLU-t (Rectified Linear Unit). Ezt követően a `pool_size`-zal megadjuk, mennyi adat kerüljön feldolgozásra, ennek két argumentuma van, egy függőleges és egy vízszintes érték. A `compile` függvénnyel megindul a tanítási folyamat.

```
model.fit(train_X, train_Y_one_hot, batch_size=64, epochs=1)

test_loss, test_acc = model.evaluate(test_X, test_Y_one_hot)
print('Test loss', test_loss)
print('Test accuracy', test_acc)

predictions = model.predict(test_X)
cifar_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', ' ↩
   dog', 'frog', 'horse', 'ship', 'truck']
print(cifar_classes[np.argmax(np.round(predictions[0]))])
plt.imshow(test_X[0].reshape(32, 32,-1), cmap = plt.cm.binary)
plt.show()
print(cifar_classes[np.argmax(np.round(predictions[526]))])
plt.imshow(test_X[526].reshape(32, 32,-1), cmap = plt.cm.binary)
plt.show()

print(cifar_classes[np.argmax(np.round(model.predict(im2arr)))],np. ↩
   argmax(np.round(model.predict(im2arr))))
plt.imshow(im2arr[0].reshape(32,32,3),cmap = plt.cm.binary)
plt.show()
```

A `fit` függvénnyel elkezdjük beállítani a tanítás jellemzőit. Itt inkább az utolsó két paraméter érdekes. A `batch_size` a tanulási sebesség, míg az epochs az lesz, hányszor hajtsa végig a folyamatot.

Majd kiiratjuk a tanulási folyamat során való veszteséget majd a pontossági értéket is. Eltároljuk a predictionöket. **Fontos megemlíteni az utolsó, egyik legszembetűnőbb különbséget és egyben érdekességet az MNIST feladathoz képest. Most meg kell adnunk kézzel egy tömböt, mely tartalmazza azokat a osztályokat, lényegében tárgyainkat, melyekről képeket találhatunk adatbázisunkban. Ez lesz a `cifar_classes` tömb.** Majd megjelenítjük az első, majd második feldolgozott képállományunkat a gép általi prediction-nel egyetemben. Utolsóként pedig az a kép kerül megjelenítésre amit mi adtunk be neki és remélhetőleg értelmezte, megtanulta. Lássuk, mire jutott:

Virtuális környezetben futtatjuk is:



Hát...látható, ha a számokat nézzük, nem a legpontosab...Fontos hozzátenni azért azt is, hogy ha többször futtattam volna le a tanítást, sokkal pontosabb lett volna a felismerés. Amit viszont elég impresszívnek tartottam az az, hogy még ilyen pontossági mutatók mellett is képes volt értelmezni és felismerni mindhárom képet, beleértve az általam letöltött autós képet is:

A letöltött kép forrása: https://m.blog.hu/pr/progpater/image/matchbox.png

18.3. EPAM: Back To The Future

Adott az alábbi kódrészlet:

```
        public class FutureChainingExercise {
private static ExecutorService executorService = Executors. ↩
   newFixedThreadPool(2);
public static void main(String[] args) {
CompletableFuture<String> longTask
= CompletableFuture.supplyAsync(() -> {
sleep(1000);
return "Hello";
```

DRAFT

```
}, executorService);
CompletableFuture<String> shortTask
= CompletableFuture.supplyAsync(() -> {
sleep(500);
return "Hi";
}, executorService);
CompletableFuture<String> mediumTask
= CompletableFuture.supplyAsync(() -> {
sleep(750);
return "Hey";
}, executorService);
CompletableFuture<String> result
= longTask.applyToEitherAsync(shortTask, String::toUpperCase,
   executorService);
result = result.thenApply(s -> s + " World");
CompletableFuture<Void> extraLongTask
= CompletableFuture.supplyAsync(() -> {
sleep(1500);
return null;
}, executorService);
result = result.thenCombineAsync(mediumTask, (s1, s2) -> s2 + ", " + s1,
   executorService);
System.out.println(result.getNow("Bye"));
sleep(1500);
System.out.println(result.getNow("Bye"));
result.runAfterBothAsync(extraLongTask, () -> System.out.println("After
   both!"), executorService);
result.whenCompleteAsync((s, throwable) -> System.out.println("Complete: "
   + s), executorService);
executorService.shutdown();
}
/** * * @param sleeptime sleep time in milliseconds */
private static void sleep(int sleeptime) {...}
}
```

Megoldás videó:

Megoldás forrása:

A kimenet:

Az ExecutorService egy olyan JDK-s keretrendszer, amely lehetővé teszi, hogy egyes taskok aszinkron módban fussanak. Ezért is kell példányosításkor megadni, hogy hány szálat szeretnénk létrehozni:

```
   public class FutureChainingExercise {
    private static ExecutorService executorService = Executors.
       newFixedThreadPool(2);
```

Létrehozunk különböző hosszúságú task-okat, amelyeknek a hosszúságát az fogja befolyásolni, hogy mennyi ideig altatjuk a szálat, de egyébként csak egy szimpla string-et founk visszatéríteni eredményként. A Future

interfész és ennek az implementáló osztálya, a CompletableFuture egy aszinkron módon végzett számítást/-feladatot tud reprezentálni.

```
CompletableFuture<String> longTask = CompletableFuture.supplyAsync(() -> ↩
    {
        try {
            sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "Hello";
    }, executorService);

    CompletableFuture<String> shortTask = CompletableFuture.supplyAsync ↩
       (() -> {
        try {
            sleep(500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "Hi";
    }, executorService);

    CompletableFuture<String> mediumTask = CompletableFuture. ↩
       supplyAsync(() -> {
        try {
            sleep(750);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return "Hey";
    }, executorService);

     CompletableFuture<Void> extraLongTask = CompletableFuture. ↩
        supplyAsync(() -> {
        try {
            sleep(1500);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        return null;
    }, executorService);
```

A taskokat a supplyAsync() metódussal adjuk meg, ahol a Supplier egy lambdafüggvény, azaz ez az elvégzendő feladat, a végrehajtó az osztályunk statikus executorService mezője.

```
    CompletableFuture<String> result = longTask.applyToEitherAsync( ↩
       shortTask, String::toUpperCase, executorService);
```

```
result = result.thenApply(s -> s + " World");
```

Az eredmény egy CompletableFuture objektum lesz. Az eredményt az applyToEitherAsync() metódus fogja meghatározni. A longTask illetve a shortTask közül azt választja, amelynek a befejezése a lehető leghamarabb megtörténik. Az eredményre alkamazni fogja a második paraméterként megadott toUpperCase() függvényt. Ezért a kimeneten a nagybetűs HELLO jelenik meg eredményként. Utána erre az eredményre még egy lambdafüggvénnyel hozzáfűzzük a "World" szót.

Eddig az eredményünk a "HELLO World".

```
result = result.thenCombineAsync(mediumTask, (s1, s2) -> s2 + ", " ↩
    + s1, executorService);
```

Ezután még egy mediumTask-al hozzáadunk egy "hey"-t az eredényhez, amellyet egy vesszővel választunk el a már meglévő eredménytől. Ez a metódus megvárja, hogy mindkét CompletionStage elvégezze a feladatát.

```
System.out.println(result.getNow("Bye"));
sleep(1500);
System.out.println(result.getNow("Bye"));
```

Utána a getNow() lekéréssel megpróbáljuk lekérni az eredményt.Mivel az eredényünk mégnem készült el, ezért csak egy "Bye" fog megjelenni az első sorban. 1500 milliszekundummal később megjelenik a második sorban az eredményünk, a "Hey, HELLO World".

```
result.runAfterBothAsync(extraLongTask, () -> System.out.println(" ↩
   After both!"), executorService);
result.whenCompleteAsync((s, throwable) -> System.out.println(" ↩
   Complete: " + s), executorService);
executorService.shutdown();
```

Az előbb említett két utsaításból az első a runAfterBothAsync() metódus miatt vár az extraLongTask lefutására, ezért nem jelenik meg a következő sorban az "After both!" A második utasítás le tud futni az executor shutdown-ja előtt, mivel már van eredménye a "result CompletionStage"-nek.

A végeredményünk pedig nem más mint a:

```
Bye

Hey, HELLO World

Complete:  Hey, HELLO World
```

IV. rész

Irodalomjegyzék

DRAFT

18.4. Általános

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

18.5. C

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

18.6. C++

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

18.7. Lisp

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

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

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

DRAFT