

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 16, v. 0.0.3

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY	Bátfai, Norbert	2019. december 2.

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	2019-02-25	Első csokor befejezése.	antalbalazs
0.0.5	2019-03-02	Második csokor befejezése.	antalbalazs
0.0.6	2019-03-13	Harmadik csokor befejezése.	antalbalazs
0.0.7	2019-03-19	Negyedik csokor befejezése.	antalbalazs
0.0.8	2019-03-27	Ötödik csokor befejezése.	antalbalazs

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-04-05	Hatodik csokor befejezése.	antalbalazs
0.0.10	2019-05-10	Olvasónapló befejezése	antalbalazs

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?	6
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	9
2.6. Helló, Google!	11
2.7. A Monty Hall probléma	12
3. Helló, Chomsky!	15
3.1. Decimálisból unárisba átváltó Turing gép	15
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	15
3.3. Hivatalos nyelv	15
3.4. Saját lexikális elemző	16
3.5. Leetspeak	17
3.6. A források olvasása	19
3.7. Logikus	20
3.8. Deklaráció	21

4. Helló, Caesar!	23
4.1. double ** háromszögmátrix	23
4.2. C EXOR titkosító	24
4.3. Java EXOR titkosító	24
4.4. C EXOR törő	25
4.5. Neurális OR, AND és EXOR kapu	25
4.6. Hiba-visszaterjesztéses perceptron	26
5. Helló, Mandelbrot!	27
5.1. A Mandelbrot halmaz	27
5.2. A Mandelbrot halmaz a std::complex osztállyal	29
5.3. Biomorfok	30
5.4. A Mandelbrot halmaz CUDA megvalósítása	32
5.5. Mandelbrot nagyító és utazó C++ nyelven	32
5.6. Mandelbrot nagyító és utazó Java nyelven	32
6. Helló, Welch!	33
6.1. Első osztályom	33
6.2. LZW	33
6.3. Fabejárás	37
6.4. Tag a gyökér	38
6.5. Mutató a gyökér	39
6.6. Mozgató szemantika	39
7. Helló, Conway!	41
7.1. Hangyszimulációk	41
7.2. Java életjáték	42
7.3. Qt C++ életjáték	42
7.4. BrainB Benchmark	43
III. Második felvonás	44
8. Helló, Berners-Lee!	46
8.1. Python: Bevezetés a mobilprogramozásba	46
8.2. Java: Java 2 útikalauz	47

9. Helló, Arroway!	49
9.1. OO szemlélet	49
9.2. "Gagyi"	52
9.3. Yoda	54
9.4. Kódolás from scratch	55
10. Helló, Liskov!	59
10.1. Liskov helyettesítés sértése	59
10.2. Szülő-gyerek	60
10.3. Anti OO	62
10.4. Ciklomatikus komplexitás	70
11. Helló, Mandelbrot!	72
11.1. Reverse engineering UML osztálydiagram	72
11.2. Forward engineering UML osztálydiagram	73
11.3. BPMN	81
12. Helló, Chomsky!	82
12.1. Encoding	82
12.2. Leet	84
12.3. Fullscreen	85
13. Helló, Stroustrup!	87
13.1. JDK osztályok	87
13.2. Másoló-mozgató szemantika és Összefoglaló	88
14. Helló, Gödel!	91
14.1. Gengszterek	91
14.2. Alternatív Tabella rendezése	92
14.3. GIMP Scheme hack	94
15. Helló, !	96
15.1. FUTURE tevékenység editor	96
15.2. Samu Cam	97
15.3. BrainB	100

16. Helló, Lauda!	103
16.1. Port scan	103
16.2. Junit Teszt	105
16.3. AOP	105
17. Helló, Lauda!	108
17.1. MNIST	108
17.2. Android telefonra a TF objektum detektálója	111
17.3. Deep MNIST	114
IV. Irodalomjegyzék	116
17.4. Általános	117
17.5. C	117
17.6. C++	117
17.7. Lisp	117

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

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 csemegeznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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: <https://gitlab.com/baluka94/bhax/blob/master/vegtelen>

Tanulságok, tapasztalatok, magyarázat...

C - ben végtelen ciklust 2 féle képpen lehet írni vagy for ciklussal vagy pedig while - al.

Az első megoldásban 1 magott kellett 100%-on futtatni, ehhez while ciklust használtam.

```
#include <stdio.h>
#include <unistd.h>
int main() {
while(1) {
}
}
```

A main függvény előtt meghívtam az unistd.h-t valamint az stdio.h-t ezekre szükség van hogy a program leforduljon és fusson.

A while ciklusban feltételként 1-et adtam, amely minden teljesül így a program addig fut amíg meg nem szakítom, ezáltal végtelen ciklust kialakítva. Ez alapból 1 magot fog 100%-os intenzitással használni.

A második része a feladatnak 1 magot 0% on futtatni, ehhez a while ciklusban elhelyeztem egy sleep függényt, amely a zárójelben megadott ideig "elaltatja" a futó szálat, így elérve a 0%-ot.

```
#include <stdio.h>
#include <unistd.h>
int main() {
while(1) {
sleep(100);      <-----
}
}
```

Végül az utolsó része minden magnak a 100%-on való futtatása.

```
#include <stdio.h>
#include <omp.h>
int main() {
#pragma omp parallel
#pragma omp while
while(1) {
}
}
```

Ennek elkészítéséhez az OpenMP-t használtam. A kód elején meghívtam az omp.h headert és az stdio.h-t. A main függvény elején meghívom az omp parallel valamint omp while pragmákat, amelyek lehetővé teszik a while ciklus több szálon való futtatását ezzel elérve, hogy minden szál 100%-os intenzitással fusson.

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

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

Megoldás videó:

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

```
Program T100
{
    boolean Lefagy(Program P)
    {
        if (P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra 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 nászlata nélkül!

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/valtozokcsereje>

Tanulságok, tapasztalatok, magyarázat... :

Elsőnek a 2 változó értékét összeadással, valamint kivonással próbáltam meg felcserélni. Ehhez deklaráltam két int(egész) típusú változót,a és b néven. Az a értékét 2-re állítottam a b értékét 3-ra.

```
int a=2;  
int b=3;
```

Ezeket ki is írtam a terminálba printf függvény segítségével, ami C-ben a képernyőre való kiíratást teszi lehetővé. A %d a decimális egész típusú változók kiíratásához kell.

```
printf(" A változók értéke %d, %d\n",a,b);
```

Ezután az a értékét a(2)+b(3) re módosítottam, így az aktuális értéke 5 lett. Majd a b értékét a(5)-b(3) re azaz 2-re végül megint az a értékét változtatom ezúttal a(5)-b(2) re vagyis 3-ra.

A felcserélt értékeket végül kiíratom.

```
a = a+b;  
b = a-b;  
a = a-b;  
printf(" A változók értéke felcserélve %d, %d\n",a,b);
```

Megoldva szorzással is:

```
int c = 5;  
int d = 10;  
printf(" A változók értéke %d, %d\n",c,d);  
c = c*d;  
d = c/d;  
c = c/d;  
printf(" A változók értéke felcserélve %d, %d\n",c,d);
```

2.4. Labdapattogás

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

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/labdapattog>

Tanulságok, tapasztalatok, magyarázat... .

Az első programban if-ekkel kell írnunk egy programot ami egy labdát pattogtat a terminálon.

A program elején 3 headert hívunk meg ezek a: <stdio.h>,<curses.h>,<unistd.h>.

Megkezdjük a main ciklust amelyben elhelyezzük a void-ot mivel ebben a programban nem lesz return érték.

Létrehozzuk az ablakot amelyen a labda pattogni fog. Felveszünk két int típusú változót egy x és egy y-t 0-ás kezdő értékkel ez a 0. sor és 0. oszlop lesz vagyis a tengelyek kezdő értékei. Az xnov és ynov változók azt adják meg, hogy hányat lépünk jobbra és le,fel vagy balra és le,fel, ezeket 1-1 re állítottam. Felveszünk egy mx és my változót, amelyek azt adják meg, hogy a labda meddig mehet el amíg el nem éri az ablak falát, ekkor irányt vált és halad tovább.

```
#include <stdio.h>
#include <curses.h>
#include <unistd.h>

int
main ( void )
{
    WINDOW *ablak;
    ablak = initscr ();

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;
```

Létrehozunk egy forciklust, ami egy végtelen ciklus lesz így érjük el a labda folyamatos pattogását. A cikluson belül a getmaxyx függvény segítségével lekérjük az ablak méretét. Az mvprintw függvénytel pedig kirajzolunk egy O betűt ami a labdánk lesz minden egyes sor-oszlop metszéspontjára. Ezután egy refresh függvényt hívunk ami folyamatosan és valós időben frissíti a kijelzőt. A usleep függvénnnyel azt adom meg, hogy milyen időközönként léptesse tovább a labdánkat. A 100000-es érték 0,1 másodpercnek felel meg ennyi időnként lépteti. Elkezdem léptetni az x és y-t majd ameddig az if függvényben megadott feltétel teljesül addig lépegetek ameddig el nem értem az ablak határát ekkor az irányváltást egy -1-el való szorzással érem el. 4 db if függvényre van szükség ahhoz hogy az ablak oldalainál a tetejénél és az aljánál irányt válson.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Ír 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: <https://gitlab.com/baluka94/bhax/blob/master/bogomips> Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... :

A BogoMips Linus találma. A kernelnek szüksége van egy időzítő ciklusra, amelyet a processzor sebessége alapján kell meghatározni. A MIPS a Million Instructions Per Second (Millió Utasítás Másodper-

cenként) rövidítése. Ez a számítógép számítási sebességének a mértékegysége. Ezért a kernel a rendszer-induláskor megméri, hogy egy bizonyos ciklus mennyi idő alatt fut le az adott számítógépen. A "Bogo" az angol "bogus" szóból ered, melynek jelentése "hamis", "nem igazi". Vagyis a BogoMIPS érték következtetni enged a processzor sebességére, de annyira áltudományos, hogy csak a BogoMIPS kifejezés illik rá.

```
#include <time.h>
#include <stdio.h>
void
delay (unsigned long long loops)
{
    for (unsigned long long i = 0; i < loops; i++);
}
int
main (void)
{
    unsigned long long loops_per_sec = 1;
    unsigned long long ticks;
    printf ("Calibrating delay loop..");
    fflush (stdout);
    while ((loops_per_sec <= 1))
    {
        ticks = clock ();
        delay (loops_per_sec);
        ticks = clock () - ticks;
        if (ticks >= CLOCKS_PER_SEC)
    {
        loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;
        printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
               (loops_per_sec / 5000) % 100);

        return 0;
    }
    }
    printf ("failed\n");
    return -1;
}
```

A time.h és stdio.h headereket hasdználjuk. A main függvényünk előtt létherozunk egy void-ot elhelyezzük benne a delay függvényt melyben felveszünk egy előjel nélküli long long típusú loops változót. A loops_per_sec értékét 1 re állítjuk. Ez vizuálisan 000...1 lenne. A while ciklusban elkezdjük ezt az 1-est léptetni addig amíg csupa nullát nem kapunk, ekkor kilép mivel már a feltétel nem teljesül. A ciklusban meghívjuk a clock függvényt ami meghatározza a processzor időt. Meghívjuk a delay függvényt aminek értékül adjuk a ciklusváltozót. Az elején lévő for ciklus elszámol 0-tól a határig(loops). Ezután lekérjük mennyi idő telt el a ciklus lefutása alatt. Megnézzük, hogy az óraütések száma mikor nagyobb mint a clock_per_sec(jelen esetben 1 millió ütés). A while ciklus addig vizsgálja a kettő hatványokat, amíg már a forciklus végrehajtásához szükség lesz 1 millió óra ütésre. Majd elosztjuk a loops_per_sec-et a tickes felszorozzuk clocks_per_sec-cel majd elosztjuk az arányokkal. A program azt próbálja meghatározni, hogy milyen ciklus érték kéne ahhoz hogy a loops_per_sec-et megkapjam.

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: <https://gitlab.com/baluka94/bhax/blob/master/pagerank> Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... :

A Page-Rank egy olyan a Google által létrehozott program amely a weboldalakat jóságuk szerint rangsorolja. Megnézi, hogy milyen weboldalról mennyi link mutat és annak szavazata mennyit ér. Minnél több és erősebb linkek mutatnak egy weboldalról annál jobb lesz az oldal.

```
#include <stdio.h>
#include <math.h>

void
kiir(double tomb[], int db)
{
for(int i=0;i<db;++i){
printf("pagerank %d: %lf\n",i,tomb[i]);
}
}

double
tavolsag (double PR[], double PRv[], int n)
{
double osszeg=0.0;
for (int i=0;i<n;++i){
osszeg+= (PRv[i] - PR[i]) * (PRv[i] - PR[i]);
return sqrt (osszeg);
}
}

int main(void)
{

double L[4][4] = {
{0.0, 0.0, 1.0 / 3.0, 0.0},
{1.0,1.0/2.0,1.0/3.0,1.0},
{0.0,1.0/2.0,0.0,0.0},
{0.0,0.0,1.0/3.0,0.0}
};
double PR[4] = {0.0,0.0,0.0,0.0};
double PRv[4] = {1.0/4.0,1.0/4.0,1.0/4.0,1.0/4.0};
int i, j;
for (;;)
{
for(int i=0;i<4;++i){
PR[i]=PRv[i];
}
for(int i=0;i<4;++i){
```

```
double seged=0.0;
for (int j=0; j<4; ++j) {
    seged+=L[i][j]*PR[j];
    PRv[i]=segd;
}
}
if (tavolsag (PR, PRv, 4) < 0.0000000001)
break;
}
kiir (PR, 4);
return 0;
}
```

Egy kiir nevű függvénynek paraméterül adunk egy double típusú tömböt és egy int db változót. A függvényben deklarálunk egy i változót, for ciklussal végigmegyünk az elemeken és kiíratjuk őket kimenetre. Egy távolság nevű függvénynek paraméterül adunk 2 tömböt és egy n változót, amiben a tömb hosszát tároljuk. Felbeszünk egy osszeg nevű változót, ami a függvény elvégzése után és ha lefutott a for visszaadja az osszeg négyzetgyökét.

2.7. 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... :

A Monty Hall probléma egy valószínűségi paradoxon, ami az USA-ban futott Let's Make a Deal című vetélkedő utolsó feladatán alapul, a műsorvezető Monty Hall után kapta a nevét. A feladat lényege, hogy a műsor végén a játékosoknak mutatnak 3 ajtót, amelyek közül 2 mögött 1-1 kecske van, ám a harmadik mögött egy vadonatúj autó. A játékos rámutat egy ajtóra, ám ezt nem nyitják ki rögtön. A műsor vezető kinyit egy ajtót nem azt amelyik mögött az autó van majd megkérdezi a játékostól, hogy akar-e változtatni a döntésén. Változtatott-e vagy sem utána kinyílik az ajtó mögötte a nyereménnyel.

Az R-ben készített program:

```
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

    if(kiserlet[i]==jatekos[i]) {

        mibol=setdiff(c(1,2,3), kiserlet[i])
```

```
else{

    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))


}

musorvezeto[i] = mibol[sample(1:length(mibol),1)]


}

nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]


}

valtoztatesnyer = which(kiserlet==valtoztat)

sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

A program elején beírjuk a kísérletek számát, ami azt adja meg, hogy hányszor végezzük el a játékot.

Az ajtókat 1,2,3 számokkal adjuk meg. A kísérlet nevű vektorba beleírjuk, hogy a különböző esetekben melyik "ajtó" mögött szerepel a nyeremény. Az 1-3 közötti értéket a kísérletek számával megegyező értékben fogja elhelyezgetni, majd a replace=T -vel érjük el az ismétlődést. A játékos vektor a játékosunkat fogja szimbolizálni aki ugyanúgy 1-3 ig választ itt is annyiszor ahány kísérlet van. Itt is elhelyezzük a replace=T-t mivel a játékos választhatja minden ugyanazt az "ajtót". A műsorvezetőnek külön létrehozunk egy vektort, melynek mérete megegyezik a kísérletek számával. Egy for ciklussal végig megyünk a vektorokon, ha a kísérlet vektor i-edik eleme megegyezik a játékos vektor i-edik elemével, azaz a játékos találja, hogy melyik ajtó mögött található a nyeremény akkor létrehozunk egy mibol vektort amelyben az 1,2,3 halmazból kivesszük a kísérlet vektort, azaz ezt az ajtót nem nyithatjuk ki hanem a másik 2-ből kell választanunk. Else ágon tovább haladva ha a játékos nem találja el elsőre a nyereményt, akkor nem nyithatjuk ki se a nyereményt tartalmazó ajtót, sem azt amelyiket a játékos választotta, így ezeket kivesszük a halmazból. A műsorvezető vektorban megkapjuk, hogy a különböző esetekben melyik ajtót nyitja ki. Létrehozzuk a nem-valtoztatesnyer vektort ahol azoknak az eseteknek a számát tároljuk el ahol a játékos az első választás után rögtön találta a nyereményt és nem is változtatott ajtót. Ha mégis változtatna a valtoztat vektor hosszát egyenlővé tesszük a kísérletek számával, ez azért kell, hogy ha minden kísérletben történne változtatás. Egy forral végig megyünk a kísérleteken és az 1,2,3 ajtók közül kivesszük azt amelyiket választottuk és azt amelyiket a műsorvezető kinyitotta. Ezt a holvalt vektorban tároljuk. Majd a valtoztat vektor ilyenkor a holvalt értékét kapja meg, ami a megváltoztatott ajtó lesz. A valtoztatesnyer vektor azoknak az eseteknek

a számát tárolja amikor a játékos megváltoztatja az elsőre kiválasztott ajtót és úgy nyer. Végül kiíratjuk a kísérleteknek a hosszát azokat az eseteket ahol nem változtatott és nyert illetve fordítva. Majd ezek hányadosát, hogy megkapjuk melyik eset volt a kedvezőbb és ellenőrzés képpen ezeknek az összegét, hogy megkapjuk pontosan a kísérletek számát.

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 $a^n b^n c^n$ nyelv nem környezetfüggetlen

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... : A formális nyelvek vizsgálatának egyik legmeghatározóbb alakja Noam Chomsky. Léteznek generatív és analitikus nyelvtanok. A generatív a legismertebb ez egy szabályhalmaz, amellyel a nyelv minden jelsorozata előállítható. Chomsky 1950-ben javasolta a nyelv formalizálását. Egy G nyelv nem-terminális szimbólumok véges halmazából(N), terminális szimbólumok véges halmazából(SUM-jel), produkciós szabályok véges halmazából(P) és az N-hez tartozó S szimbólumból áll.

3.3. Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiál 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: <https://gitlab.com/baluka94/bhax/blob/master/Chomsky/c89> <https://gitlab.com/baluka94/bhax/blob/master/Chomsky/c99>

Tanulságok, tapasztalatok, magyarázat... :

Az utasítások egymást követően, sorban hajtódnak végre, az ettől való eltérést külön jelezzük.

A legtöbb utasítás kifejezés jellegű, alak: kifejezés; . BNF: <kifejezés>;

Annak érdekében, hogy ott, ahol elvileg csak egy utasítás helyezhető el, több utasítás is használható legyen, van az összetett utasítás, blokk. BNF: <block> ::= {<block statements>}

Az if utasítás. A gép minden esetben végrehajtja az utasítást, ha az értéke nem 0 az első fut le, ha 0 akkor a második. BNF: if (<kifejezés>) <utasítás>

A while utasítás: Az alutasítás végrehajtása mindaddig ismétlődik, amíg a kifejezés értéke nem nulla marad. A vizsgálat mindenkor az utasítás egyes végrehajtásai előtt történik. BNF: while(<kifejezés>) <utasítás>

A do utasítás: Az alutasítás végrehajtása mindenkor ismétlődik, amíg kifejezés értéke nullává nem válik. A vizsgálat mindenkor az utasítás egyes végrehajtásai után történik. do <utasítás> while <kifejezés>

A for utasítás: Utasításokat hajt végre, amíg a benne leírt feltétel nem teljesül, van benne léptető kifejezés. BNF:

A switch utasítás: A switch hatására az adott kifejezés értékétől függően a vezérlés több utasítás valamelyikére adódik át. BNF: switch (kifejezés)

A break utasítás: A break hatására befejeződik az őt körülvevő utasítás végrehajtása. A vezérlés a befejezett utasítást követő utasításra tevődik át. (for, while). BNF: break;

A return utasítás: A függvény a hívójához a return utasítással tér vissza. BNF: return<kifejezés>;

A continue utasítás: Hatására a vezérlés az utasítást körülvevő legbelő utasítás ciklusfolytatón részére adódik át. BNF: continue;

A goto utasítás: A vezérlés feltétel nélkül a goto utasítással adható át. BNF: goto <azonosító>

A nulla utasítás: Tartalmazhat címkét, vagy ciklustörzset képezhet. BNF: ;

3.4. Saját lexikális elemző

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU (15:01-től).

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1

```
%{  
#include <stdio.h>  
int realnumbers = 0;  
%}  
digit [0-9]  
%%  
{digit}*(\.{digit}+)? {++realnumbers;  
    printf("[realnum=%s %f]", yytext, atof(yytext));}
```

```
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat... : Kezdő érték adással deklaráltuk a realnumbers változót ami int típusú. A digit a számjegyeket jelenti 0-9 ig. Következnek a fordítási szabályok. A digit * jelzi, hogy ebből akármennyi lehet lehet 0 is. A \. levédi a pontot ezzel tizedes pont lesz nem pedig karaktereket helyettesít, megint jönnek a számjegyek a + jelenti, hogy akárhány db, de legalább 1. A ? jelenti, hogy vagy van vagy nincs. Ha ilyet talált növelje a változót és kiíratom a valószámot először stringként majd az atof függvényel stringből átalakítva double kent is. A mainben meghívjuk a lexert és kiíratjuk a valószámokat.

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.1

```
% {
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>

#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))

struct cipher {
    char c;
    char *leet[4];
} l337d1c7 [] = {

{'a', {"4", "4", "@", "/-\\"}}, ,
{'b', {"b", "8", "|3", "|"}}, ,
{'c', {"c", "(", "<", "{"}}, ,
{'d', {"d", "|)", "[", "|"}}, ,
{'e', {"3", "3", "3", "3"}}, ,
{'f', {"f", "|=", "ph", "|#"}}, ,
{'g', {"g", "6", "[", "+"]}}, ,
{'h', {"h", "4", "|-", "[ - ]"}}, ,
{'i', {"i", "1", "|", "!"}}, ,
{'j', {"j", "7", "_|", "_/"}}}, ,
{'k', {"k", "|<", "1<", "|{"}}},
```

```
{'l', {"l", "1", "|", "|_|"},  
{'m', {"m", "44", "(V)", "|\\|/"},  
{'n', {"n", "|\\|", "/\\/", "/V"}},  
{'o', {"0", "0", "()", "[]"}},  
{'p', {"p", "/o", "|D", "|o"}},  
{'q', {"q", "9", "O_", "(,)"}},  
{'r', {"r", "12", "12", "|2"}},  
{'s', {"s", "5", "$", "$"}},  
{'t', {"t", "7", "7", "'|'"}}},  
{'u', {"u", "|_|", "(_)", "[_]"}},  
{'v', {"v", "\\/", "\\/", "\\/"}}},  
{'w', {"w", "VV", "\\//\\/", "(/\\)"}},  
{'x', {"x", "%", ")("}},  
{'y', {"y", "", "", ""}}},  
{'z', {"z", "2", "7_", ">_"}},  
  
'0', {"D", "0", "D", "0"}},  
{'1', {"I", "I", "L", "L"}},  
{'2', {"Z", "Z", "Z", "e"}},  
{'3', {"E", "E", "E", "E"}},  
{'4', {"h", "h", "A", "A"}},  
{'5', {"S", "S", "S", "S"}},  
{'6', {"b", "b", "G", "G"}},  
{'7', {"T", "T", "j", "j"}},  
{'8', {"X", "X", "X", "X"}},  
{'9', {"g", "g", "j", "j"}}}  
  
// https://simple.wikipedia.org/wiki/Leet  
};  
  
%}  
%%  
. {  
  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
    {  
  
        if(l337d1c7[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int)(100.0*rand()/(RAND_MAX+1.0));  
  
            if(r<91)  
                printf("%s", l337d1c7[i].leet[0]);  
            else if(r<95)  
                printf("%s", l337d1c7[i].leet[1]);  
            else if(r<98)  
                printf("%s", l337d1c7[i].leet[2]);  
            else
```

```
    printf("%s", 1337d1c7[i].leet[3]);\n\n    found = 1;\n    break;\n}\n\n}\n\nif(!found)\n    printf("%c", *yytext);\n\n}\n%%\nint\nmain()\n{\n    srand(time(NULL)+getpid());\n    yylex();\n    return 0;\n}
```

Tanulságok, tapasztalatok, magyarázat... : Létehozunk egy struktúrát cipher néven eben összetartozó adatokat tárolunk. Deklaráljuk a c char típusú változót. A char leet azt mondja, hogy minden karakternek 4 megfelelője lesz. Ebből csinálunk l337d1c néven egy tömböt, amibe betölthjük a betűket, számokat és az ōket helyettesítő 4db szimbólumot vagy karaktert stringként. Ezek voltak a programban a definíciók. A következő blokkban, ha pont következik lefut a beleírt program rész. Itt a pont bármilyen karaktert helyettesíthet. Felveszünk egy int típusú found változót 0 kezdő értékkel, ami azt nézi találtunk-e ilyen karaktert. A for ciklusban végig megyünk L337SIZE-ig aminek mérete a tömb leosztva 1 db cipher méretével. Az if feltételben a betűket vizsgálom ha a tömb i-edik-eleme megegyezik e a beírt karakterrel akkor random számot generál 0-100-ig. Ha 91-nél kisebb az első eset teljesül hivatkozok a leet 0-dik elemére. Ha 91-nél nagyobb 95-nél kisebb második eset, ha 95-nél nagyobb, 98-nál kisebb harmadik eset, 98-100-nál harmadik eset. A found értéke 1 lesz és kilép a ciklusból. Ha nem találtuk meg akkor visszaadja azt a karaktert amit beírtunk. A harmadik részében a mainben inicializálunk random számlálót és indítjuk a lexikális elemzést. Ekkor kezd el futni a definíciós rész.

3.6. A források olvasása

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

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

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

Bugok

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

i.

```
if(signal(SIGINT, SIG_IGN) !=SIG_IGN) //Ha a SIGINT jel kezelése ←  
    figyelmen kívül volt hagyva, akkor legyen ezután is, ha nem akkor a ←  
    jelkezelő függvény kezelje.  
    signal(SIGINT, jelkezelo);
```

ii.

```
for(i=0; i<5; ++i) //A for utasítás, ami 5x hajtja végre a benne lévő ←  
    utasítást. A ++i megnöveli az i értékét és a megnövelt értéket adja ←  
    vissza.
```

iii.

```
for(i=0; i<5; i++) //A for 5x hajta végre a benne lévő utasítást. Az i ←  
    ++ megnöveli az i értékét, de a növelés előtti értéket adja vissza.
```

iv.

```
for(i=0; i<5; tomb[i] = i++) // A ciklus 5-ször fut le lecseréli a ←  
    tömb értékeit az i++ visszatérési értékeivel.
```

v.

```
for(i=0; i<n && (*d++ = *s++); ++i) // A ciklus addig megy míg i ←  
    kisebb mint n és a d*++ visszatérés értéke érvényes.
```

vi.

```
printf("%d %d", f(a, ++a), f(++a, a)); // Standard kimenetre írjuk ki ←  
    a függvény által visszaadott értéket. A kód bugos.
```

vii.

```
printf("%d %d", f(a), a); // Kiíratjuk standard kimenetre a függvény ←  
    által visszatérített értéket.
```

viii.

```
printf("%d %d", f(&a), a); // Kiíratjuk standard kimenetre a függvény ←  
    által visszatérített értéket.
```

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{ prim})))$  
$ (\forall x \exists y ((x < y) \wedge (y \text{ prim})) \wedge (\exists y \forall x (x \text{ prim}) \supset (x < y))) \leftrightarrow  
$ (\exists y \forall x (y < x) \supset \neg (x \text{ prim}))$
```

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... :

Bármilyen x esetén egy y ahol $x < y$ és y prím szám.

Bármilyen x esetén egy y ahol $x < y$ y prím és y rákövetkezője is prím.

Létezik olyan y, hogy bármilyen x esetén, ha x prím akkor $x < y$.

Létezik olyan y, hogy bármilyen x esetén, ha $y < x$ akkor x nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciaja
- egészek tömbje
- egészek tömbjének referenciaja (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; Egy a nevű egész típusú változót.

- `int *b = &a;` Egy b nevű egészre mutató mutatót, ami az a változó címét tartalmazza.
- `int &r = a;` Egy r nevű egészre mutató mutatót, ami az a értékét mint cím tartalmazza. r értéke változik.
- `int c[5];` Egy 5 elemű egészekből álló tömböt.
- `int (&tr)[5] = c;` Egy 5 elemű egészekből álló tömbre mutató mutatót, a c tömbre.
- `int *d[5];` Egy 5 elemű egészekre mutató mutatólból álló tömböt.
- `int *h();` Egy jelölt egéssel visszatérő null paraméteres függvényre mutató mutatót.
- `int *(*l)();` Egy egészre mutató mutatót visszaadó függvényre mutató mutatót.
- `int (*v (int c)) (int a, int b);` Egészet visszaadó, 2 egészet kapó függvényre mutató mutatót visszaadó egészet váró függvény.
- `int (*(*z) (int)) (int, int);` Függvénymutató egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó egészet kapó függvényre.

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Chomsky/deklaracio>

4. fejezet

Helló, Caesar!

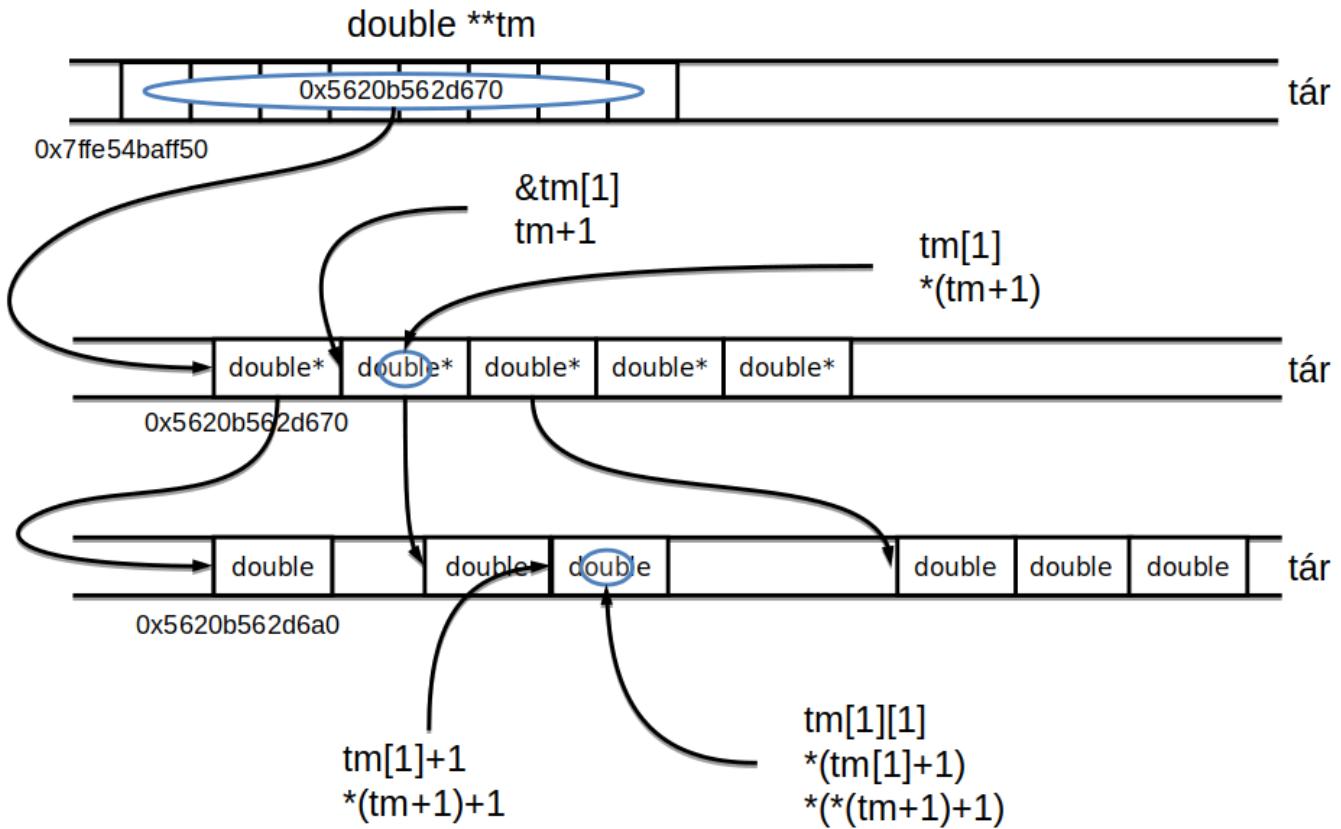
4.1. double ** háromszögmátrix

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Caesar/haromszogmatrix.c>
Bátfa Norbert forráskódja alapján

<https://gitlab.com/baluka94/bhax/tree/master/Caesar/doublepics> A kép Bátfa Norbert előadás fóliájáról származik, a futtatásról készült saját

Tanulságok, tapasztalatok, magyarázat... : A program elején lefoglalunk 8bájt helyet a tm számára a memóriában és kiíratjuk az első bájt címét prtinf függvény használatával. A malloc függvénytel 5*8 azaz 40 bájtnyi helyet foglal le az operációs rendszer a memóriában, mivel a függvény egy void típusú mutatót ad vissza típuskényszerítjük double**-ra. Ha nem tudott elegendő helyet lefoglalni, eredményül 0-t kapunk és kilép a programból."". Továbbiakban a malloc függvénytel n*8bájtnyi helyet foglalunk és double*-ra kényszerítjük kapott értéket. Ha nem sikerült megfelelő méretű memóriát lefoglalni -1-et kapunk értékül és kilép a program. A fennmaradó részekben a mátrix elemeit határozzuk meg és íratjuk ki.



4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás video:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Caesar/e.c> Bát-fai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : Meghatározunk kettő értéket amiket a MAX_KULCS(100) illetve BUFFER_MERET(256) helyeken használunk. Kettő karakter típusú tömböt deklarálunk melyben szereplő értékek számát a definiált értékekre állítjuk, ezzel megadva a kulcs méretét és a maximálisan beolvasott bájtok számát. A kulcs index és az olvasott bájtok számára fenntartott változók értékét kinullázzuk. A kulcs méretét az első parancssori argumentumunk hossza adja, mivel ebben adjuk meg a kulcsunkat. Ezután a kulcs nevű tömbbe fogjuk belemásolni a parancssori argumentum bájtjait, ami maximum MAX_KULCS-nyi bájtot jelent. A while ciklus addig fut, amíg a beolvasott bájtok száma el nem éri a szöveg méretét, ezt a read függvény adja vissza, mivel vele a beolvasott bájtok számát kapjuk meg. A for ciklussal a beolvasott bájtokon fogunk végigmenni és elvégezzük a titkosítást a kulcstól függően. A write függvény bájtonként írja bele a titkosított szöveget a buffer tömbbe.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Caesar/javaex.java> Bátfa Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A C-ben megírt programhoz képest itt a következő változások figyelhetőek meg: Az elején beírjuk, hogy kelleni fog nekünk egy kulcs egy bemeneti -és egy kimeneti fájl. A while ciklus addig megy míg be nem olvastunk minden bájtot. Az xor-ozott bájtokat beletöljtük a buffer tömbbe, a művelet végrehajtása a kulcs segítségével történik. A tömb elemeitt olvasottbajtok-ig írja bele egy kimeneti fájlba. Az olvasott bájtokat a bejovocsatorna-ból olvassuk a bufferbe. Megnézzük futtatáskor megfelelő argumentumokat kapott-e a program, ha nem ezt jelezzen a felhasználó irányába.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Caesar/t.c> Bátfa Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A double típusú átlagos_szohossz függvénytel a szavak átlagos hosszúságát akarjuk megadni. A for ciklussal végig megyünk a szövegen és ha szóközt találunk akkor megnöveljük a szóközt. A titkos_meret-et ami bájtokat tárol elosztjuk az sz-el ami a szavak száma, így megkapva egy szónak az átlagos hosszúságát.

A tiszta_lehet függvényben dekralájuk a szóhosszt ami egyenlő lesz az előző függvényben írt átlagos szóhosszal. A return akkor ad vissza igaz értéket, ha a vizsgálatok igazak, de ha már az egyik nem teljesül akkor nem lép tovább hamisat adva.

A void exor függvényben történik az exorozás, ami úgy zajlik mint a titkosítóban csak itt a buffer helyén a titkos áll.

Az exor_tores meghívja az exor függvényt a return pedig a tiszta_lehet függvényt.

A main függvény while ciklusában elkezdi olvasni a bájtokat a p titkos tömbre mutató pointerból. Ha a p-titkos+OLVASAS_BUFFER kisebb mint 4096 bájt akkor OLVASAS_BUFFER-ig olvas különben titkos+MAX_T p-ig. A p értékét megnöveli az olvasott bájtok számával.

A for végig megy a szövegen, majd azokat a helyeket amik nincsenek lefoglalva kinullázza.

A sok for ciklussal a kulcsnak az összes lehetséges karaktereit állítjuk elő, ami maximum 8 karakter lehet, ameddig az exor_tores hamis, addig generálja a kulcsokat, ha az exor_tores igaz akkor kiírja a kulcsot, újra össze-exorozzuk így nem fog kelleni második buffer.

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... : A neuron matematikai modellje szerint "tüzel" egy neuron ha a bemeneti értékek súlyozott összege egy határétként túllép. A bemeneten számok jönnek amelyek rendelkeznek egy súly értékkal. Meghívjuk a neuralnet könyvtárat és a1-be, a2-be betöljtük a neuronok lehetséges értékeit, az or-ba betöljtük a kettő logikai vagy műveletének értékét. Ahhoz, hogy a tanítás sikeres legyen, elmondjuk az összes lehetséges esetet, ezután a program elkezdi tanítani magát és beállítja a súlyokat. Ugyan olyan módon tanítjuk meg neki az és műveletet és az exort, ami még nem olyan hatékony. Annál effektívebb a tanulási algoritmus minnél több benne a rejtett neuronok száma.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Caesar/perceptron>
Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A main-ben létrehozzuk a képet. Deklaráljuk a size változót amiben eltároljuk a kép méretét, amit úgy kapunk, hogy a kép szélességét megszorozzuk a magasságával. Új perceptronról csinálunk amiben megadjuk, hogy 3 réteg lesz az első rétegre annyi neuront akarunk amekkora a size a másodikra 256-ot harmadik pedig 1, ezt számoljuk. A double* image-el helyet foglalunk a képnél és ezt a helyet fel is töltjük 2db forcklussal, ide a pixelek rgb kódjai kerülnek. Az első forban az i a szélességig megy a másodikban a j a hosszúságig. Ezután a betöltött pixeleket pirosra színezzük. A p perceptronról meghívjuk mint függvényt és értékként átadjuk neki a perceptron objektumot amit a double value-ban tárolunk. Majd kiíratjuk a value-t. Ezt a p nevű perceptronról és a törzsben sem foglalt image-et a kód végén töröljük is, hogy ne foglaljon helyet.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Mandelbrot/mandelp.cpp> Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A Mandelbrot halmazt Benoit Mandelbrot találta meg 1980-ban a komplex számsíkon. A komplex számokkal olyan kérdéseket tudunk megválaszolni például, hogy melyik két szám szorzata -9 ez a 3i. A Mandelbrot halmaz megtalálásához egy origó középpontú 4 oldalhosszúságú négyzetre van szükségünk, amiben lefektetünk mondjuk egy 800x800-as rácscsötét és kiszámoljuk, hogy a rács pontjai mely komplex számoknak a megfelelői. A $z_{n+1} = z_n^2 + c$ ($0 \leq n$) képlet alapján a rács minden pontját megvizsgáljuk, úgy hogy a c az aktuális rácspont. A z_0 az origó. Azaz kiindulunk az origóból elugrunk az első rács pontjába $z_1 = c$ -be aztán c-től függetlenül a többi z-be. Ha az "utazás" kivezet a 2 sugarú körből, akkor a rácspont nincs benne a halmazban. minden rácsponthoz véges z elemet nézünk meg. Ha nem lép ki a körből feketére színezük. A színes képet úgy is elérhetjük, hogy változatosan színezünk, minél későbbi z-nél lép ki a körből annál sötétebb.

Létrehozunk egy void típusú függvényt, ami egy 2 dimenziós tömböt ad vissza egy 600x600-ast. Megnézzük a program futásához szükséges óraütések számát. Következnek a számításhoz szükséges adatok, a kép szélessége, magassága és egy változó az iterációs határ meghatározásához, ami később a ciklusban fog kelleni. Felvesszük külön a valós és külön a képzetes valós szám részeknek a változókat és egy gyűjtőváltozót az iterációk számához 0 kezdőértékkel.

```
void
mandel (int kepadat [MERET] [MERET]) {

    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;
```

```
// a számítás
float dx = (b - a) / szelesseg;
float dy = (d - c) / magassag;
float reC, imC, reZ, imZ, ujreZ, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
```

Egymásba ágyazott for ciklusokkal végigmegyünk a rács sorain és előállítjuk a komplex számok valós és imaginárius részeit. Az első elem 0 lesz a második az első elem négyzetéhez a c szám hozzáadva.

```
// Végigzongorázzuk a szélesség x magasság rácsot:
for (int j = 0; j < magassag; ++j)
{
    //sor = j;
    for (int k = 0; k < szelesseg; ++k)
    {
        // c = (reC, imC) a rács csomópontjainak
        // megfelelő komplex szám
        reC = a + k * dx;
        imC = d - j * dy;
        // z_0 = 0 = (reZ, imZ)
        reZ = 0;
        imZ = 0;
        iteracio = 0;
        // z_{n+1} = z_n * z_n + c iterációk
        // számítása, amíg |z_n| < 2 vagy még
        // nem értük el a 255 iterációt, ha
        // viszont elértek, akkor úgy vesszük,
        // hogy a kiinduláci c komplex számra
        // az iteráció konvergens, azaz a c a
        // Mandelbrot halmaz eleme
```

A while ciklus addig fut, amíg az előállított komplex szám szám négyzete eléri a 4-et az iteráció pedig a 32000-et. A ciklusban létrehozzuk a következő komplex számot az előző értékét felülírjuk az újjal és növeljük az iterációk számát.

A main-ben megvizsgáljuk, hogy elegendő argumentum lett-e megadva futtatáshoz, ha nem kiíratjuk a felhasználó számára, hogy hogyan tudja lefuttatni a programot.

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

    if (argc != 2)
    {
        std::cout << "Használat: ./mandelpng fajlnev";
        return -1;
    }
```

Létrehozunk egy két dimenziós tömböt, meghívjuk a mandel függvényt és előállítjuk a képet. Előállítjuk a pixelek rgb kódját.

```
int kepadat[MERET][MERET];  
  
mandel(kepadat);  
  
png::image<png::rgb_pixel> kep(MERET, MERET);  
  
for (int j = 0; j < MERET; ++j)  
{  
    //sor = j;  
    for (int k = 0; k < MERET; ++k)  
    {  
        kep.set_pixel(k, j,  
                      png::rgb_pixel(255 -  
                                      (255 * kepadat[j][k]) / ITER_HAT ←  
                                      ,  
                                      255 -  
                                      (255 * kepadat[j][k]) / ITER_HAT ←  
                                      ,  
                                      255 -  
                                      (255 * kepadat[j][k]) / ITER_HAT ←  
                                      ));  
    }  
}
```

A parancssori argumentum első eleme a név, valamelyen néven kell létrehozni a képet, értesítjük a felhasználót, hogy elkészült a kép a választott néven.

```
kep.write(argv[1]);  
std::cout << argv[1] << " mentve" << std::endl;
```

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Mandelbrot/3.1.2.cpp> Bátfa Norbert forráskódja alapján

Meghívjuk a complex headert, amivel lehetővé válik a komplex számok használata. Itt a programban a függvények használata helyett minden a main-ben írunk meg. Felvesszük a szélességet, magasságot és az a,b,c,d változókat. Megnézzük, hogy 9 parancssori argumentumt kaptunk-e futtatáskor, ha igen az atoifüggénnyel betöljtük a szélességbe az argumentum második elemét ugyanígy a magassággal és az iterációs határral, ezek rendre a 3. és 4. argumentumt kapják. Az a,b,c,d változók mivel double típusúak itt az atoifüggényt kell használnunk ezek az 5. 6. 7. 8. argumentumokat kapják. Ha a feltétel nem teljesül kiíratjuk, hogy hogyan kell a programot futtatni. Létrehozzuk a képet, felvesszük a szükséges váltokat rögzítjük a számítás állapotát. Végigmegyünk a pixeleken, előállítjuk a komplex számot, ami minden hozzáadódik a sorozat aktuális eleméhez, majd a sorozat első elemét komplex típusú változóként vesszük fel 0 kezdőértékkel. A while ciklus addig fut, amíg a komplex szám értéke 4 lesz, illetve az iterációk száma eléri az iterációs határt. A ciklusban vesszük ennek a számnak a négyzetét és hozzáadjuk az előállított c komplex számot és megnöveljük az iterációk számát. A kép pixeleinek rgb kódját az iteráció változó értéke alapján

módosítjuk. A számítás állapotát %-ban közöljük a felhasználóval, ez folyamatosan frissül köszönhetően az std::flush-nek. Ez után a parancssori argumentum első elemeként kapott néven legyártjuk a képet és kiírjuk, hogy a kép elkészült.

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbqRzY76E>

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A különbség a Mandelbrot és a Julia halmazok között az, hogy a komplex iterációban az előbbiben c változó, utóbbiban állandó. A main-ben deklaráljuk a változókat int szelesseg,magassag, ami a monitor felbontásához van igazítva. Felvesszük az iterációs határt az x és y max értékét, valamit az ec valós és imaginárius részét. Ha a megadott argumentumok száma megegyezik a programban megadottével a program lefut különben visszaadja, hogy milyen argumentumok kellenek a program megfelelő futtatásához.

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );
```

```
}

else
{
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesség magasság n a b c ←
        d reC imC R" << std::endl;
    return -1;
}

png::image<png::rgb_pixel> kep ( szelesség, magasság );

double dx = ( xmax - xmin ) / szelesség;
double dy = ( ymax - ymin ) / magasság;

std::complex<double> cc ( reC, imC );

std::cout << "Számítás\n";

// j megy a sorokon
for ( int y = 0; y < magasság; ++y )
{
    // k megy az oszlopokon

    for ( int x = 0; x < szelesség; ++x )
    {

        double reZ = xmin + x * dx;
        double imZ = ymax - y * dy;
        std::complex<double> z_n ( reZ, imZ );

        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {

            z_n = std::pow(z_n, 3) + cc;
            //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
            if (std::real ( z_n ) > R || std::imag ( z_n ) > R)
            {
                iteracio = i;
                break;
            }
        }

        kep.set_pixel ( x, y,
                        png::rgb_pixel ( (iteracio*20)%255, (iteracio ←
                            *40)%255, (iteracio*60)%255 ) );
    }
}

int szazalek = ( double ) y / ( double ) magasság * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
}
```

```
kep.write ( argv[1] );
std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

A Mandelbrot-hoz képest itt 2 változóval és egy valós számmal többet deklarálunk. A parancssori argumentumok száma 3-al nő. A 3 argumentum az reC,imC és R értékeit adják. Az reC és imC alapján megadjuk a komplex számot. A while ciklus helyett for-t használunk, ami az iterációs határig megy. Létrejön a komplex szám a megadott módon. A feltételes utasítás tartalmazza Pickover hibáját. Itt a valós és imaginárius részre teszünk vizsgálatot nem pedig a komplex szám hosszára. A program végén történik a pixel színezés.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Mandelbrot/mandelc>
Bátfai Norbert forráskódja alapján

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/Mandelbrot/mndnagy>
A forráskód Harmati Norbert-től származik.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Mandelbrot/nagy%C3%ADt%C3%B3.java>
Bátfai Norbert forráskódja alapján

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: <https://gitlab.com/baluka94/bhax/blob/master/Welch/polargen.java> Bátfa Norbert forráskódja alapján <https://gitlab.com/baluka94/bhax/blob/master/Welch/polargen.cpp>

Tanulságok, tapasztalatok, magyarázat... térd 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!

A polártranszformációs algoritmus egy random szám generátor, ami két értéket hoz létre.

A programban meghívjuk a szükséges könyvtárakat.

Létrehozunk egy Random osztályt elhelyezzük benne a private illetve publikus részeit sz objektumnak. Publikusak a konstruktor illetve destruktur, valamint a get() függvény, ami a random szám generálásáért és visszaadásáért felel. A privát részben a változókat tároljuk, amik a random értékeket tartalmazzák és azt, hogy létezik-e már legenerált érték vagy sem logikai változóként. Az osztály után definiáljuk a konstruktort, új objektum létrehozása esetén az exist változó hamis lesz és inicializáljuk a random() függvényt. Megadjuk a get() random lekérdező függvényt, ami 2 részes. Ha nem létezik generált szám vagyis hamis az exist, akkor legenerálunk 2 számot az egyiket visszaadjuk a másikat eltesszük a logikai változót igazra állítjuk. Ha létezik a szám azaz elvan tárolva az exist-et hamisra állítjuk és visszaadjuk az eltárolt értéket. A programban példányosítjuk az osztályt egy forral és a get() segítségével kiírunk a terminálba 10 pszeudorandom számot.

6.2. LZW

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Welch/binfa.c> Bátfa Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... :

A binfa egy rekurzív adatszerkezet. Külső és belcső csúcsokat (szülőt és gyerekeket) rendez el hierarchikusan. A bináris fában minden csomópontnak 2 gyereke lehet legfeljebb. Az utódokat megkülönböztetjük, hogy bal vagy jobb oldali részfák. A program bemenetként egy szöveges fájlt kap és a benne található karaktereket a program binfa szerkezetre alakítja és ezt egy kimeneti fájlban tudjuk megnézni.

Definiáljuk a binfa struktúrát és a BINFA-PTR-t ez a felhasználó által megadott típusú lesz.

```
typedef struct binfa
{
int ertek;
struct binfa *bal_nulla;
struct binfa *jobb_egy;
} BINFA, *BINFA_PTR;
BINFA_PTR
uj_elem ()
{
BINFA_PTR p;
if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
{
perror ("memoria");
exit (EXIT_FAILURE);
}
return p;
```

A fa felépítése:

```
int
main (int argc, char **argv)
{
char b;
int egy_e;
int i;
unsigned char c;
BINFA_PTR gyoker = uj_elem ();
gyoker->ertek = '/';
gyoker->bal_nulla = gyoker->jobb_egy = NULL;
BINFA_PTR fa = gyoker;
long max=0;
while (read (0, (void *) &b, sizeof(unsigned char)))
{
for(i=0;i<8; ++i)
{
egy_e= b& 0x80;
if ((egy_e >>7)==0)
c='1';
else
c='0';
}
// write (1, &b, 1);
if (c == '0')
```

```
{  
if (fa->bal_nulla == NULL)  
{  
fa->bal_nulla = uj_elem ();  
fa->bal_nulla->ertek = 0;  
fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;  
fa = gyoker;  
}  
else  
{  
fa = fa->bal_nulla;  
}  
}  
else  
{  
if (fa->jobb_egy == NULL)  
{  
fa->jobb_egy = uj_elem ();  
fa->jobb_egy->ertek = 1;  
fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;  
fa = gyoker;  
}  
else  
{  
fa = fa->jobb_egy;  
}  
}  
}  
printf ("\n");  
kiir (gyoker);
```

Létrehozzuk a fa gyökerét és a jobb és bal oldali gyerekeket. Ahhoz, hogy ezek számát pontosan tudjuk meg kell nézni az ágak átlagos hosszát, azaz a szórás összegét az átlagot valamint a mélységet.

```
extern int max_melyseg, atlagosszeg, melyseg, atlagdb;  
extern double szorasosszeg, atlag;  
printf ("melyseg=%d\n", max_melyseg - 1);  
/* Átlagos ághossz kiszámítása */  
atlagosszeg = 0;  
melyseg = 0;  
atlagdb = 0;  
ratlag (gyoker);  
// atlag = atlagosszeg / atlagdb;  
// (int) / (int) "elromlik", ezért casoljuk  
// K&R tudatlansági védelem miatt a sok () :)  
atlag = ((double) atlagosszeg) / atlagdb;  
/* Ághosszak szórásának kiszámítása */  
atlagosszeg = 0;  
melyseg = 0;  
atlagdb = 0;  
szorasosszeg = 0.0;
```

```
rszoras (gyoker);
double szoras = 0.0;
if (atlagdb - 1 > 0)
szoras = sqrt (szorasosszeg / (atlagdb - 1));
else
szoras = sqrt (szorasosszeg);
printf ("atlag=%f\nszoras=%f\n", atlag, szoras);
szabadit (gyoker);
}
```

Inicializáljuk az átlag és a szórás függvényeket:

```
int atlagosszeg = 0, melyseg = 0, atlagdb = 0;
void
ratlag (BINFA_PTR fa)
{
if (fa != NULL)
{
++melyseg;
ratlag (fa->jobb_egy);
ratlag (fa->bal nulla);
--melyseg;
if (fa->jobb_egy == NULL && fa->bal nulla == NULL)
{
++atlagdb;
atlagosszeg += melyseg;
}
}
}

double szorasosszeg = 0.0, atlag = 0.0;
void
rszoras (BINFA_PTR fa)
{
if (fa != NULL)
{
++melyseg;
rszoras (fa->jobb_egy);
rszoras (fa->bal nulla);
--melyseg;
if (fa->jobb_egy == NULL && fa->bal nulla == NULL)
{
++atlagdb;
szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
}
}
}
```

A kiir függvény inorder bejárással íratja ki a fát:

```
int max_melyseg = 0;
void
```

```
kiir (BINFA_PTR elem)
{
if (elem != NULL)
{
++melyseg;
_melyseif (melyseg > maxg);
max_melyseg = melyseg;
kiir (elem->jobb_egy);
for (int i = 0; i < melyseg; ++i)
printf ("---");
printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek --
,
melyseg - 1);
kiir (elem->bal_nulla);
--melyseg;
}
}
```

A program végén felszabadítjuk az elemeket.

```
void
szabadit (BINFA_PTR elem)
{
if (elem != NULL)
{
szabadit (elem->jobb_egy);
szabadit (elem->bal_nulla);
free (elem);
}
}
```

6.3. Fabejárás

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Welch/zpre.c> <https://gitlab.com/baluka94/bhax/blob/master/Welch/zpost.c>

Tanulságok, tapasztalatok, magyarázat... : A fán alapból a kiir függvény inorder bejárást végzett, ezen kellett alakítani. Inorder: Előbb a jobb oldali gyereket járja be, feldolgozza a gyökeret, majd a baloldalit járja be rekurzívan. Preorder: Feldolgozzuk a gyökeret elsőként előbb a bal oldali gyereket járjuk be, majd a jobb oldalit. Postorder: A gyökeret utoljára fogjuk benne feldolgozni.

Preorder:

```
void
kiir (BINFA_PTR elem)
{
if (elem != NULL)
{
```

```
++melyseg;
_melyseif (melyseg > maxg);
max_melyseg = melyseg;
for (int i = 0; i < melyseg; ++i)
printf ("---");
printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek <-
,
melyseg - 1);
kiir (elem->bal nulla);
kiir (elem->jobb_egy);
--melyseg;
}
}
```

Postorder:

```
void
kiir (BINFA_PTR elem)
{
if (elem != NULL)
{
++melyseg;
_melyseif (melyseg > maxg);
max_melyseg = melyseg;
for (int i = 0; i < melyseg; ++i)
printf ("---");
printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek <-
,
melyseg - 1);
kiir (elem->bal nulla);
kiir (elem->jobb_egy);
--melyseg;
}
}
```

6.4. Tag a gyökér

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Welch/z3a7.cpp>
Bátai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... :

A fát egy LZWBInFa osztályba ágyazzuk. Ebbe elhelyezzük a fa egy csomópontjának a jellemzését ez a beágyazott csomópont osztály a fa részeként számolunk vele külön nincs szerepe.

```
class LZWBInFa
{
```

```
public:  
LZWBinFa () :fa (&gyoker)  
{  
}  
~LZWBinFa ()  
{  
szabadit (gyoker.egyesGyermek ());  
szabadit (gyoker.nullasGyermek ());  
}
```

6.5. Mutató a gyökér

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Welch/mutatogyoker.cpp> Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... :

A csomópontot átírjuk mutatóra.

```
Csomopont *gyoker;
```

Következőnek a konstruktornak írnunk át:

```
LZWBinFa (gyoker)  
{  
fa = gyoker;  
}
```

Mutató állítás a gyerekre a szabadít függvényben:

```
~LZWBinFa ()  
{  
szabadit (gyoker -> egyesGyermek ());  
szabadit (gyoker -> nullasGyermek ());  
}
```

Végül a &gyoker-eket a programban simán gyoker-re cseréljük, mivel nem a pointer címe kell, hanem az a cím amire a pointer mutat.

6.6. Mozgató szemantika

Írj az előző programhoz mozgató konstruktort és értékkadást, a mozgató konstruktornak legyen a mozgató értékkadásra alapozva!

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Welch/mozgato.cpp> Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A mozgató konstruktör jobb értéket vár paraméterül a dupla referencia jel jelenti a jobbértéket, a gyökér pointert null pointerré tesszük és a régi fára meghívjuk a move() függvényt, ami előkészíti nekünk mozgatásra. A függvény a paraméterként kapott változóból jobbértéket készít. A *this adja azt a fát, ahova lett mozgatva. A swap() függvény felcseréli a két fa gyökér mutatóját. Amikor átraktuk a gyoker változót null pointerre, akkor a régi gyökeret változtattuk meg valójában. Az = nél meghívódott a binFa& operator=(Binfa&& regi) ahol kicseréltük a 2 gyökeret és visszakaptuk a gyoker változót, amiben a régi fa volt, az új fa *this-be lett mozgatva. Végén kiíratjuk a fát, valamit a mozgatással létrejött fát.

DRAFT

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/Conway/hangya>
Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... :

A program a hangyák viselkedését próbálja ábrázolni.

Létrehozunk egy hangya osztályt, amiben eltároljuk a hangya tulajdonságait. Egy hangyának van oszlopa és sora ez az x és y változók és irány a dir változó.

```
class Ant
{
public:
    int x;
    int y;
    int dir;

    Ant(int x, int y): x(x), y(y) {
        dir = qrand() % 8;
    }

};

typedef std::vector<Ant> Ants;
#endif
```

Az antwin a világot hozza létre lehelyez egy rácsot a világra és cellákat helyez el rajta. Itt megadjuk a cellák értékét. Egy kis cella 6x6 pixelből áll. Az AntThread(Hangyaboly) egy származtatott osztály, a hangyák mozgatását végzi. A grids két rács a gridIdx pedig a két rácpont közül egyet tárol. A max és min a lehetséges minimum 1 és maximum 255 feromon értékét jelenti. A closeEvent() a kikapcsolási esemény megírja a finish metódust, ami befejezi a programot a running hamissá tevésével, a keyPressEvent() a gombnyomásokat dolgozza fel a paintEvent() a színezést végzi. Az evaporation a párolgás mértékét tárolja a feromonok és hangyák száma 1 cellán belül. A konstruktor és destruktur publikusak mint a run() vagy finish(). Nem publikusak a private függvények mint a setPheromone().

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Conway/Sejtautoma.java> Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : Az életjátékot John Conway hozta létre Neumann János kutatásait felhasználva. Egy matematikai modellt szeretett volna alkotni ami képes előállítani egy önreprodukáló rendszert.

A játéktér cellákból áll, ezek a cellák tartalmazzák a sejteket. 3 szabályt követünk a programban, ami egy összetett rendszert alkot.

A cellák tartalmát a szomszédos cellák tartalmától függően módosítjuk a szabályok szerint. Egy sejtnek, ha 2 vagy 3 élő szomszédja van akkor élő marad. Egy sejt akkor pusztul el, ha 3-nál több élő szomszédja van, vagy ha 2-nél kevesebb. Egy sejt élő lesz ha pontosan 3 élő szomszédja van. Két rácsot alkalmazunk, mivel a vizsgálatok a program végén mennek végbe, így az első rácsot alapul véve hozza létre a változásokat a másodikon.

Előlényeket helyezünk el azaz siklókat. Ezek adott irányba haladnak és másolják magukat a sejttérben. A rács két dimenziós tömbbe helyezi az állatkát. Az x a befoglaló téglalap bal felső sarkának oszlopa, az y pedig a sora. A sikló ágyú adott irányba lövi ki a siklókat. A program futtatása után az S billentyűvel képet tudunk készíteni a sejttér állapotáról az N-nel a sejtek méretét növeljük a K-val csökkentjük, a G-vel gyorsítani az I-vel lassítani lehet a szimulációt. Az egérmutató jobb illetve bal gombjával a sejt állapota az ellenkezőjére változik. Az egérmutató mozgatásával a sejteket életre keltjük.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/Conway/Sejtautocpp> Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A kódot szinte egy az egyben átvettük a Java-ból C++-ra igazítva. Futtatásához a Qt környezetet használjuk.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/Conway/BrainB>
Bátfai Norbert forráskódja alapján

Tanulságok, tapasztalatok, magyarázat... : A BrainB Benchmark egy játékosok kognitív tulajdonságát felmérő program és ezáltal osztályozza őket. Az *Samu* nevű entitásra kell helyezni az egérmutatót, a cél az, hogy ne vesszen el mialatt a program új karaktereket ad ki, folyamatosan újabb dobozok jelennek meg és egyre nehezebb lesz követni a pöttyöt. Ha 1 másodpercen felül nincs találat a játékos elvesztette a dobozat, ekkor lelassul a mozgása. A teszt 10 percig tart. A *updateHeroes* adja vissza, hogy elvesztettük-e Samut, vagy megtaláltuk-e, illetve a végrehajtás számát.

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

8. fejezet

Helló, Berners-Lee!

8.1. Python: Bevezetés a mobilprogramozásba

A Python egy platformfüggetlen programozási nyelv, magas szintű objektumorientált programozási nyelv. Egyik mobilprogramozáshoz használt változata a PythonS60 Symbian mobil op. rendszerre. A Python a C-től eltérően interpreteres nyelv nem szükséges fordítani futtatás előtt. A nyelv előnye, hogy tartalmaz magas szintű adattípusokat, nincs szükség zárójelekre a kódcsoportosítás tabulátorral van megoldva, illetve a változókat sem kell definiálnunk. Az értelmező a sorokat tokenekre bontja és úgy értelmezi.

A Python minden adat objektum, a változók típusait nem kell nekünk megadni, a futás során értékének megfelelő típust kap. A változótípusokból a legtöbb a C-ben is megtalálható, ezektől eltérő változók a szótár, ennesek és a listák. Ezek egyszerre több, akár eltérő típusú értéket tartalmaznak és szekvenciáknak nevezzük őket. Indexelésük 0-tól kezdődik, de ha negatívan indexelünk a szekvencia végétől kezdődik. A len és del függvényekkel lekérhetjük a hosszukat, és törölhetünk elemeket. A listák rendezettek és dinamikusak, míg a szótárak rendezetlen kulcsokkal azonosított elemeket tartalmaznak. Egy változóhoz értékadáskor hozzárendelhetünk objektumot, típust és függvényt. Az elágazások és ciklusok hasonlóak, mint C-ben, a for ciklushoz használhatunk range(x) és xrange(x) függvényeket, ami egész értékeket tartalmaz 0-tól x-ig. Az elseif Python-os változata pedig az elif. Függvényeket a def kulcsszóval hozhatunk létre. A paraméterek érték szerint adódnak, át kivéve pl., ha listákat szótárakat adunk meg. Lehetőség van az alapértelmezett paraméterek megadására is, illetve megadhatunk paramétereket függvényhíváskor is.

Címkeket a label kulcsszóval lehetünk a szövegbe, itt fontos megemlíteni a goto és comefrom függvényeket. Ha a program futáskor elér a goto függvényhez az adott label-hez ugrik. A comefrom ettől ellentétesen, ha elér a program az adott label-hez a comefrom-ot tartalmazó kódrészhez ugrik. Lehetőségünk van osztály-definiálásra is, ezeknek a példányai objektumok, és tulajdonságai lehetnek függvények és objektumok. Ha egy osztályban valamilyen tulajdonságot megváltoztatunk, akkor az megváltozik az osztály példányainban is, ha csak az nem volt bennük már előtte megváltoztatva. Az attribútum függvényeket, vagy metódusokat globális függvényként lehet definiálni, az első paraméterük mindig a self lesz, ami arra az objektumpéldányra utal, ahol a függvényt meghívjuk. Az `__init__` metódus egy speciális metódus, ami konstruktorként működik. Első paramétere self, az objektum, amit létre akarunk hozni, a többi paraméter pedig a tulajdonságok, amiket hozzá akarunk rendelni az objektumhoz.

A Python tartalmaz mobilfejlesztésre használt modulokat. Ilyenek például az appuifw, amellyel kialakíthatjuk és kezelhetjük a felhasználói felületet. A messaging modul az üzenetkezelésért, a sysinfo modul a telefonnal kapcsolatos információk lekérdezésért felelősek. A kameráért és a hangért felelős modulok a camera és audio modul.

A hibakezelést try-except blokkban oldjuk meg a try blokkba kerül a kód, amiben a hibát keressük, az except blokkban pedig az a kód, ami lefut, ha hibát találunk. A keresendő hibát az except mellé írjuk [] zárójelekbe. Kapcsolhatunk egy else ágat is, az except után vagy esetleg helyette egy finally blokk, amit például fájlok bezárására lehet használni hiba esetén

8.2. Java: Java 2 útikalauz 5

A Java nyelv teljesen objektumorientált, egy program kizártlag osztályokból és objektumokból épül fel. A programunkat futtatás előtt fordítani kell, ilyenkor bájtkódokra fordul a program, amit a Java Virtuális Gép interpreteként értelmez és futtat. A nyelv UNICODE karaktereket használ.

Az egyszerű változótípusok hasonlóak, mint a C-ben/C++-ban. A C-hez képest újdonság viszont a String típus, amit a karakterláncok kezelésére hoztak létre. Az azonosítók csak betűvel kezdődhetnek és nem tartalmazhatják a nyelv kulcsszavait. Az egyszerű típusúakhoz valódi értékadással, míg az összetett típusok esetén referencia által rendelünk hozzájuk értéket.

Az egyszerű és összetett típusok inicializálásakor literálokat használunk. Objektumoknál ez a null, ami bármely objektumreferencia helyett használható. A logikai érték lehet igaz vagy hamis, az egész számokat a kezdőkarakterük határozza meg: lehetnek oktális, hexadecimális és decimálisak

Ha egy értékét többször fel szeretnénk használni egy programban és nem akarjuk változtatni az értékét, érdemes konstansként, nem pedig változóként deklarálni. Ezt a final static kulcsszavakkal tehetjük meg. A lebegőpontos számok decimálisak tizedesponttal elválasztva. Karaktereket” szövegeket” jelek közé írunk. A szövegek létrehozásakor String objektum kerül létrehozásra.

A tömbök a C/C++-tól eltérően nem mutatók, hanem valódi típus. Nem primitív típusok, objektumhivatkozást tartalmaznak. Többdimenziós tömbök alapértelmezetten nincsenek a nyelvben, de meg lehet oldani a létrehozásokat pl.: a tömb tartalmazhat tömböket. Bizonyos értékeket felsorolási típusokban is tárolhatunk pl.: hét napjai. Ezek értékét akár tömbökben is elhelyezhetjük.

Megjegyzéseket háromféleképpen írhatunk a kódba. Ezek közül kiemelendő a dokumentációs megjegyzés, amit a javadoc alkalmazással HTML dokumentációvá tehetünk. A megjegyzések minden fajtáját figyelmen kívül hagyja a fordító.

Osztályokat a Class kulcsszóval hozhatunk létre, ezekhez adattagok és metódusok tartoznak, ezeket más nyelvekben változóknak és függvényeknek nevezzük. A Java-ban az objektumokat az new operátorral példányosíthatunk. pl. Osztály objektum1 = new Osztály (). Ilyenkor memóriát foglalunk, ezt a new operátor tárolja. Az osztály változó pedig egy referenciát tartalmaz az objektumról, ami ténylegesen az objektumra mutat nem pedig a memóriában lévő címre. Ellentétben például a C-vel, ahol mutatókat alkalmazunk, amik adott memóriacímre utalnak. Az osztály típusú változók egyben referencia típusú változók is. Ha használjuk a final kulcsszót az osztályváltozó előtt akkor minidig csak ugyanarra az objektumra fog hivatkozni, viszont a tulajdonságait változtathatjuk. A null referenciaérték egyetlen objektumra sem mutat, ez a referencia típusú változók alapértelmezett értéke. Ha egy objektumot törlni szeretnénk azt más nyelvekben általában manuálisan kell megtennünk. Java-ban, ha a rendszer látja, hogy nincs az objektumra hivatkozva, akkor automatikusan törlődik. Ezt a szemétgyűjtő mechanizmus végzi. Egy hivatkozás akkor szabadul fel, ha változója új értéket kap vagy törlődik.

A kivételek kezelésére a try-catch blokkot használjuk, mint a C-ben. Maga a program a try blokkban van, ha futás közben hibát talál a fordító a try blokkban a catch blokkra ugrik a vezérlés. A catch blokknak dobott hiba is egy objektum típusa a hiba fajtája, adattagjai pedig a hiba részletei.

A Java-ban a műveletek ugyanolyan sorrendben értékelődnek ki (balról jobbra), mint a C-ben és az operátorok is nagyjából megegyeznek. A logikai műveleteknek két fajtája van: a mohó és a lusta kiértékelésű. A lusta változatban az operandusok nem minden kerülnek kiértékelésre pl.: és művelet első tagja hamis.

Típuskonverzióra a nyelvben háromféle lehetőség van. Ha többféle primitív típussal végzünk műveletet akkor automatikus a konverzió automatikus, az eredmény minden olyan típusú lesz amelyik nagyobb tárhelyű. Explicit konverzióra akkor van szükség, ha alaptípusok között a kisebb értelmezési tartományra szeretnénk váltani, illetve objektumok referenciatípusa esetén, ha a statikus és dinamikus típusa nem egyezik meg. Azaz, ha a deklarálásnál megadott típus különbözik annak az objektumnak a típusától amire hivatkozik. A harmadik típus a szövegkonverzió, amikor egy nem String típust String típusra konvertálunk, vagy konkatenációt használunk.

Ahhoz, hogy elérjünk egy adott struktúra egy részelemét minden esetben a „.” -ot kell használni, nincs megkülönböztetett jelölés az osztálytagok elérésére mint C++-ban a „::” operátor. Az utasításoknak két fajtája van: a kifejezés (értékkedás, metódushívások stb.), és a deklarációs-utasítások. Elágazásoknak két fajtáját különböztetjük meg: az egyszerűt (if) és összetettet (switch). Ezek ugyanúgy működnek, mint C-ben/C++-ban. Ciklusok tekintetében a jól ismert fajták (while, do-while, for) itt is megtalálhatók. A bejáró ciklus egy különleges for ciklus, ami adatszerkezetek bejárására használható. A címkek, break, continue utasítások ugyanazok, mint C-ben viszont a Java elhagyta a nem túl biztonságos goto utasítást.

DRAFT

9. fejezet

Helló, Arroway!

9.1. OO szemlélet

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/prog2/00%20szeml%C3%A9let>

Tanulságok, tapasztalatok, magyarázat... : A polártranszformációs algoritmus egy random szám generátor, ami 2 véletlen értéket hoz létre. Egy lépéssel két normál eloszlású számot állít elő. A páratlanadik számításnál nem kell újraszámolni, csak az előző lépés számát kell visszaadni.

Javában PolárGenerátor osztályt használunk melynek tagjai egy bool nincsTárolt és egy double tárolt változók melyek azt tárolják el, hogy van-e eltártolt szám és ha van mi az. A konstruktőrben igazra állítjuk be a bool értékét, azaz még nincsen eltárolt számunk. Ha nincs eltárolt számunk az algoritmus végbe megy a második érték bekerül a tárolt változóba és a nincsTárolt értéke hamisra vált visszatérési értékként pedig az első értéket kapjuk. Ha volt tárolt számunk a nincsTárolt-at negáljuk, visszaadja a metódus az eltárolt számot, az algoritmus generáló része nem fut le. A main-ben objektumot példányosítunk PolárGenerátor néven és ebben a következő() metódussal állítjuk elő a számokat.

```
public class PolárGenerátor {  
    boolean nincsTárolt = true;  
    double tárolt;  
  
    public PolárGenerátor() {  
        nincsTárolt = true;  
    }  
  
    public double következő() {  
        if(nincsTárolt) {  
            double u1, u2, v1, v2, w;  
            do{  
                u1 = Math.random();  
                u2 = Math.random();  
                v1 = 2 * u1 - 1;  
                v2 = 2 * u2 - 1;  
                w = v1 * v1 + v2 * v2;  
            } while (w > 1);  
            nincsTárolt = false;  
            tárolt = w;  
        } else {  
            return tárolt;  
        }  
    }  
}
```

```
        double r = Math.sqrt((-2 * Math.log(w)) / w);
        tárolt = r * v2;
        nincsTárolt = !nincsTárolt;
        return r * v1;
    } else {
        nincsTárolt = !nincsTárolt;
        return tárolt;
    }
}

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

C++-ban a PolarGen osztályt header fájlba írjuk meg. Az osztály public részében elhelyezett konstruktur beállítja, hogy kezdésnek nincsen eltárolt változtó, meghívja a srand() függvényt, ami a számgenerálásnál lesz fontos az algoritmus során. Egy destruktor is található benne és deklarálja a kovetkezo() függvényt. A private részben tároljuk el a változókat, mivel nem kell közvetlenül elérnünk őket, elég a függvényen keresztül.

```
#ifndef POLARGEN__H
#define POLARGEN__H

#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iostream>

class PolarGen
{
public:
    PolarGen ()
    {
        nincsTarolt = true;
        std::srand (std::time (NULL));
    }
    ~PolarGen ()
    {
    }
    double kovetkezo ();

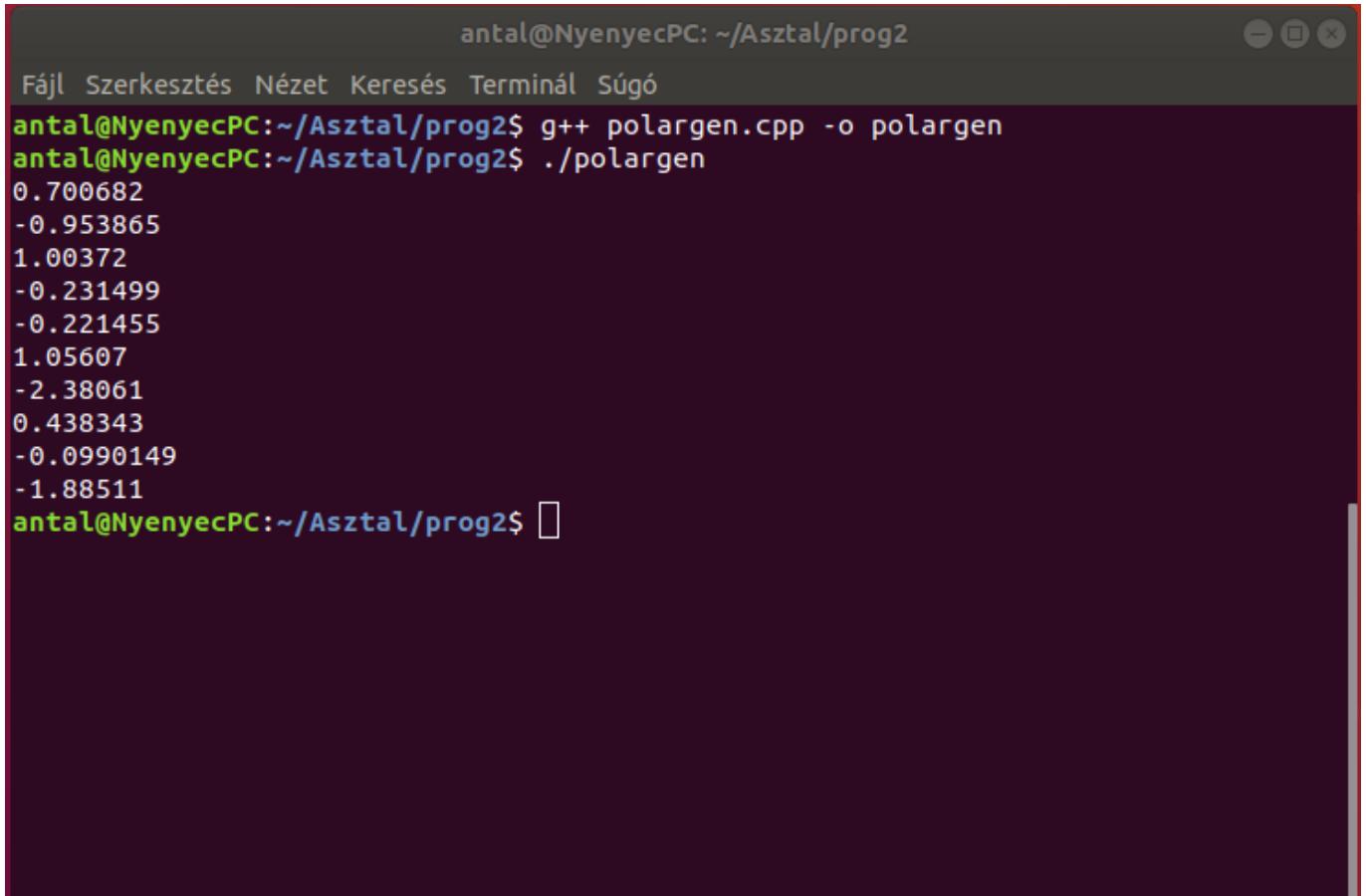
private:
    bool nincsTarolt;
```

```
    double tarolt;  
};  
#endif
```

Magában a program részben használjuk a :: operátort, mivel ez a PolarGen osztályon belüli függvény. A függvény a Java-hoz hasonló módon üzemel, C++-os függvényekkel módosítva pár helyen. A main-ben példányosítás zajlik, meghívjuk a kovetkezo() függvényt egy for-ban.

```
#include "polargen.h"  
#include <iostream>  
  
double PolarGen::kovetkezo()  
{  
    if(nincsTarolt) {  
        double u1,u2,v1,v2,w;  
        do{  
            u1 = std::rand() / (RAND_MAX + 1.0);  
            u2 = std::rand() / (RAND_MAX + 1.0);  
            v1 = 2*u1 -1;  
            v2 = 2*u2 -1;  
            w = v1 * v1 + v2 *v2;  
        }  
        while(w>1);  
  
        double r = std::sqrt ((-2 * std::log(w)) /w);  
  
        tarolt = r*v2;  
        nincsTarolt = !nincsTarolt;  
  
        return r * v1;  
    }  
    else{  
        nincsTarolt = !nincsTarolt;  
        return tarolt;  
    }  
}  
  
int main(int argc,char **argv)  
{  
    PolarGen pg;  
  
    for(int i=0;i<10;++i)  
        std::cout<<pg.kovetkezo()<<std::endl;
```

```
    return 0;  
}
```



```
antal@NyenyecPC: ~/Asztal/prog2  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
antal@NyenyecPC:~/Asztal/prog2$ g++ polargen.cpp -o polargen  
antal@NyenyecPC:~/Asztal/prog2$ ./polargen  
0.700682  
-0.953865  
1.00372  
-0.231499  
-0.221455  
1.05607  
-2.38061  
0.438343  
-0.0990149  
-1.88511  
antal@NyenyecPC:~/Asztal/prog2$
```

9.2. "Gagyí"

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/prog2/gagyi>

Tanulságok, tapasztalatok, magyarázat... : A feladat az volt, hogy írunk olyan programot, ami a(z) ($x \leq t \& \& x \geq t \& \& t \neq x$) feltételre egyszer végtelen ciklust adjon, más értékeknél pedig nem.

Végtelen ciklus:

```
public class Gagy2  
{  
public static void main (String[] args)  
{  
Integer x = -129;  
Integer t = -129;  
System.out.println (x);  
System.out.println (t);  
while (x <= t && x >= t && t != x);  
}  
}
```

Lefut:

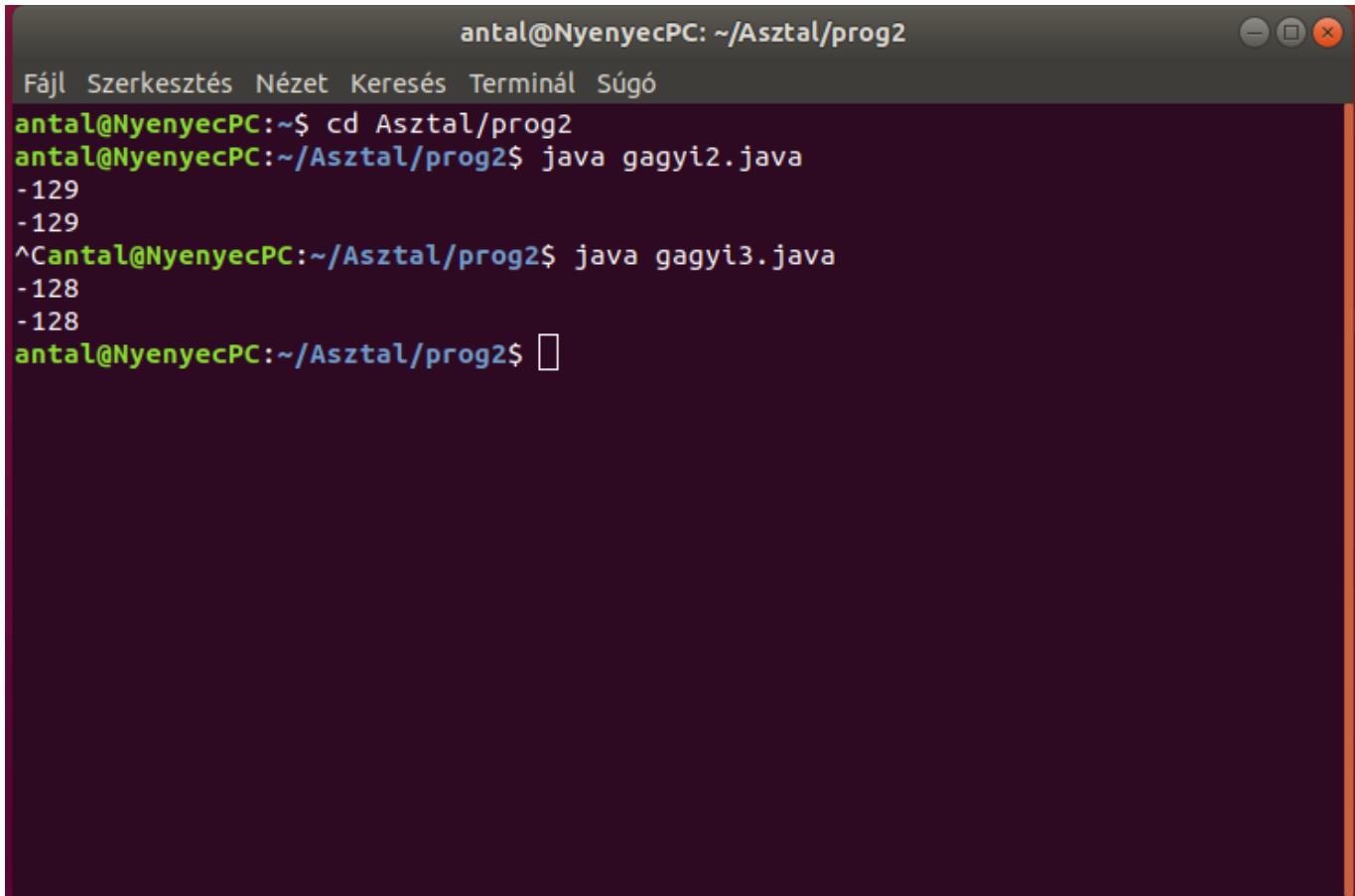
```
public class Gagy13
{
    public static void main (String[] args)
    {
        Integer x = -128;
        Integer t = -128;
        System.out.println (x);
        System.out.println (t);
        while (x <= t && x >= t && t != x);
    }
}
```

Az Integer -128-127-ig nem példányosít új objektumot, hanem előre meglévőket társít a számokhoz. Futásnál a két érték ugyanabból a cahce-ből szedett objektumot használja. Ez azt jelenti, hogy pl. a Integer.valueOf(20) mindenkor ugyanazt az objektumot fogja visszadni nem úgy, mint a következő esetben. A -129-es esetben, mivel ez már a tartományon kívül esik, különböző objektumot kapunk 2 változónak. Mivel ezek a memóriában másolat helyezkednek el, összehasonlításuknál nem kapunk megegyezést. A gyorsítótár működés közbeni megtékinésének módja egyszerű két kicsi egész számot hasonlítunk össze olyan számokkal, amik a gyorsító hatókörén kívülre esnek.

```
Integer.valueOf(17) == Integer.valueOf(17)
```

```
Integer.valueOf(200) != Integer.valueOf(200)
```

A fenti kódcsipetek igazak.



The screenshot shows a terminal window titled "antal@NyenyecPC: ~/Asztal/prog2". The window has standard Linux-style window controls (minimize, maximize, close) in the top right corner. The terminal content is as follows:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
antal@NyenyecPC:~$ cd Asztal/prog2
antal@NyenyecPC:~/Asztal/prog2$ java gagyi2.java
-129
-129
^Cantal@NyenyecPC:~/Asztal/prog2$ java gagyi3.java
-128
-128
antal@NyenyecPC:~/Asztal/prog2$
```

9.3. Yoda

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/prog2/yoda.java>

Tanulságok, tapasztalatok, magyarázat... : A Yoda conditions vagy más néven Yoda notation egy programozó stílus, ahol egy kifejezés két részét megfordítjuk az utasításban a megszokott sorrendtől. A Yoda feltétel a kifejezés állandó részét a feltételes kijelentés bal oldalára helyezi. Ennek a stílusnak a neve a Yoda nevű Star Wars karakterből származik, mivel ő nem szabványos szintaxisával beszéli az angolt. A Yoda feltételek a Symfony kódolási szabványok és a WordPress kódolási szabványok részét képzi.

Általában egy feltételes kifejezést ilyen formában adunk meg: (if \$value == 42) "Ha az érték egyenlő 42-vel.."

Ám ez a Yoda feltétel szerint így néz ki: (if 42 == \$value) "Ha 42 egyenlő az értékkkel.."

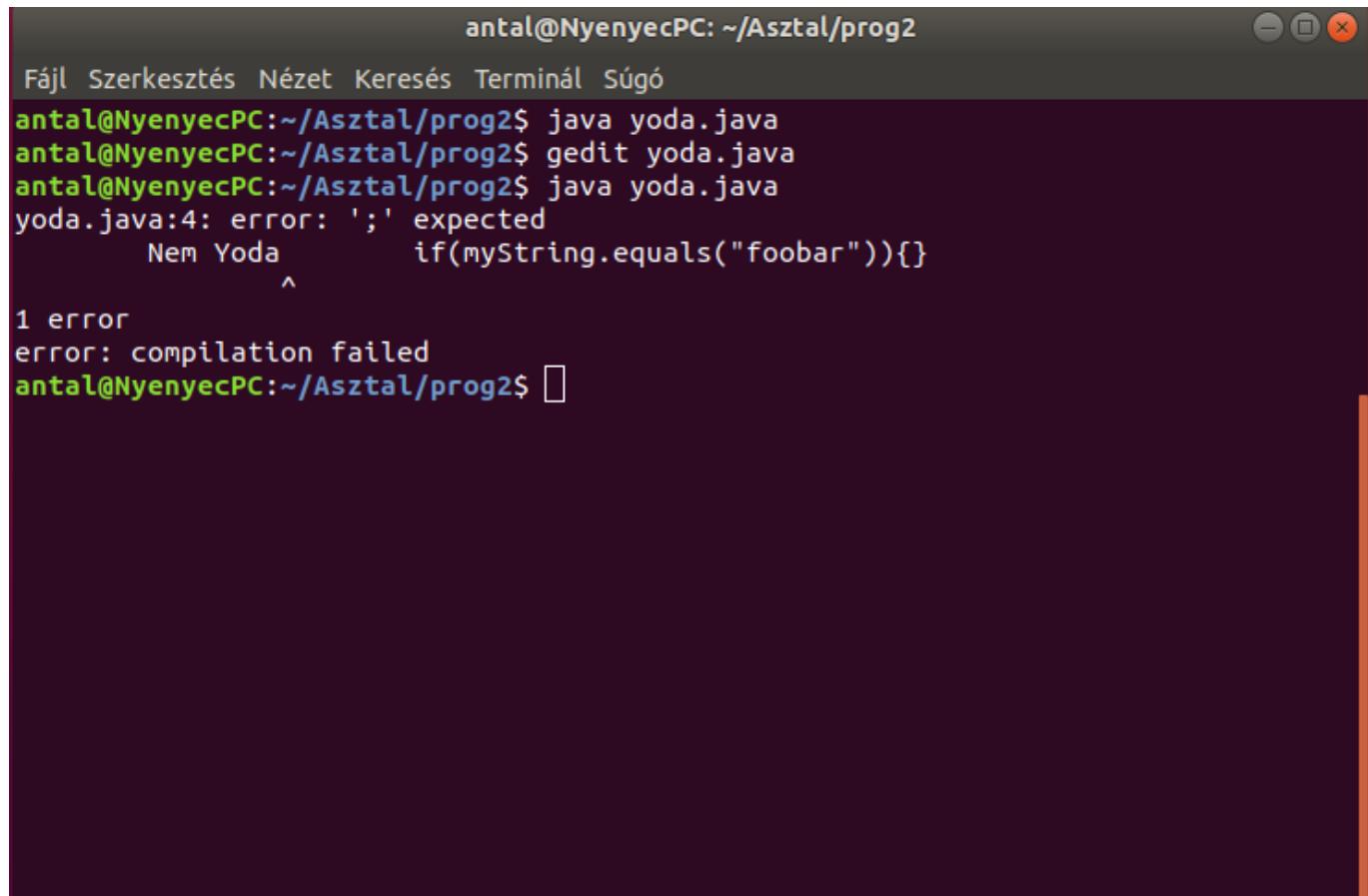
Az állandót az összehasonlító operátor bal oldalára írjuk, a változót, amelynek értékét az állandóhoz viszonyítjuk, jobbra írjuk. Ez a sorrend összehasonlítható Yoda nem szabványos angol nyelvű stílusával, amely nagyjából: (tárgy-alany-ige) pl.: "When nine hundred years old you reach, look as good you will not." ("Ha eléréd a kilencszáz éves kort, olyan jól kinézni nem fogsz.")

Az állandó érték behelyezése a kifejezésbe nem változtatja meg a program viselkedését.(kivéve, ha az értékek hamisak lesznek)

Azokban a nyelvekben ahol az 1db (=) jelet a hozzárendeléshez és nem az összehasonlításhoz használják, lehetséges hiba az, hogy egy értéket akaratlanul rendelnek hozzá, ahelyett, hogy feltételes kifejezést

írnának.

A program:



The screenshot shows a terminal window titled "antal@NyenyecPC: ~/Asztal/prog2". The window has standard Linux-style window controls (minimize, maximize, close) at the top right. The terminal content is as follows:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
antal@NyenyecPC:~/Asztal/prog2$ java yoda.java
antal@NyenyecPC:~/Asztal/prog2$ gedit yoda.java
antal@NyenyecPC:~/Asztal/prog2$ java yoda.java
yoda.java:4: error: ';' expected
    Nem Yoda      if(myString.equals("foobar")){}
                           ^
1 error
error: compilation failed
antal@NyenyecPC:~/Asztal/prog2$
```

```
public class Yoda{
    public static void main(String[] args) {
        String myString = null;
        //Nem Yoda  if(myString.equals("foobar")) {}

        if("foobar".equals(myString)) {} //Yoda
    }
}
```

A Yoda feltételt használva nem kapunk hibát a programban, mert a Stringre hivatkozik az equals, ami ugyan üres, de nem null referencia.

9.4. Kódolás from scratch

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei

Tanulságok, tapasztalatok, magyarázat... : Meg kellet írni a BBP algoritmust. A main-ben lévő k változó értéke azt fogja nekünk megadni, hogy a pi-nek 10 a hanyadikon számjegyétől kezdve írjuk ki a következő

számjegyeket hexadecimális alakban. Jelen esetben 6 az értéke vagyis 10 a hatodikon utáni számjegyeket kapjuk meg. Kiíratjuk a magic(k) függvény eredményét. A madic függvény értékadással kezdődik. Az s1,s4,s5,s6 változóknak az értékét úgy adjuk meg, hogy meghívjuk a solve() függvényt, ami a mod() függvényt fogja használni. A solve() egyik paramétere 10 a k-adikon lesz, a másik egy szám. Ha az értékadás megvolt, ezeket ráeresztjük a cut() függvényünkre. Ennek feladata, hogy visszaadja a paraméterként kapott double változó nem egész részét. Úgy fog ez megvalósulni, hogy ha az értéke negatív, akkor saját magához hosszáadja a saját maga egész részét, ha pozitív akkor kivonja. Ezután jöhet a pi létherozzuk és értékkel látjuk el. Kiszámoljuk ennek a nem egész részét és létrehozzuk a hexadecimális jeleket és a végeredményt. egy while-ban addig számoljuk a pi értékét, amíg a nem egész részének az értéke nem lesz egyenlő 0-val. Ha nem egyenlő, akkor pi értékét megszorozzuk 16-tal. Ha az egész része nagyobb vagy egyenlő 10-nél, akkor a végeredményt koncatenáljuk az értéknek megfelelő hexadecimális jellel. Egyébként a string-é alakított számjegyeket koncatenáljuk a végeredménnyel, majd pedig elveszszük a pi-ból az egész részét és kezdődik előről a ciklus. A legvégén visszaadjuk az eredményt. A solve() egy összeget számol. Egy for addig megy, amíg az első kapott paraméter vagyis a d értéke nagyobb, vagy egyenlő i-vel. A cikluson belül pedig lépésenként hozzáadja az összeg értékéhez a mod() függvény által kiszámolt értéket. A mod() két változót hoz létre t és r néven, mind kettőnek a kezdőértéke 1 lesz. Egy while ciklus addig fog menni, amíg t kisebb vagy egyenlő n-el. "n" a második paramétere a függvénynek. A cikluson belül minden iterációban t értéke meg lesz szorozva 2-vel. Következik méggy while, ami a break-el leáll. A függvény visszaadja az r értékét.

```
public class BBP {
    public static void main(String[] args) {
        int k = 6;
        System.out.println(magic(k));
    }

    public static String magic(int k) {
        double s1 = solve(Math.pow(10,k), 1);
        double s4 = solve(Math.pow(10,k), 4);
        double s5 = solve(Math.pow(10,k), 5);
        double s6 = solve(Math.pow(10,k), 6);

        s1 = cut(s1);
        s4 = cut(s4);
        s5 = cut(s5);
        s6 = cut(s6);

        double pi = 4*s1 -2*s4 - s5 -s6;

        pi = cut(pi);

        String[] hexa = {"A", "B", "C", "D", "E", "F"};
        String result = "";

        while(cut(pi) != 0) {
            pi = pi*16;
            if((int)pi >= 10) {
                result = result.concat(hexa[(int)pi - 10]);
            }
        }
    }
}
```

```
        else {
            result = result.concat(Integer.toString((int)pi));
        }
        pi = cut(pi);
    }
    return result;
}

public static double cut(double db) {
    if(db < 0) {
        return db - (int)db+1;
    }
    else {
        return db - (int)db;
    }
}

public static double solve(double d, double num) {
    double sum = 0.0;

    for(int i = 0; i <= d; i++) {
        sum += mod(16, (d-i), 8*i+num) / (8*i + num);
    }
    return sum ;
}

public static double mod(double b, double n, double k) {
    double t = 1;
    double r = 1;

    while(t <= n) {
        t = t * 2;
    }

    while(true) {
        if(n >= t) {
            r = (b * r) % k;
            n= n - t;
        }
        t = t / 2;
        if(t >= 1) {
            r = (r*r) % k;
        }
        else {
            break;
        }
    }
    return r;
}
}
```

A Bailey-Borwein-Plouffe formula a pi kiszámítására kötött létre. Simon Plouffe fedezte fel 1995-ben és a cikk szerkesztőiről kapta a nevét. (David H. Bailey, Peter Borwein, Plouffe)

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

A formula:

A formula használatával megadhatjuk a pi hexadecimális számjegyeit.

Futtatáskor:

```
antal@NyenyecPC:~/Asztal$ java BBP  
6C65E5308  
antal@NyenyecPC:~/Asztal$  
antal@NyenyecPC:~/Asztal$ █
```

10. fejezet

Helló, Liskov!

10.1. Liskov helyettesítés sértése

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/prog2/liskov>

Tanulságok, tapasztalatok, magyarázat... : A Liskov Substitution Principle vagyis Liskov helyettesítési elv az objektum-orientált programozásban azt állítja, hogy ha S altípusa T-nek, akkor minden olyan helyen ahol T-t felhasználjuk S-t is minden gond nélkül behelyettesíthetjük anélkül, hogy a programrész tulajdonságai megváltoznának. A feladatunk az volt, hogy ennek az elvnek a megsértésére írunk egy programot. A programunk pedig a következő képpen néz ki:

```
class Madar {  
public:  
    virtual void repul() {};  
};  
  
class Program {  
public:  
    void fgv ( Madar &madar ) {  
        madar.repul();  
    }  
};  
  
class Sas : public Madar  
{};  
  
class Pingvin : public Madar  
{};  
  
int main ( int argc, char **argv )  
{  
    Program program;  
    Madar madar;  
    program.fgv ( madar );
```

```
Sas sas;
program.fgv ( sas );

Pingvin pingvin;
program.fgv ( pingvin );

}
```

Létrehozzuk a madár osztályt, melynek alosztályai lesznek a Sas és a Pingvin. A Liskov helyettesítési elv szerint, ha a 2 osztály alosztálya a Madárnak, akkor a Madár osztály függvényét felhasználhatjuk a 2 al-osztálynál is, anélkül, hogy a programrész tulajdonsága változna. A madár osztály része a repül függvény. A sértés ott történik, amikor megakarnánk hívni a pingvin alosztályunkra a repül függvényt. A pingvinek köztudottan nem tudnak repülni. Szóval még gyakorlatban le is fut elméletben még sem jön össze a dolog.

10.2. Szülő-gyerek

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/Liskov/szulogyerek>

Tanulságok, tapasztalatok, magyarázat...: Java-ban, C++-ban az ősön keresztül csak az ős üzenetei közvetíthetőek. Amikor megpróbálnánk egy gyerek osztályban példányosított függvényt az ősosztályban lévő objektumra használni, a képen láthatóan hibaüzenetet fogunk kapni a fordítás során.

C++ kód:

```
#include<iostream>

class Madar{
public:
};

class Sas : public Madar{
public:
    void repul() {
        std::cout << "Repül";
    }
};

int main(){

    Madar* sas = new Sas();
    Sas* sas2 = new Sas();

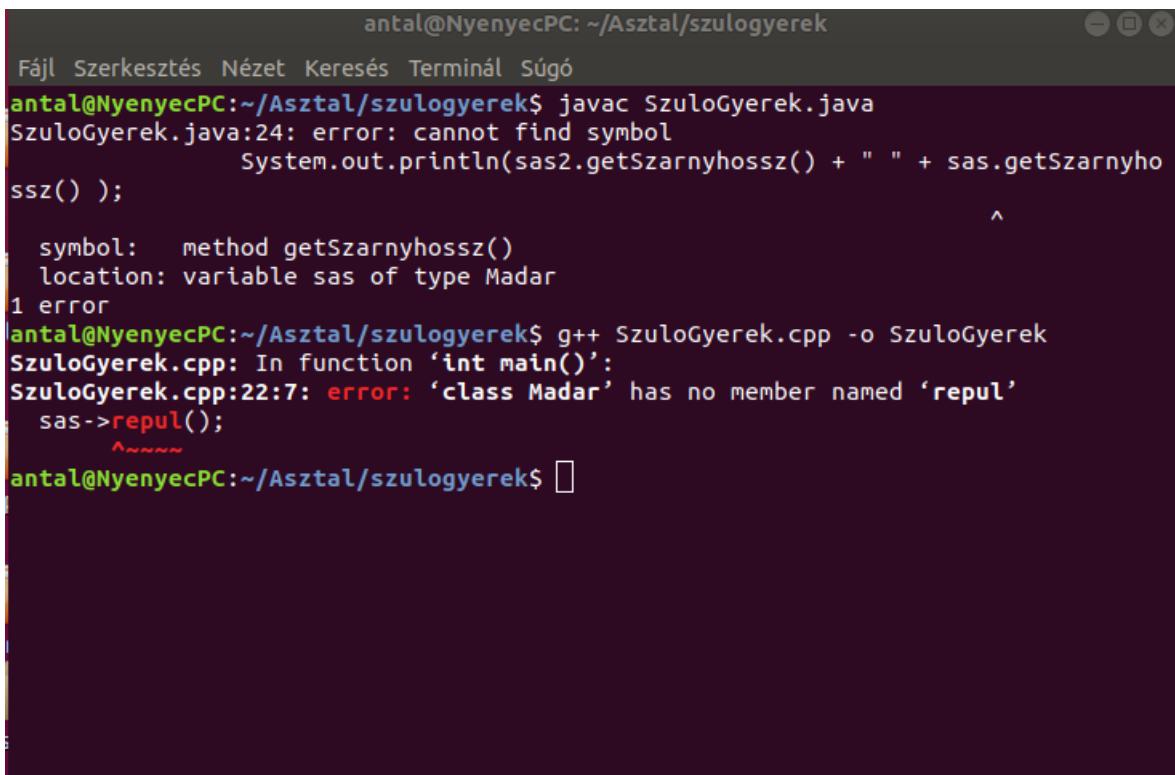
    sas->repul();
    sas2->repul();
}
```

```
}
```

Java kód:

```
public class SzuloGyerek{  
  
    static class Madar{  
        protected int szarnyhossz;  
        public void setSzarnyhossz(int szarnyhossz) {  
            this.szarnyhossz = szarnyhossz;  
        }  
  
    }  
  
    static class Sas extends Madar{  
        public int getSzarnyhossz() {  
            return szarnyhossz;  
        }  
    }  
  
    public static void main(String args[]){  
        Madar sas = new Sas();  
        sas.setSzarnyhossz(80);  
  
        Sas sas2 = new Sas();  
        sas2.setSzarnyhossz(50);  
  
        System.out.println(sas2.getSzarnyhossz() + " " + sas.getSzarnyhossz());  
    }  
}
```





The screenshot shows a terminal window with the following text:

```
antal@NyenyecPC: ~/Asztal/szulogyerek
Fájl Szerkesztés Nézet Keresés Terminál Súgó
antal@NyenyecPC:~/Asztal/szulogyerek$ javac SzuloGyerek.java
SzuloGyerek.java:24: error: cannot find symbol
        System.out.println(sas2.getSzarnyhossz() + " " + sas.getSzarnyhossz());
                                         ^
symbol:   method getSzarnyhossz()
location: variable sas of type Madar
1 error
antal@NyenyecPC:~/Asztal/szulogyerek$ g++ SzuloGyerek.cpp -o SzuloGyerek
SzuloGyerek.cpp: In function 'int main()':
SzuloGyerek.cpp:22:7: error: 'class Madar' has no member named 'repul'
    sas->repul();
          ^
antal@NyenyecPC:~/Asztal/szulogyerek$
```

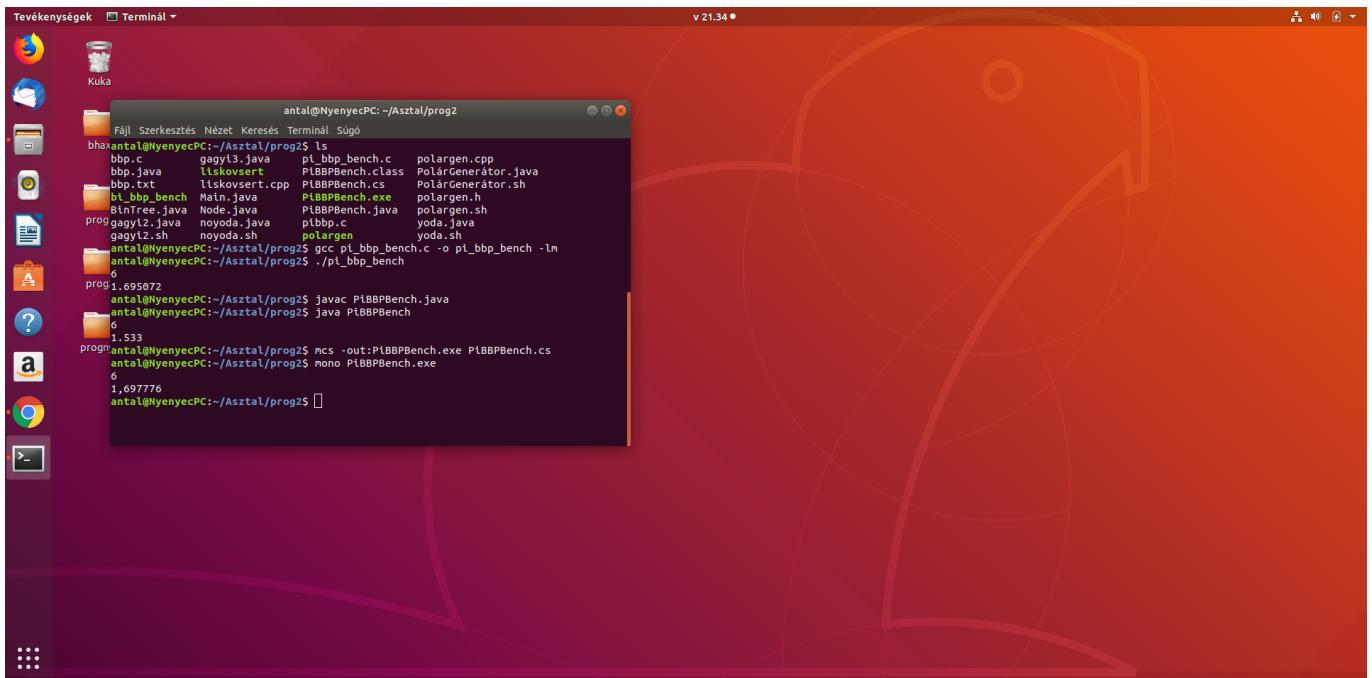
Mindkét esetben létezik egy Sas típusú Sas típus, és egy Madár típust Sas típusként. C++-ban a repul() metódussal megpróbáljuk őket reptetni, a Sas gond nélkül menni fog, de a Madár nem. A Java kódban minden típusnak beállítjuk a szárnyhossz tulajdonságát a setSzarnyhossz() segítségével, ezzel nem lesz gond mivel ez a Madár osztály metódusa, amit a Sas megörökölt. Azoban a getSzarnyhossz() metódus segítségével, amikor kiíratnánk a két objektum szárnyhosszát, akkor csak az egyik esetben fog sikerülni. Itt látszik tehát, hogy a gyerek osztályban ha létrehozunk egy metódust és az osztályt szülőosztályként akarnánk használni, akkor a gyerekosztály metódusait nem lesz módunk felhasználni.

10.3. Anti OO

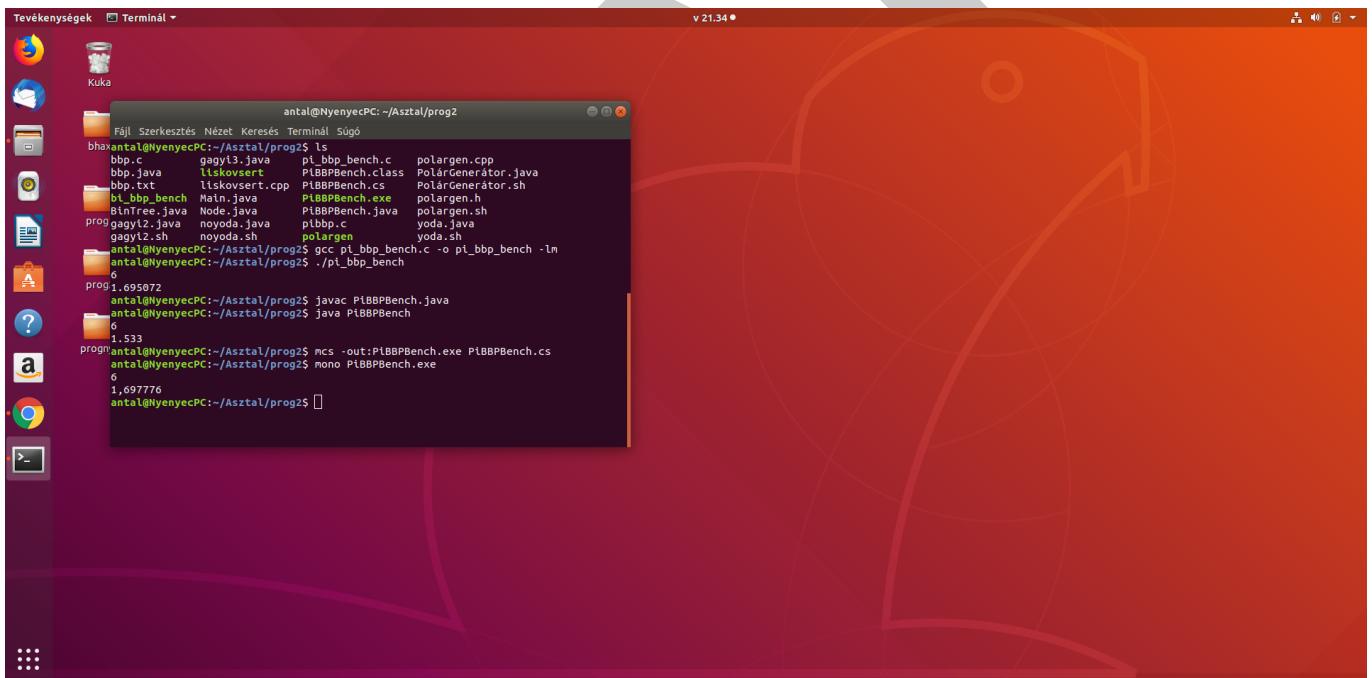
Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/prog2/PiBBP>

Tanulságok, tapasztalatok, magyarázat... : A feladatunk az volt, hogy a BBP algoritmus felhasználásával írunk egy programot Java,C illetve C# nyelveken, amik kiszámolják a Pi hexadecimális kifejezésének 0. pozíciótól számított 10 a hatodikon,hetediken,nyolcadikon db jegyét és össze kell hasonlítanunk a futási idejüköt.

10 a hatodikonnál a programok futási ideje:



A futási idő 10 a hetedikennél:



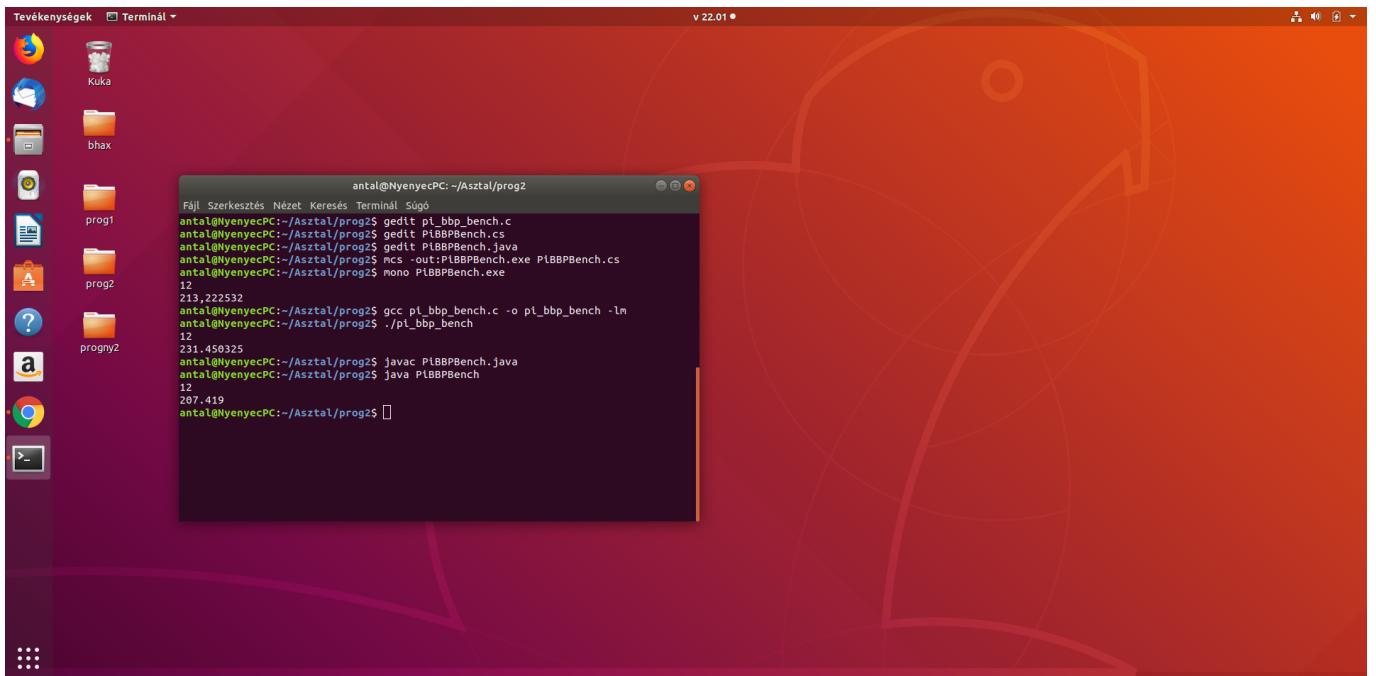
A futási idő 10 a nyolcadikonnál:

The image displays two terminal windows on a Linux desktop. The top terminal window shows the following command-line session:

```
antal@NyenyecPC:~/Asztal/prog2$ javac PiBBPBench.java
antal@NyenyecPC:~/Asztal/prog2$ java PiBBPBench.cs
0
1.533
antal@NyenyecPC:~/Asztal/prog2$ mcs -out:PiBBPBench.exe PiBBPBench.cs
antal@NyenyecPC:~/Asztal/prog2$ mono PiBBPBench.exe
prog6
1,697776
17.996
antal@NyenyecPC:~/Asztal/prog2$ gedit PiBBPBench.java
antal@NyenyecPC:~/Asztal/prog2$ gedit pi_bb_bench.c
antal@NyenyecPC:~/Asztal/prog2$ gedit PiBBPBench.cs
prog7
antal@NyenyecPC:~/Asztal/prog2$ javac PiBBPBench.java
antal@NyenyecPC:~/Asztal/prog2$ java PiBBPBench
7
20.04588
antal@NyenyecPC:~/Asztal/prog2$ mcs -out:PiBBPBench.exe PiBBPBench.cs
antal@NyenyecPC:~/Asztal/prog2$ mono PiBBPBench.exe
7
18,33613
antal@NyenyecPC:~/Asztal/prog2$
```

The bottom terminal window shows the following command-line session:

```
antal@NyenyecPC:~/Asztal/prog2$ ll
bbp.java  llsvksovert.cpp  PiBBPBench.cs  polargen.h
bbp.txt  Main.java  PiBBPBench.exe  polargen.sh
bt_bb_bench  Node.java  PiBBPBench.java  yoda.java
btNtree.java  noyoda.java  piBBP.c  yoda.sh
gagyl2.sh  pi_bb_bench  polargen.cpp
gagyl3.java  pi_bb_bench.c  PolárGenerátor.java
antal@NyenyecPC:~/Asztal/prog2$ gedit pi_bb_bench.c
antal@NyenyecPC:~/Asztal/prog2$ gedit piBBP.cpp
antal@NyenyecPC:~/Asztal/prog2$ g++ piBBP.cpp -o piBBP
antal@NyenyecPC:~/Asztal/prog2$ ./piBBP
12
373.662203
antal@NyenyecPC:~/Asztal/prog2$ gedit piBBP.cpp
antal@NyenyecPC:~/Asztal/prog2$ g++ piBBP.cpp -o piBBP
antal@NyenyecPC:~/Asztal/prog2$ ./piBBP
7
25.614153
antal@NyenyecPC:~/Asztal/prog2$ gedit piBBP.cpp
antal@NyenyecPC:~/Asztal/prog2$ g++ piBBP.cpp -o piBBP
antal@NyenyecPC:~/Asztal/prog2$ ./piBBP
6
2.278021
antal@NyenyecPC:~/Asztal/prog2$
```



Ahogy az a táblázatból is kivehető összesítésben a Java volt a leggyorsabb és a C++ a leglassabb.

A kód C-ben:(A C++ kód megegyezik a C-ben írttal)

```
#include <stdio.h>
#include <math.h>
#include <time.h>

long
n16modk (int n, int k)
{
    long r = 1;

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

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

        t = t / 2;

        if (t < 1)
            break;
    }
}
```

```
r = (r * r) % k;
}

return r;
}

double
d16Sj (int d, int j)
{
    double d16Sj = 0.0;
    int k;

    for (k = 0; k <= d; ++k)
        d16Sj += (double) n16modk (d - k, 8 * k + j) / (double) (8 * k + j);

    return d16Sj - floor (d16Sj);
}

int main ()
{
    double d16Pi = 0.0;

    double d16S1t = 0.0;
    double d16S4t = 0.0;
    double d16S5t = 0.0;
    double d16S6t = 0.0;

    int jegy;
    int d;

    clock_t delta = clock ();

    for (d = 1000000; d < 1000001; ++d)
    {
        d16Pi = 0.0;

        d16S1t = d16Sj (d, 1);
        d16S4t = d16Sj (d, 4);
        d16S5t = d16Sj (d, 5);
        d16S6t = d16Sj (d, 6);

        d16Pi = 4.0 * d16S1t - 2.0 * d16S4t - d16S5t - d16S6t;
    }
}
```

```
d16Pi = d16Pi - floor (d16Pi);

jegy = (int) floor (16.0 * d16Pi);

}

printf ("%d\n", jegy);
delta = clock () - delta;
printf ("%f\n", (double) delta / CLOCKS_PER_SEC);
}
```

A kód Java-ban:

```
public class PiBBPBench {

    public static double d16Sj(int d, int j) {

        double d16Sj = 0.0d;

        for(int k=0; k<=d; ++k)
            d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);

        /* (bekapcsolva a sorozat elején az első utáni jegyekben növeli pl.
           a pontosságot.)
        for(int k=d+1; k<=2*d; ++k)
            d16Sj += Math.pow(16.0d, d-k) / (double)(8*k + j);
        */

        return d16Sj - Math.floor(d16Sj);
    }

    public static long n16modk(int n, int k) {

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

        long r = 1;

        while(true) {

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

            t = t/2;

            if(t < 1)
                break;
        }
    }
}
```

```
r = (r*r) % k;

}

return r;
}

public static void main(String args[]) {

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
    double d16S6t = 0.0d;

    int jegy = 0;

    long delta = System.currentTimeMillis();

    for(int d=1000000; d<1000001; ++d) {

        d16Pi = 0.0d;

        d16S1t = d16Sj(d, 1);
        d16S4t = d16Sj(d, 4);
        d16S5t = d16Sj(d, 5);
        d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - Math.floor(d16Pi);

        jegy = (int)Math.floor(16.0d*d16Pi);

    }

    System.out.println(jegy);
    delta = System.currentTimeMillis() - delta;
    System.out.println(delta/1000.0);
}
}
```

A kód C#-ban:

```
public class PiBBPBench {

    public static double d16Sj(int d, int j) {

        double d16Sj = 0.0d;
```

```
for(int k=0; k<=d; ++k)
    d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);

/*
for(int k=d+1; k<=2*d; ++k)
    d16Sj += System.Math.Pow(16.0d, d-k) / (double)(8*k + j);
*/

return d16Sj - System.Math.Floor(d16Sj);
}

public static long n16modk(int n, int k) {

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

    long r = 1;

    while(true) {

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

        t = t/2;

        if(t < 1)
            break;

        r = (r*r) % k;
    }

    return r;
}

public static void Main(System.String[] args) {

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
    double d16S6t = 0.0d;

    int jegy = 0;

    System.DateTime kezd = System.DateTime.Now;
```

```
for(int d=1000000; d<1000001; ++d) {  
  
    d16Pi = 0.0d;  
  
    d16S1t = d16Sj(d, 1);  
    d16S4t = d16Sj(d, 4);  
    d16S5t = d16Sj(d, 5);  
    d16S6t = d16Sj(d, 6);  
  
    d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;  
  
    d16Pi = d16Pi - System.Math.Floor(d16Pi);  
  
    jegy = (int)System.Math.Floor(16.0d*d16Pi);  
  
}  
  
System.Console.WriteLine(jegy);  
System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);  
System.Console.WriteLine(delta.TotalMilliseconds/1000.0);  
}  
}
```

10.4. Ciklomatikus komplexitás

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Welch/z3a7.cpp>

Tanulságok, tapasztalatok, magyarázat... :

A ciklomatikus komplexitás egy szoftvermetrika, amit Thomas J. McCabe adott ki 1976-ban. A metrika egy adott program forráskódja alapján határozza meg annak komplexitását, egy konkrét számban kifejezve. A komplexitás számítása a gráfelméletre alapul. A forráskódban az elágazásokból felépülő gráf pontjai, és a köztük lévő élek alapján számítható. $(M = E - N + 2P)$ Az LZWBinfá C++-ban megírt kódjának ciklomatikus komplexitását vizsgáltam meg.

Function Name	NLOC	Complexity	Token #	Paramet #
LZWBinFa::LZWBinFa	3	1	11	
LZWBinFa::~LZWBinFa	5	1	23	
LZWBinFa::operator < <	29	4	108	
LZWBinFa::kiir	5	1	20	
LZWBinFa::operator < <	5	1	26	
LZWBinFa::kiir	5	1	22	
LZWBinFa::Csomopont::Csomopont	3	1	24	
LZWBinFa::Csomopont::~Csomopont	3	1	5	
LZWBinFa::Csomopont::nullasGyermek	4	1	9	
LZWBinFa::Csomopont::egyesGyermek	4	1	9	
LZWBinFa::Csomopont::ujNullasGyermek	4	1	12	
LZWBinFa::Csomopont::ujEgyesGyermek	4	1	12	
LZWBinFa::Csomopont::getBetu	4	1	9	
LZWBinFa::kiir	13	3	94	
LZWBinFa::szabadit	9	2	37	
LZWBinFa::getMelyseg	6	1	25	
LZWBinFa::getAtlag	7	1	36	
LZWBinFa::getSzoras	12	2	68	
LZWBinFa::rmelyseg	12	3	52	
LZWBinFa::ratlag	15	4	69	
LZWBinFa::rszoras	15	4	81	
usage	4	1	18	
main	60	13	335	

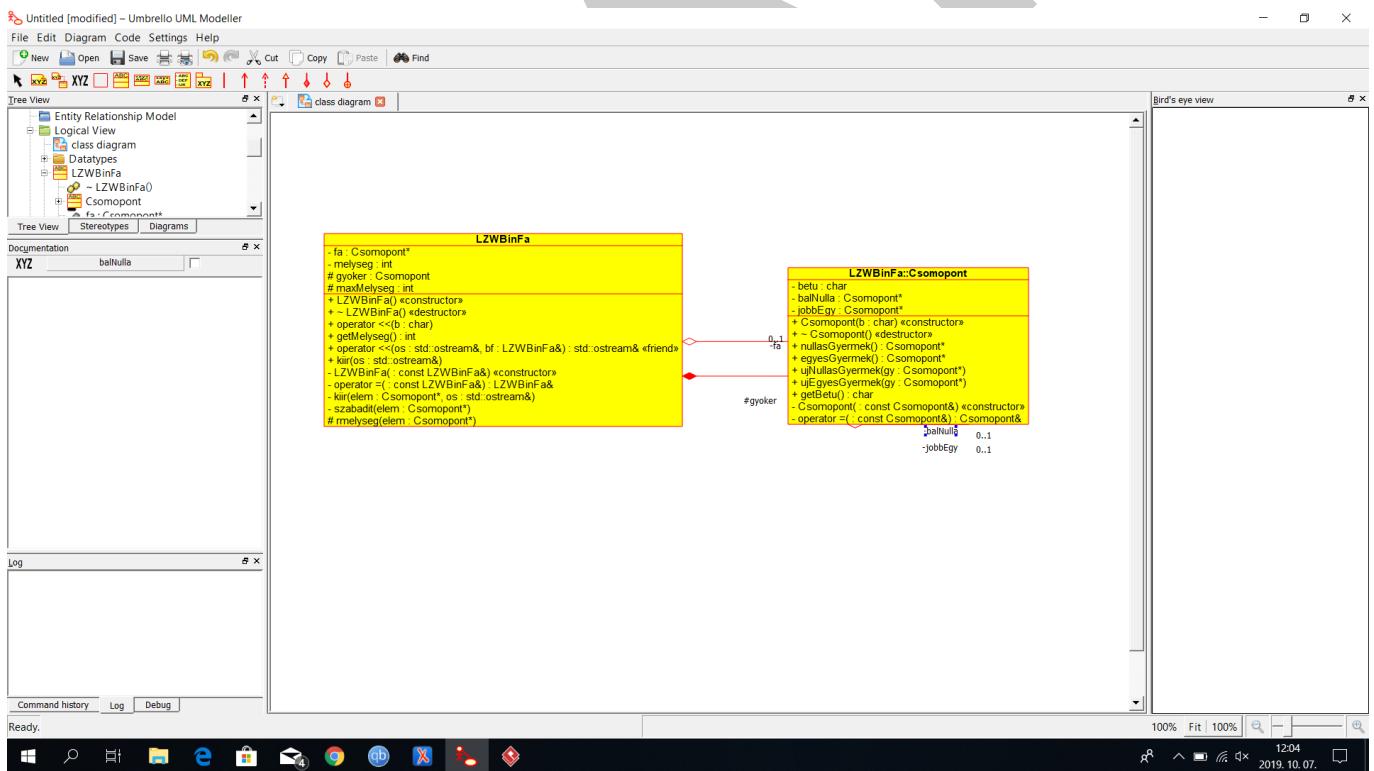
11. fejezet

Helló, Mandelbrot!

11.1. Reverse engineering UML osztálydiagram

Forrás:https://gitlab.com/baluka94/bhax/blob/master/Mandel/Welch_binfa_osztaly.cpp

Az UML diagramot az első védési program vagyis a BinFa alapján kellett elkészíteni.



Az UML legenerálásához Umbrellót használtam. Az aggregáció egy olyan asszociáció, ami tartalmazást jelöl.(üres rombusz) A kompozíció jelölése a tele rombusz, abban különbözik, hogy a tartalmazott objektum nem lehet megosztva, ez esetén a tartalmazó és a tartalmazott együtt jelenik meg illetve szűnik meg. A tartalmazottnak magában nincs értelme. Ebben a kódban a kompozíció a gyökér elem. (Csonopont gyoker;)//asszociációról még

11.2. Forward engineering UML osztálydiagram

Forrás: <https://gitlab.com/baluka94/bhax/tree/master/Mandel>

A BinFa diagramjából generáltam vissza magát a kódot:

```
#include "LZWBinFa.h"

// Constructors/Destructors
//

LZWBinFa::LZWBinFa () {
    initAttributes();
}

LZWBinFa::~LZWBinFa () { }

//
// Methods
//

// Accessor methods
//

// Other methods
//

void LZWBinFa::initAttributes () {
}
```

```
#ifndef LZWBINFA_H
#define LZWBINFA_H

#include <string>

/**
 * class LZWBinFa
 * Lérehozunk még egy destruktort is amiben meghívjuk a szabadít() ↫
 * függvényt a
 * gyoker egyes és nullas gyermekéire
 */

class LZWBinFa
{
public:

    // Constructors/Destructors
```

```
//  
  
/**  
 * Empty Constructor  
 */  
LZWBinFa ();  
  
/**  
 * Empty Destructor  
 */  
virtual ~LZWBinFa ();  
  
// Static Public attributes  
//  
  
// Public attributes  
//  
  
// Public attribute accessor methods  
//  
  
// Public attribute accessor methods  
//  
  
/**  
 * @param b  
 */  
void operator_ (char b)  
{  
}  
  
/**  
 * @return int  
 */  
int getMelyseg ()  
{  
}  
  
/**  
 * @return std::ostream&  
 * @param os  
 * @param bf  
 */
```

```
std::ostream& operator_ (std::ostream& os, LZWBinFa& bf)
{
}

/***
 * @param os
 */
void kiir (std::ostream& os)
{
}

protected:

// Static Protected attributes
//

// Protected attributes
//

// A fában a gyökér csomópont ki van tüntetve
Csomopont gyoker;
int maxMelyseg;

public:

// Protected attribute accessor methods
//

protected:

public:

// Protected attribute accessor methods
//

/***
 * Set the value of gyoker
 * A fában a gyökér csomópont ki van tüntetve
 * @param new_var the new value of gyoker
 */
void setGyoker (Csmopont new_var)      {
    gyoker = new_var;
}

/***
 * Get the value of gyoker
 * A fában a gyökér csomópont ki van tüntetve
```

```
* @return the value of gyoker
*/
Csomopont getGyoker () {
    return gyoker;
}

/**
 * Set the value of maxMelyseg
 * @param new_var the new value of maxMelyseg
 */
void setMaxMelyseg (int new_var) {
    maxMelyseg = new_var;
}

/**
 * Get the value of maxMelyseg
 * @return the value of maxMelyseg
 */
int getMaxMelyseg () {
    return maxMelyseg;
}
protected:

/**
 * @param elem
 */
void rmelyseg (Csmopont* elem)
{
}

private:
// Static Private attributes
//

// Private attributes
//

// Mindig az algoritmus szerint megfelelő Csomopontra mutat
Csmopont* fa;
int melyseg;
public:
// Private attribute accessor methods
//

private:
```

```
public:

// Private attribute accessor methods
//


/***
 * Set the value of fa
 * Mindig az algoritmus szerint megfelelő Csomopontra mutat
 * @param new_var the new value of fa
 */
void setFa (Csomopont* new_var)      {
    fa = new_var;
}

/***
 * Get the value of fa
 * Mindig az algoritmus szerint megfelelő Csomopontra mutat
 * @return the value of fa
 */
Csomopont* getFa ()      {
    return fa;
}

/***
 * Set the value of melyseg
 * @param new_var the new value of melyseg
 */
void setMelyseg (int new_var)      {
    melyseg = new_var;
}

/***
 * Get the value of melyseg
 * @return the value of melyseg
 */
int getMelyseg ()      {
    return melyseg;
}

private:

/***
 * @param
 */
LZWBinFa (const LZWBinFa& )
{
```

```
}
```

```
/***
 * Másolókonstruktur ismételt letiltása
 * @return LZWBinFa&
 * @param
 */
LZWBinFa& operator_ (const LZWBinFa& )
{
```

```
}
```

```
/***
 * @param elem
 * @param os
 */
void kiir (Csomopont* elem, std::ostream& os)
{
```

```
}
```

```
/***
 * @param elem
 */
void szabadit (Csmopont* elem)
{
```

```
}
```

```
void initAttributes () ;
```

```
};
```

```
#endif // LZWBINFA_H
```

```
#include "std.h"

// Constructors/Destructors
//

std:::std ()
{
```

```
}

std:::~std ()
{ }
```

```
//

// Methods
//
```

```
// Accessor methods
//
```

```
// Other methods
//
```

```
#ifndef STD_H
#define STD_H
```

```
#include <string>
```

```
/***
 * class std
 *
 */
```

```
class std
{
public:
```

```
    // Constructors/Destructors
    //
```

```
    /**
     * Empty Constructor
     */
    std ();
```

```
    /**
     * Empty Destructor
     */
    virtual ~std ();
```

```
    // Static Public attributes
    //
```

```
    // Public attributes
    //
```

```
    // Public attribute accessor methods
    //
```

```
    // Public attribute accessor methods
    //
```

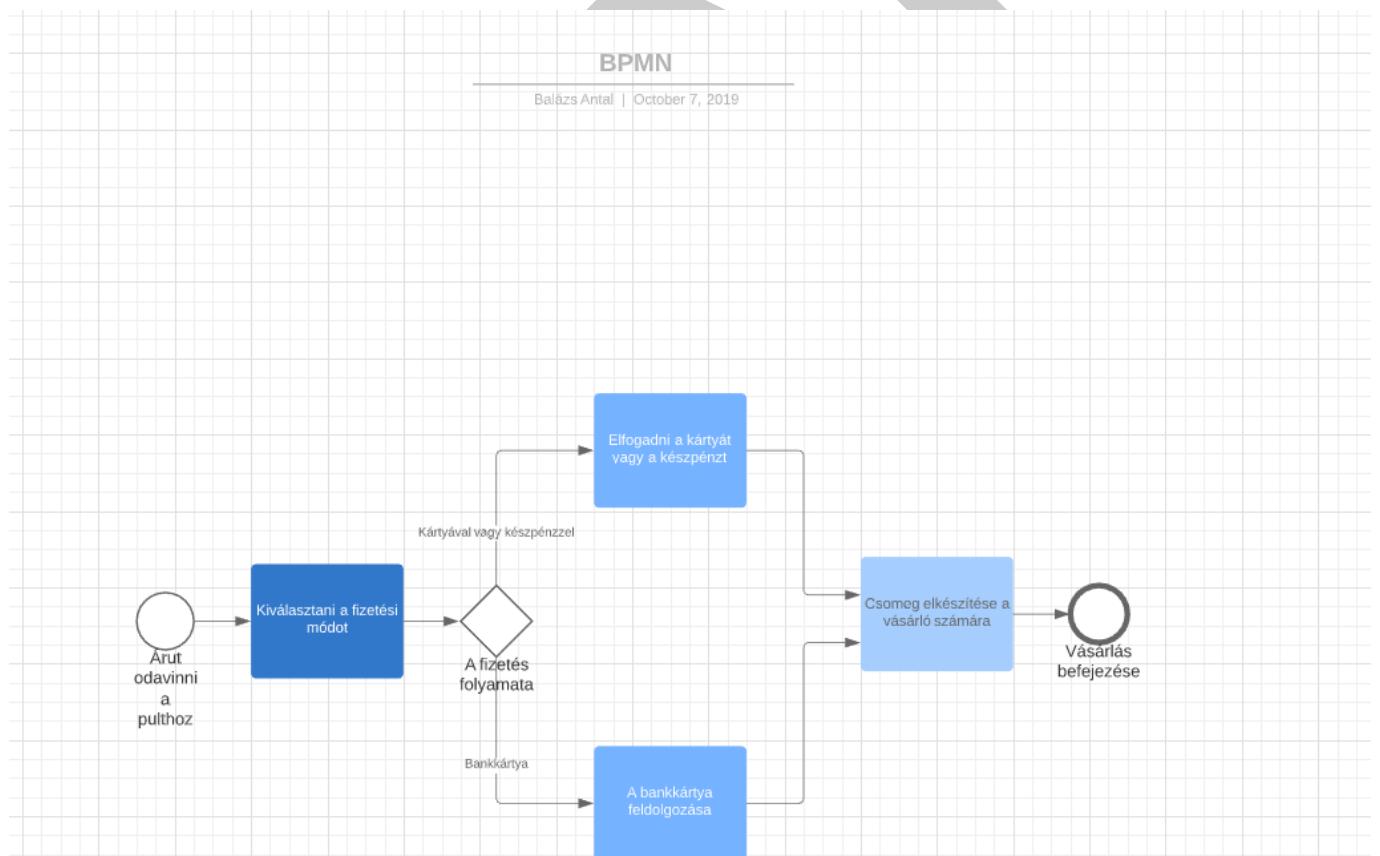
```
protected:  
  
// Static Protected attributes  
//  
  
// Protected attributes  
//  
  
public:  
  
  
// Protected attribute accessor methods  
//  
  
protected:  
  
public:  
  
  
// Protected attribute accessor methods  
//  
  
protected:  
  
  
private:  
  
// Static Private attributes  
//  
  
// Private attributes  
//  
  
public:  
  
  
// Private attribute accessor methods  
//  
  
private:  
  
public:  
  
  
// Private attribute accessor methods  
//  
  
private:
```

```
};  
  
#endif // STD_H
```

A header fájlok tökéletesen demonstrálják az eredeti osztályok vázát. A kódgenerálás esetén a header jön csak létre és a függvények törzse üresen marad. Ezzel a programmal elérhetjük, hogy ne kelljen saját kezűleg készítenünk UML táblát, mivel a kódból képes nekünk legenerálni, valamit visszafele is jól működik.

11.3. BPMN

A következő folyamatábra egy egyszerű vásárlás menetét írja le. Odavisszük a kiválasztott árut az eladóhoz és elővesszük a tárcánkat. Fizethetünk készpénzzel vagy bankkártyával. Odaadom a kézpénzt, az eladó elfogadja és összekészíti a csomagom, vagy odaadom a kártyát a kártya feldolgozódi a pénzt elutalja, az eladó összekészíti a csomagom, amit átveszek és a vásárlásnak vége.



12. fejezet

Helló, Chomsky!

12.1. Encoding

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Chomsky2/Encoding/MandelbrotHalmaz.java>

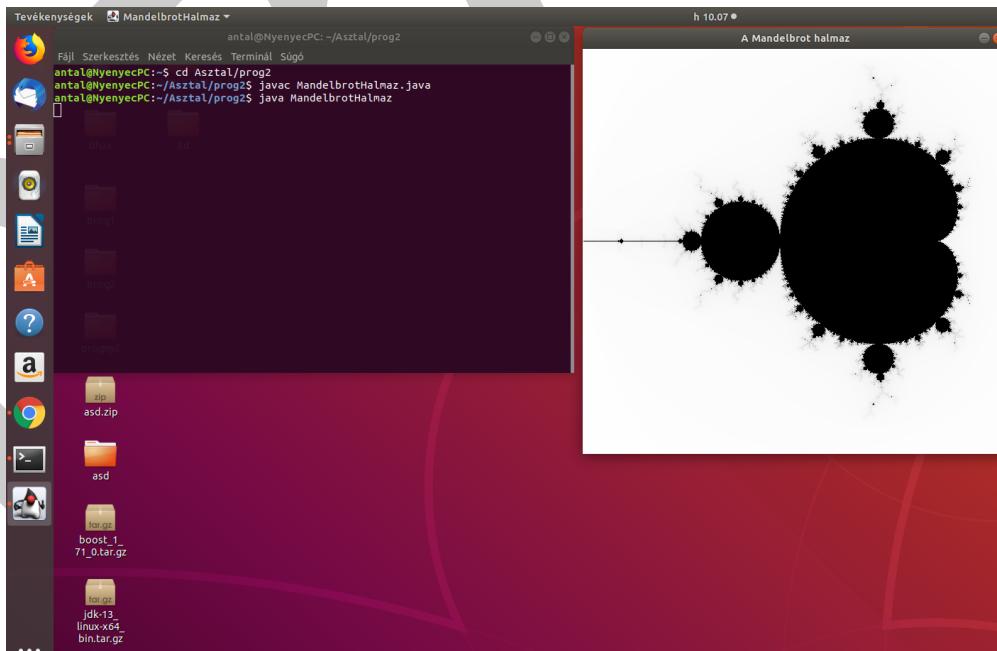
Tanulságok, tapasztalatok, magyarázat... : A feladatunk az volt, hogy bírjuk futtatásra a MandelbrotHalmaz.java forrást úgy, hogy a fájlnévben és a benne szereplő karakterek lehetnek ékezetesek. Fordításnál látszik, hogy a kódolással vannak a gondok, miszerint a program UTF-8 kódolással lett megírva, de vannak olyan karakterek a programban, amelyeket az UTF-8 nem ismer fel és hibát jelez. A következtetés, hogy a program Latin1 vagy Latin2 nyelven íródhatott, mivel azok tartalmaznak ékezetes karaktereket. A program a windows1252-es kódolással íródott, ami megfelelője a Latin1-nek.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
UTF-8
    // vizsgáljuk egy adott pont
    ^
MandelbrotHalmazNagyító.java:40: error:
    UTF-8
        // vizsgáljuk egy adott pont
MandelbrotHalmazNagyító.java:40: error:
    UTF-8
        // vizsgáljuk egy adott pont
MandelbrotHalmazNagyító.java:42: error:
    UTF-8
        // Az egérmutató pozíció
        ^
MandelbrotHalmazNagyító.java:42: error:
    UTF-8
        // Az egérmutató pozíció
        ^
MandelbrotHalmazNagyító.java:42: error:
    UTF-8
        // Az egérmutató pozíció
        ^
100 errors
```

A program első futtatási kísérleténél ezt az üzenetet kapjuk:

Az alábbi parancs beírásával a program fordult és megfelelően működött:

```
$ $ javac -encoding windows1252 MandelbrotHalmaz.java
```



Kép a futtatásról:

12.2. Leet

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...: A feladatunk az volt, hogy írunk saját osztályt, ami leet cipher-ként funkcionál és képes megvalósítani ezt a betűhelyettesítést:https://simple.wikipedia.org/wiki/Leet?fbclid=IwAR0EHWnxFQtNOcl1p5rORCoHyyj1wkjKbAAbs_RXKWUMb2nVHok2FKuA

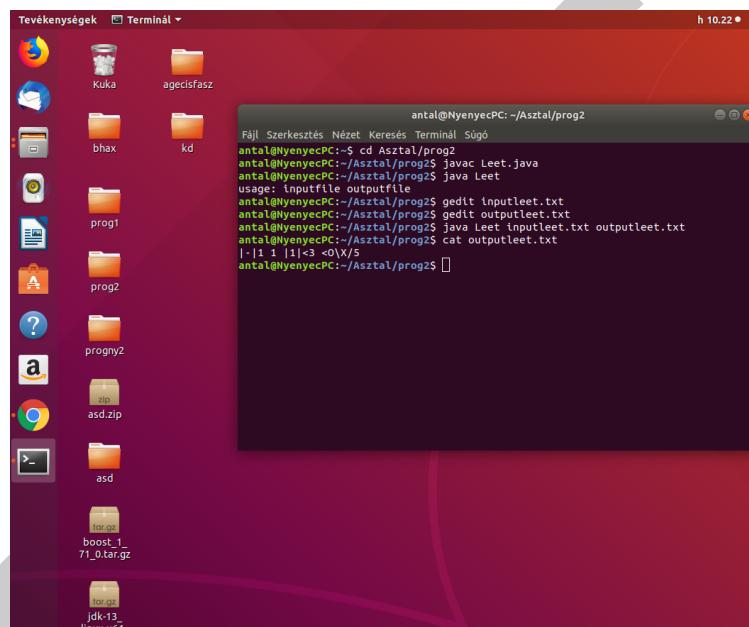
A leet speak a 80-as években formálódott, lényege az, hogy bizonyos karaktereket átalakít az eredetire hasonló karakterekre. Aki nem ismeri a kódolást kis időbe beletelhet neki, még rajón miképpen kell kiolvasni egy adott karakterláncot. A leet-el már a prog1-en is foglalkoztunk, akkor egy Lex készítse el a kódunkat, a megadott szabályainkat alkalmazva, most viszont mi írtunk egyet.

A programot Java-ban készítettem el:

```
public class Leet {  
    public static void main(String[] args) throws Exception{  
        if(args.length != 2){  
            System.out.println("usage: inputfile outputfile");  
            System.exit(-1);  
        }  
        java.io.FileReader file = new java.io.FileReader(args[0]);  
        java.io.FileWriter fw = new java.io.FileWriter(args[1]);  
        LeetCipher lc = new LeetCipher();  
        int k = 0;  
        while ((k=file.read()) != -1) {  
            fw.write(lc.chiper((int)Character.toUpperCase((char)k)));  
        }  
        file.close();  
        fw.close();  
    }  
}  
  
class LeetCipher {  
    private String[] leetchars = new String[]{  
        "4", "8", "<", "[)", "3", "|=", "6", "|-", "1", "_|", "|<", "|", "|V|", "|\\|", "O",  
        "|>", "0.", "|2", "5", "7", "|_|", "\\/", "\\\\X/", "}"}, {"\\/", "2"  
};  
    private String[] leetnums = new String[]{  
        "O", "I", "Z", "E", "A", "S", "G", "T", "B", "g"  
    };  
    public String chiper(int ch) {  
        if (ch >= 65 && ch <= 90){  
            return leetchars[ch - 65];  
        }  
        else if (ch >= 48 && ch <= 57){  
            return leetnums[ch - 48];  
        }  
        else {  
            return String.valueOf((char)ch);  
        }  
    }  
}
```

}

A program elején argumentum vizsgálat történik. Azt nézzük meg, hogy futtatásnál meg-e adtak egy be-meneti és egy kimeneti fájlt, mivel ha nem, akkor értesítjük erről a felhasználót és jelezzük felé a hiányzó részeket. Abban az esetben, ha megfelelő volt az argumentumok száma akkor létrejön egy FileReader és egy FileWriter, ezek a fájlba írást és olvasást fogják szolgálni. While ciklussal addig olvassuk be a fájlt, amíg vége nem lesz, majd kiírjuk a fájlban lévő karakterek leet megfelelőjét egy kimeneti fájlba. Ezt ellenőrizhetjük a fájl megnyitásával, vagy egy egyszerű cat parancsal még a terminálban. A LeetCipher osztályban van egy String típusú tömb, ami tárolja a számunkra fontos leet karaktereket. Található itt még egy String tömb, ami pedig a számokat és azok leet megfelelőit tárolja. A karakterek átalakítását a cipher metódus fogja elvégezni, az alapján, hogy számot vagy betűt olvas, annak fogja visszaadni a megfelelő karakterét.



A program futtatás után:

12.3. Fullscreen

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/Chomsky2/Fullscreen>

Tanulságok, tapasztalatok, magyarázat... : A szereplő osztályt megírjuk, ebben lesz megtalálható a karakterünk és annak tulajdonságai.(oszlop,sor pozíció, a labirintus amiben szerepel, annak szélessége és magassága, véletlenszámgenerátor a szereplők mozgatására) Egy void függvényel meghatározzuk szereplőnk kezdetbeli helyzetét, többször megpróbáljuk őt elhelyezni, ezeket a próbálkozásokat számon tartjuk. Max 10x próbálkozhatunk, amíg a szereplőt nem sikerül a "falba tennünk" akkor kiléünk. Ezután a szereplő léptetésének implementálása következik, valamit a szereplő másik szereplőtől való távolsága. Beállítjuk a this.sor valamit oszlop-pal szereplő labirintus beli oszlop és sor kordinátáit, majd ezeket a return visszaadja. A szereplő osztály tagként tartalmazza annak a labirintusnak a referenciáját, amelyikbe helyezzük. A szereplőnek van labirintusa nem pedig fordítva. Ha a szereplő osztály nem tartalmazná ezt a tagot, minden át kellene neki adni a labirintus referenciáját, hogy a mozgató függvények le tudják kérdezni, hogy hová léptetik a szereplőket.

A hős osztályunkban értelemszerűen a hős tulajdonságait adjuk meg. (életeinek száma, maximális életeinek száma, valamint a kincsek értékeit) A labirintus objektumban mozog a hősünk. Gyűjtögeti a megtalált kincseket: kincs objektum amit éppen megtalált a hős. A megtalált értékhez adjuk a talált kincs értékét.

DRAFT

13. fejezet

Helló, Stroustrup!

13.1. JDK osztályok

Megoldás forrása: https://gitlab.com/baluka94/bhax/blob/master/Stroustrup/JDK/jdk_classes.cpp

Tanulságok, tapasztalatok, magyarázat... : Az src mappában található .java kiterjesztésű fájlok kiíratása volt a dolgunk. Ehhez alkalmaztam a boost könyvtárat, amely hatalmas segítséget nyújt azzal, hogy rekurzívan bejár minden mappát a megadott elérési útvonalon belül és megszámolja a ".java" kiterjesztésű fájlokat. Fordításnál szükséges különböző kapcsolókat megadni hozzá, majd a futtatásnál meg kell adni a tartalmazó mappa nevét ebben az esetben az src.

A programban használnunk kell a következő fájlt, mint headert, aminek elérési útvonalát meg kell adnunk:

```
#include "boost_1_71_0/boost/filesystem.hpp"
```

A fordítás és futtatás így néz ki:

```
g++ jdk_classes.cpp -o jdk_classes -lboost_system -lboost_filesystem -lboost_program_options -std=c++14  
./jdk_classes src
```

```
antal@NyenyecPC: ~/Asztal/mappa
Fájl Szerkesztés Nézet Keresés Terminál Súgó
inflating: src/jdk.xml.dom/org/w3c/dom/css/DocumentCSS.java
inflating: src/jdk.xml.dom/org/w3c/dom/css/ViewCSS.java
inflating: src/jdk.xml.dom/org/w3c/dom/css/CSSPrimitiveValue.java
inflating: src/jdk.xml.dom/org/w3c/dom/css/package-info.java
inflating: src/jdk.xml.dom/org/w3c/dom/css/CSS2Properties.java
inflating: src/jdk.xml.dom/org/w3c/dom/css/CSSCharsetRule.java
inflating: src/jdk.xml.dom/module-info.java
inflating: src/jdk.zipfs/module-info.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ByteArrayChannel.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileSystem.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileStore.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipPath.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipDirectoryStream.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/JarFileSystem.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/JarFileSystemProvider.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipInfo.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipUtils.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileSystemProvider.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileAttributes.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipCoder.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipConstants.java
inflating: src/jdk.zipfs/jdk/nio/zipfs/ZipFileAttributeView.java
Az src.zip-ben található java osztályok száma: 17844
antal@NyenyecPC:~/Asztal/mappa$
```

A programot a fénykard-ra nevű programra alapozva valósítottuk meg. Futtatásnál létrehoz egy src nevű mappát, amibe kicsomagolja az src.zip állomány tartalmát.

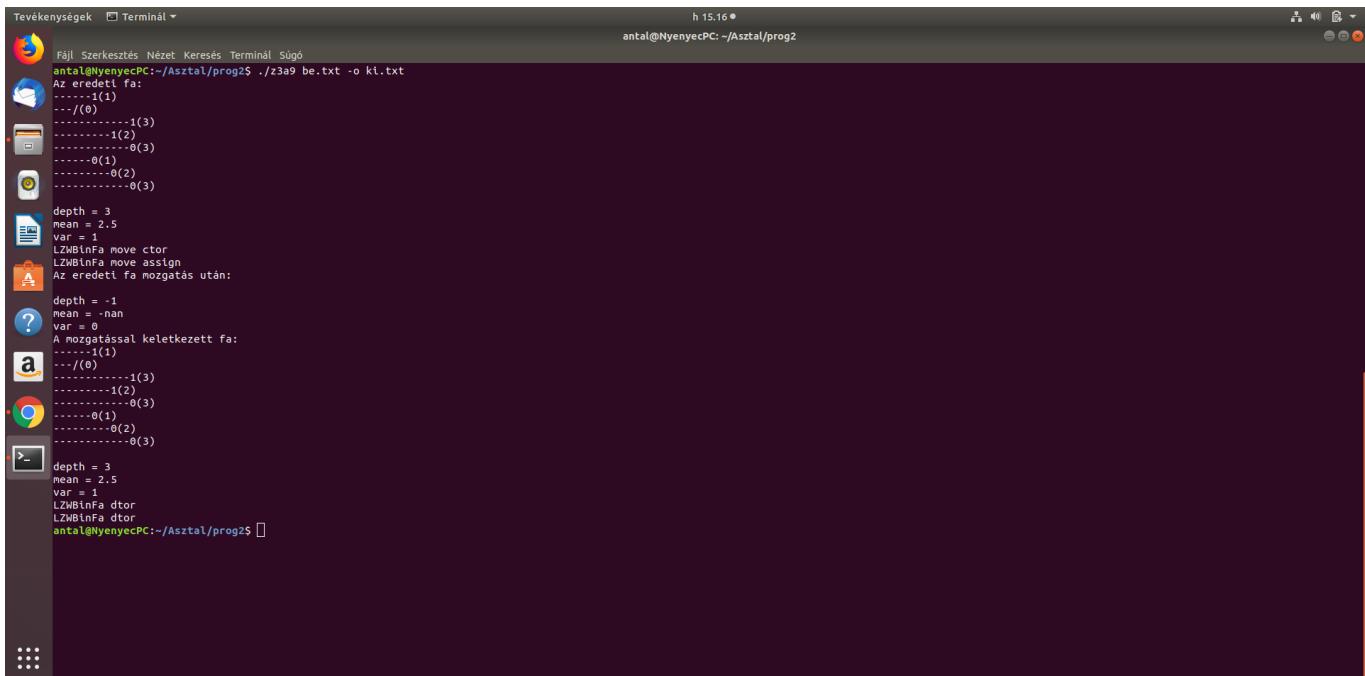
13.2. Másoló-mozgató szemantika és Összefoglaló

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/Stroustrup/mozgato.cpp>

Tanulságok, tapasztalatok, magyarázat...: A feladat a másoló és mozgató konstruktörök összehasonlítása volt C++11-ben.

A feladat megoldásához tökéletesen megfelel az LZWBInFa program.

A futtatás után:



```
Fájl Szerkesztés Nézet Keresés Terminál Súgó h 15:16 ●
antal@NyenyecPC:~/Asztal/prog2$ ./z3a9 be.txt -o ki.txt
Az eredeti fa:
....1(1)
.../(0)
.....-1(3)
.....-1(2)
.....-0(3)
....0(1)
....-0(2)
.....-0(3)

depth = 3
mean = 2.5
var = 1
LZWBinFa move ctor
LZWBinFa move assign
Az eredeti fa mozgatás után:

depth = -1
mean = -nan
var = 0
A mozgatással keletkezett fa:
....1(1)
.../(0)
.....-1(3)
.....-1(2)
.....-0(3)
....0(1)
....-0(2)
.....-0(3)

depth = 3
mean = 2.5
var = 1
LZWBinFa dtor
LZWBinFa dtor
antal@NyenyecPC:~/Asztal/prog2$
```

Másoló konstruktur:

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

Mozgató konstruktur:

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

Mindkét konstruktornak szüksége van létező objektumokra, de létrehozásuk másképp lett implementálva. A másoló konstruktur a régi objektumot teljes egészében lemásolja, ez nem igazán szerencsés, mivel sok memóriát használ, különösen szerencsétlen abban a helyzetben, amikor az objektumok nagyok. Az eredeti objektum másolás hsználható marad. Ezt a másolási módszert sekély másolásként ismerjük. A mozgató konstrukturunk mély másolást hajt végre. Ez a fajta konstruktur nem foglal le a memóriában új területet az új objektumnak, hanem a régi objektum adatait adja át az újnak. Ennek a következménye az, hogy a mozgatás után nem leszünk képesek továbbá hozzáérni a régi objektumunkhoz, mivel minden eleme át lett adva az újonnan létrehozottnak. A különbség még a kettő között az is, hogy a másoló konstruktur bal érték referenciát vár, míg a mozgató konstrukturunk jobb érték referenciát. A kettő kötözzi eltérést a legegy-szerűbben úgy tudjuk megmagyarázni, hogy bal értéknek számítanak a változók és jobb értéknek minősül minden konstans és a változóknak a szorzata. Másoló értékkadást, akkor használunk, amikor egy meglévő objektumunkból szeretnénk legenerálni méggy példányt. A másoló szemantika megvalósításához először meg kell írnunk egy másoló konstruktort. Két féle másolás létezik az első az úgy nevezett shallow copy vagyis sekély másolás, ami azt jelenti, hogy másolás ténylegesen nem megy végbe csak a lemasolandó objektum címét fogja megkapni a másolt objektumunk. Tehát ez az objektum nekünk csak egy referenciaként fog szolgálni, ami abban az esetben jelenthet problémát, hogy ha a másolt objektumunkon módosításokat

eszközölünk, mivel ez az eredetinek a memóriacímére mutat, így az is változni fog. A következő másolás pedig a deep azaz mély másolás, ilyenkor beszélünk tényleges másolásról. Ennél a másolásnál egy új memóriacím kerül lefoglalásra, ezért a futási ideje is több időt fog igénybe venni, a sekély másolással szemben. Ez abban az esetben lesz hasznos, ha változtatásokat szeretnénk eszközölni a lemásolt objektumon, mivel ez új memóriacímet kapott, így módosításnál az eredeti objektumunk változatlan marad. Összetett objektumok esetén rekurzív másolást kell végrehajtanunk, ami azt jelenti, hogy az összes adattag tulajdonságait le kell másolnunk. A mozgató szemantika hasonló a másoló szemantikához. Az új és a régi objektumunk meg fog egyezni. A mozgató értékadásban, ha "=" jel operátort használunk, akkor az std::swap() függvénytel megcserélődik a 2 gyökér mutatója. A mozgató konsturktor: Itt először nullptr értéket adunk az abban a fában lévő gyökérnek, amelybe mozgatni szeretnénk a másik fát. A másik fát azt std::move() függvényünkkel átmozgatjuk, ami azt jelenti, hogy a gyökérmutató mostmár a paraméterül kapott másik fára mutat. Ez azért van így, mivel az std::move() függvény valójában nem végez mozgatást, hanem a paraméterül kapott értéket fogja átalakítani jobbérték referenciává.

DRAFT

14. fejezet

Helló, Gödel!

14.1. Gengszterek

Megoldás forrása: <https://github.com/nbatfai/robocar-emulator>

Tanulságok, tapasztalatok, magyarázat... : A projekt felélesztése nem sikerült.

Lambda kifejezésekéről a Java 8-tól beszélünk. A lambda kifejezés segítségével ideiglenes függvényeket tudunk írni.

Általános szintaxis:

```
[captures] (params) {body}
```

A [capture] részben változókat, objektumokat adhatunk meg, amiket majd a kifejezésben használunk. Megtehetjük ezt a változók felsorolásával, vagy pedig át lehet adni "mindent". "=" használatakor másolással vagy pedig "&" jel használatakor referenciaiként történik.

A (params) részben fogjuk megadni azokat a paramétereket, amiket a függvény használ.

A {body} rész maga a függvényben végrehajtódó kód.

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

Az std::sort-tal, ami a lamdba kifejezést használja azt oldjuk meg, hogy minden rendőr megtalálja a hozzá legközelebb eső gengsztert (fordítva pontosabban, mivel a lényeg ez lenne).

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

A gangsters vector-t fogjuk rendezni a lambda kifejezést használva. Paraméterként két Gangster objektumot vár, akkor ad vissza igaz értéket ha x gangster közelebb van az elkapott cop-hoz, mint az y gangster. A rendezés után a vector elején a rendőrhöz legközelebb elhelyezkedő gangsterek lesznek.

14.2. Alternatív Tabella rendezése

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/G%C3%B6del/tabellazAlternativTabella.java>

Tanulságok, tapasztalatok, magyarázat...: A Comparable<T> szerepének bemutatása volt a feladat. Ennek segítségével egy összehasonlító osztály hozható létre. A compareTo(T obj) függvény összehasonlítja a meglévő objektumot a kapott objektummal. Ha a kapott objektum értéke kisebb akkor -1-el, ha nagyobb, akkor 1-el, egyéb esetén pedig 0-val tér vissza. A képen látható Csapat osztály definíciója implementálja a Comparable interface-t. Az alternatív tabellát a labdarúgó bajnokságokra alkották meg, ez egy olyan táblázat ami figyelembe veszi az ellenfél csapat erejét, vagyis ha valaki erősebb csapatot győz le, akkor nyilván több pontot fog kapni, mint az aki gyengébbet ver el.

The screenshot shows a Linux desktop environment with a dark orange background. In the top right corner, there is a terminal window titled "AlternativTabella.java" which contains the following Java code:

```
System.out.println("||-");
System.out.println("|| " + e[0]);
System.out.println("|| " + ep[0]);
System.out.println("|| " + csapat.nev);
System.out.println("|| " + e4);

    ++i;
}
System.out.println("||-");
}

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;
        }
    }
}
```

The terminal window also displays some status information at the bottom: "Java", "Tabulárorszélesség: 8", "47. sor, 9. oszlop", and "BESZ".

To the left of the terminal, there is a file manager window showing various files and folders on the desktop, including "Kuka", "q", "bhax", "kd", "prog1", "map.ppa", "prog2", "src.zip", "JDKe.txt", "asd", "qt2", "boost_1_71_Utar.gz", "tar.gz", and "jdk-13_linux-x64_bin.tar.gz".

Hazai \ Vendég ¹	HON	DEB	DVTK	FTC	HAL	KISV	MEZ	MTK	PAK	PUS	ÚJP	VIDI
Budapest Honvéd		3–0	1–0	0–1	3–2	4–0	1–1	2–1	1–0	2–1	2–2	0–3
Debreceni VSC	2–0		2–0	2–1	1–1	3–1	1–1	3–3	2–1	2–1	0–0	0–1
Diósgyöri VTK	2–1	1–0		1–4	1–0	1–1	1–1	3–2	0–0	2–2	1–2	0–1
Ferencváros	1–0	2–2	4–1		3–1	2–0	3–2	2–0	1–1	4–0	1–0	2–2
Haladás	0–1	0–2	1–1	1–0		0–1	1–2	1–2	1–2	2–1	2–2	0–2
Kisvárda	0–3	3–0	1–1	0–2	4–1		1–2	1–0	0–0	1–0	1–4	2–2
Mezőkövesd-Zsóry FC	0–1	2–2	4–2	0–1	2–0	2–2		2–3	3–1	2–0	0–0	1–0
MTK	1–1	0–1	1–0	1–4	4–0	0–1	2–2		1–2	3–2	1–0	1–3
Paksi FC	0–0	2–1	1–2	0–3	1–1	4–1	2–1	3–0		3–2	1–1	0–4
Puskás Akadémia	1–0	0–1	2–1	1–0	2–0	1–1	2–0	1–2	1–1		0–1	2–1
Újpest	0–0	1–0	5–0	1–1	2–0	1–0	1–1	0–2	1–1	2–0		2–0
Vidi	2–0	1–1	1–2	2–1	1–0	4–0	3–1	0–3	1–1	0–3	1–0	

A táblázatban láthatóak a pontozások:

üres: 0 zöld: 1 sárga: 2 piros: 3

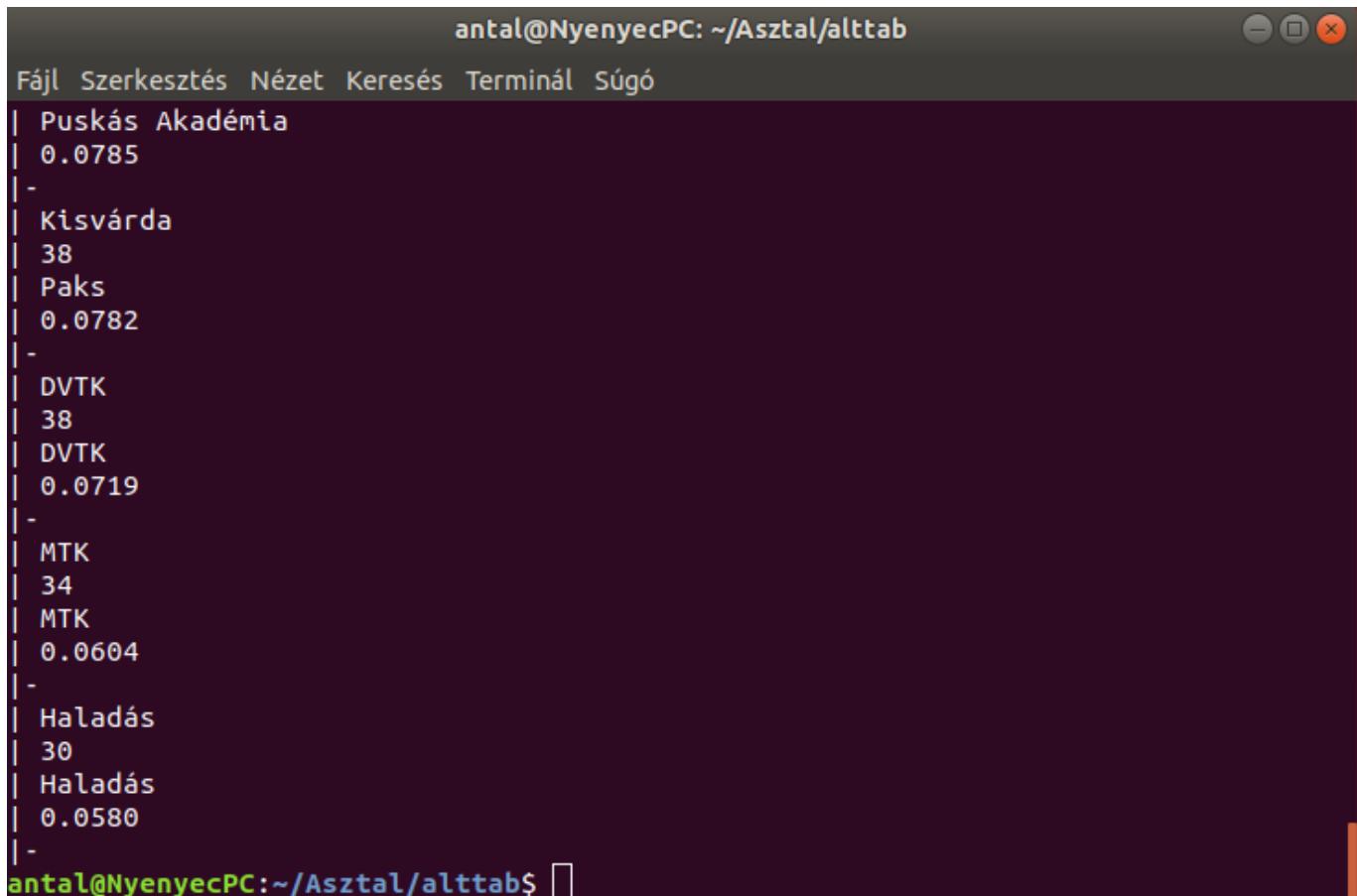
A kereszttábla pontjait a Wiki2Matrix programban adjuk meg egy mátrixban és futtatáskor egy linkmátrixot kapunk, amit megadunk az AlternativTabella programnak. A program pontszámítási része nem volt a feladatunk része.

Az interface-re a következő sorok miatt van szükség:

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

Az első sorban semmi probléma, létrehozunk egy listát, ez az adatszerkezet a Collection interface-t egészíti ki. Ebben a listában a csapatokat tömbbé fogjuk konvertálni az Arrays.asList() segítségével.

A második sor miatt pedig a következő miatt. A sort() metódussal listákat rendezhetünk a természetes sorrendjük szerint. Ez a rendezés a Comparable<T> interface által létrehozott rendezés, tehát kiegészít a sort() az interface-t ezért is implementáljuk.



```
antal@NyenyecPC: ~/Asztal/alttab
Fájl Szerkesztés Nézet Keresés Terminál Súgó
| Puskás Akadémia
| 0.0785
|
| Kisvárda
| 38
| Pak
| 0.0782
|
| DVTK
| 38
| DVTK
| 0.0719
|
| MTK
| 34
| MTK
| 0.0604
|
| Haladás
| 30
| Haladás
| 0.0580
|
antal@NyenyecPC:~/Asztal/alttab$
```

14.3. GIMP Scheme hack

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

Tanulságok, tapasztalatok, magyarázat....:

Króm effekt:

A program elején függvényeket deklarálunk. A color-curve függvényben tárolunk egy tömbben 8 értéket, amik a GIMP színgörbékének értékei. Az elem függvénnnyel lekérjük egy listáról, hogy hanyadik a megadott elem. A lisp-ben a változók listák lehetnek, tehát több elemet is tartalmazhatnak. A wh-ban az adott szöveg magasságát és szélességét kérjük le. A chrome függvényünk paraméterei a következők: szöveg, betűtípus, betűméret, szélesség, magasság, szín, színátmennet

Majd a szöveg haladását figyeljük lépésekben:

Egy fekete háttéren elhelyezünk egy középre tolta fehér szöveget.

Egy erős gauss elmosódást helyezünk a képre.

A színek határszintjeinek alsó és felső határának módosításával élesítünk a képen.

Újból alkalmazzuk a gauss elmosódást.

Szín szerinti kijelöléssel kijelöljük a fekete részt.

Létrehozunk és kijelölünk egy átlátszó réteget.

Az átlátszó réteget kitöljük sötét -és világosszürkéből álló színátmenettel.

Bucka leképezést alkalmazunk az első réteg felhasználásával a második rétegen.

A színgörbéket módosítjuk, hullám mintát állítunk be.

Elmentjük a script-et a GIMP menüjébe és alapértelmezett paraméterekkel látjuk el.

Mandala:

Itt is függvény deklarálások történnek a program elején. Lekérjük a szöveg szélességét a text-width függvénnyel. Az alapból beépített gimp függvén több értéket is visszaad, ezért szükséges a car függvény, amivel elérjük, hogy csak az első vagyis a szélesség függvényt adja vissza. A text-wh-val egy listába lekérjük az adott szöveg szélességét és magasságát. A mandala script definiálása 8 függvénnnyel történik: törzs szöveg, szöveg középre, betűtípus, betűméret, szélesség, magasság, szöveg színének RGB kódja, színátmenet. A let*-al definiálunk változókat a lisp-ben. RGB képet hozunk létre a megadott magassággal és szélességgel. Elkészítjük a háttérét illetve a szükséges szövegeket. Beállítunk egy előtérszínt, amivel a háttérét kitöljük, értékül adjuk neki a paraméterként kapott színt. Betűméretet és betűstílust állítunk.

Elforgatjuk a szöveget, úgy hogy pi hatványaival szorozzatjuk a fokszámukat.

Elipszis kivágással kereteket hozunk létre.

Beállítjuk a színátmenetet, létrehozzuk a középen elhelyezett szöveget.

Regisztráljuk a GIMP fájlmenüjében a script-et, alapértelmezett paramétereket állítunk be.

15. fejezet

Helló, !

15.1. FUTURE tevékenység editor

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/hello/future/ActivityEditor.java>

Tanulságok, tapasztalatok, magyarázat... : A feladat az volt, hogy javítsunk valamit a future programon. A program futtatáshoz Java8-at telepítettem pontosabban a zulufx 8.0.202-es verziót, mivel magasabb Java-n nem futott a program. sdkman-el telepítettem. A hiba amit javítottam, az volt, hogy ha létrehozunk egy új altevékenységet egy tevékenységen, akkor azután már nem lehet másikat létrehozni, mivel ilyenkor egy könyvtárszerkezetet hozunk létre és 2 ugyan olyan nevű mappa keletkezne.

Megoldás:

```
int i=1;
while(true) {
    java.io.File file = getTreeItem().getValue();

    java.io.File f = new java.io.File(file.getPath() + System. ←
        getProperty("file.separator") + "Új altevékenység"+i);

    if (f.mkdir()) {
        javafx.scene.control.TreeItem<java.io.File> newAct
        // = new javafx.scene.control.TreeItem<java.io. ←
        File>(f, new javafx.scene.image.ImageView(actIcon));
        = new FileTreeItem(f, new javafx.scene.image. ←
            ImageView(actIcon));
        getTreeItem().getChildren().add(newAct);
        break;
    } else {
        ++i;

        System.out.println("Can't create file, trying to create ←
            +" Új altevékenység"+i+" instead+" path: "+f. ←
            getPath());
    }
}
```

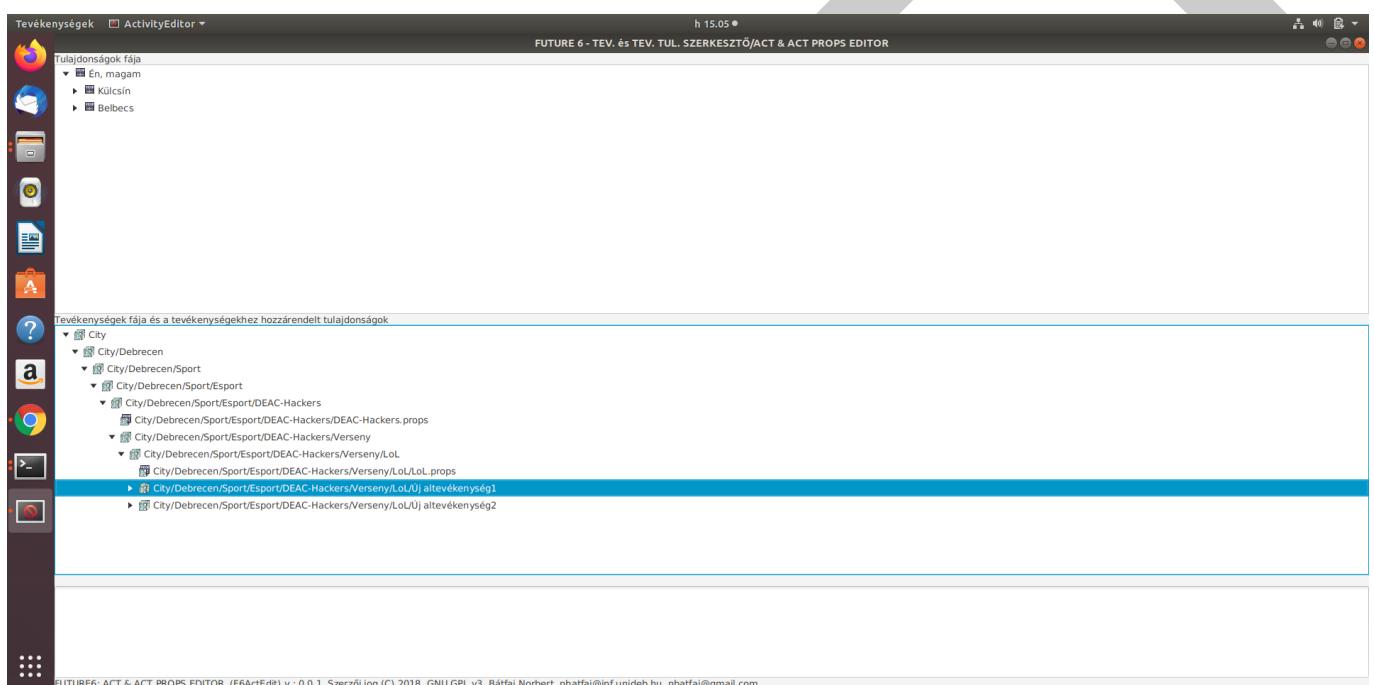
```
}
```

```
}
```

```
} );
```

Deklaráljuk az i ciklusváltozót és végtelen while-ba helyezzük a program Új altevékenységet létrehozó részét. Létrehozunk a könyvtárban egy Új altevékenységet a nevében a while aktuális indexét használva. Egy if-ben megnézzük létezik-e már a tevékenység, amit létre akarunk hozni, ha nem break-el kilépünk a ciklusból és létrehozhatjuk gond nélkül az Új altevékenységet. Ha mégis létezne már a mappa, akkor növelni kell az altevékenység indexét vagyis a ciklusváltozót, majd kiírjuk, hogy nem sikerült létrehozni a fájlt és újat próbálunk létrehozni. A while ismét lefut ezúttal megnövelt i-vel. Hátránya a while annyiszor fut le, ahány tevékenység már eleve létezik és nem tud létrehozni egy újat.

A program futtatás után:



15.2. Samu Cam

Megoldás forrása: <https://github.com/nbatfai/SamuCam>

Tanulságok, tapasztalatok, magyarázat...: A feladat az volt, hogy a webkamera kezelésére mutassunk rá a programban. Az openCV könyvtárat használjuk.

```
std::string videoStream = parser.value( webcamipOption ).toString();  
SamuLife samulife( videoStream, 176, 144 );
```

A parancssori argumentumként használt ip címet átadjuk paraméterként.

Példányosítás:

```
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 );
}
```

Megnyitjuk a videoStream-et. A videoCapture objektum open() függvénye megnyitja a paraméterként kaptott fájlt, ezután beállítjuk a méretét és a frame/sec-et 10-re.

Futás előtt CascadeClassifier betöltése, ami a képek elemzését végzi:

```
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;
}
```

A load segítségével egy classifier megnyitása, ami egy arcot ír le.

```
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, 4, ←
    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 );
}
```

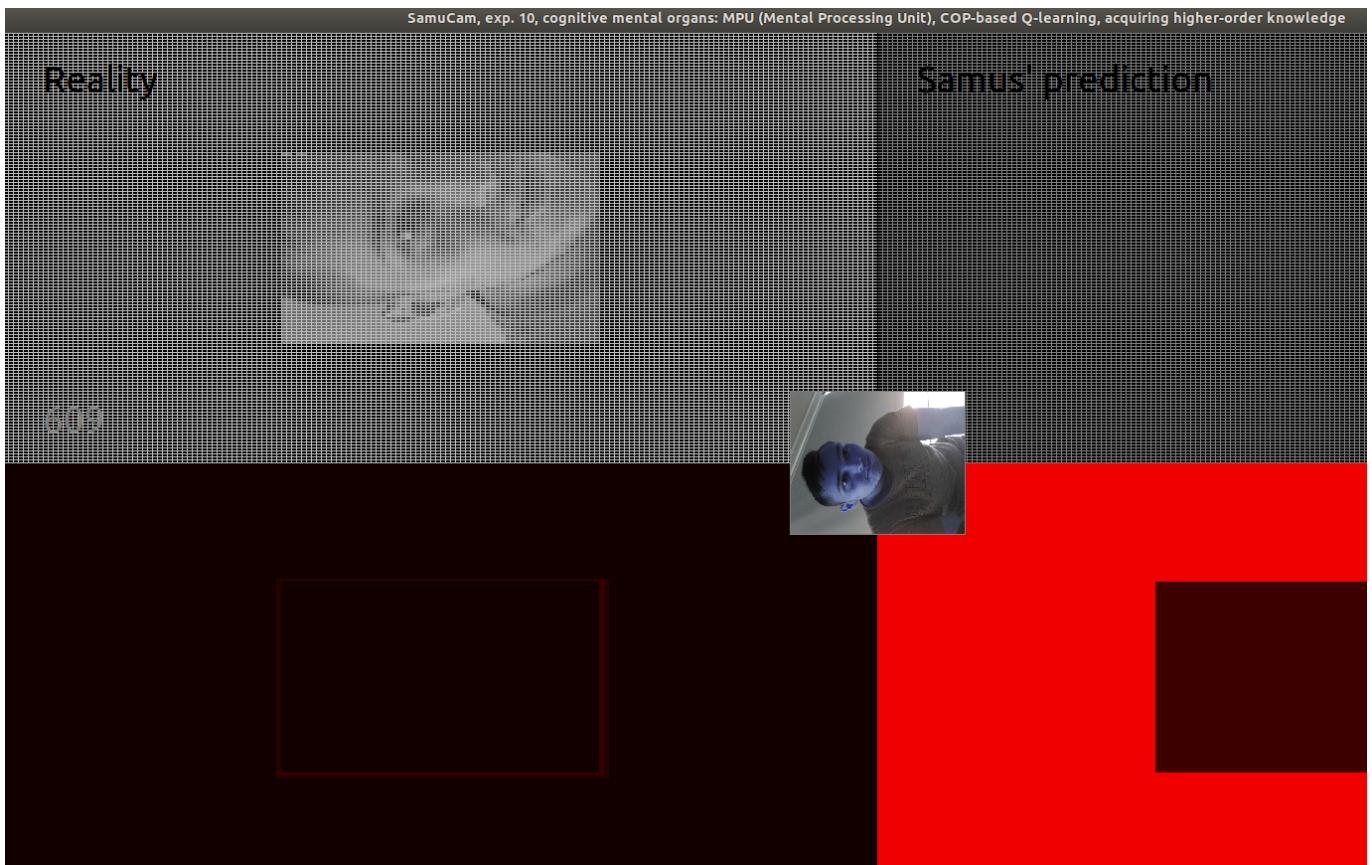
80ms-ként a read függvény segítségével beolvasunk egy pixelt, ami a cv::mMat tömbbe menti a képet. A képet átméretezzük és interpoláljuk. Kép color space -> grayscale, kiegyenlítjük a hisztogramját. A detectMultiScale-el arcokat keresünk az input képben, a talált objektumok egy rectangle listaként térnek vissza. Az első talált arcból QImage-t készítünk, amit a SamuBrain osztály dolgoz fel a faceChanged signal után. Az arc köré keretet rajzolunk, amiből újabb QImage készül, ezt továbbadjuk a SamuLife számára, ami frissíteni fogja a képet.

Futtatáskor átírtam ezt a sort:

```
videoCapture.open ( 0 );
```

Így nem egy file-t vár paraméterként hanem egy device IP-t.

A program futtatás után:



15.3. BrainB

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...: A Qt slot-signal mechanizmus bemutatása volt a feladatunk. Qt-ban a slot-signal mechanizmust objektumok közötti kommunikációra használhatjuk. Ha egy bizonyos esemény bekövetkezik signal-t bocsátunk ki. A slot egy függvény amit akkor hívunk meg, ha egy ilyen signal bekövetkezik. Ezzel eltudjuk kerülni a cross reference-t, mivel nem kell egy osztályon belül egy másik osztály függvényeit használnunk. A slot függvényeket a signal-ohoz connect függvényel kapcsoljuk. A szignatúrája a slot-nak és a signal-nak meg kell egyezzen.

Nekünk 2 connenct függvényünk van a forrásban:

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

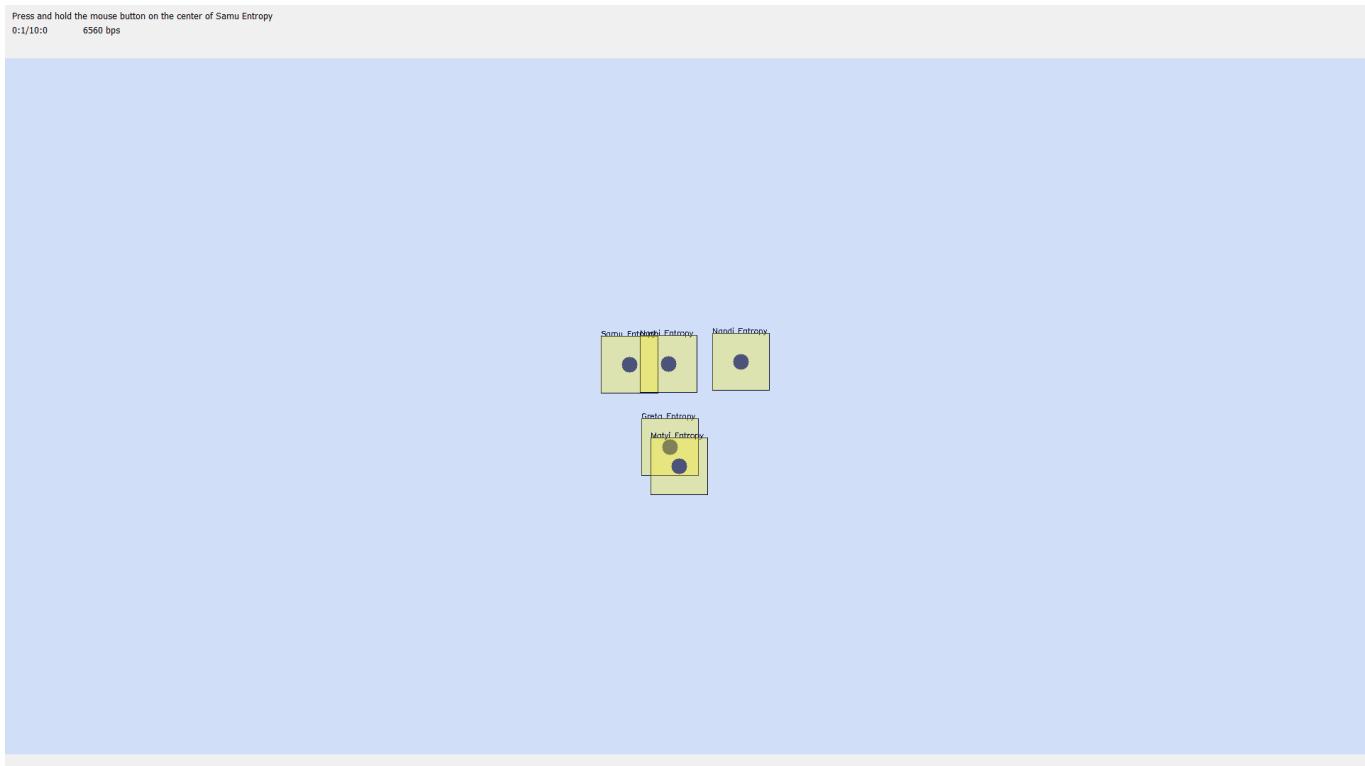
Ha brainBThread-ben a heroesChanged signal bekövetkezik, lefut a BrainBWin updateHeroes függvénye. endAndStats-nál szintén, ez akkor fut le ha a futási időnek vége, kiíródik egy debug üzenet, ami jelzi, hogy a játéknak vége és ment egy eredményt.

```
void BrainBThread::run ()
```

```
{  
    while ( time < endTime ) {  
  
        QThread::msleep ( delay );  
  
        if ( !paused ) {  
  
            ++time;  
  
            devel();  
  
        }  
  
        draw();  
  
    }  
  
    emit endAndStats ( endTime );  
  
}  
  
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();  
}
```

Futtatáskor:





DRAFT

16. fejezet

Helló, Lauda!

16.1. Port scan

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/lauda/port/KapuSzka.java>

Tanulságok, tapasztalatok, magyarázat... : A program azt nézi meg, hogy a gépünk milyen portokat figyel. A program végigmegy a parancssorban megadott nevű gép 1024 alatti számú TCP kapuin: megpróbál egy TCP kapcsolatot létrehozni, ha sikerül, akkor a célpontot ül egy szerver folyamat, ha pedig nem azaz kivétel keletkezik, akkor nem. Siker esetén nem csinálunk semmit, hanem csak bezárjuk az éppen elkészített kliensoldali kommunikációs végpontot reprezentáló socket objektumot, ha pedig nem sikerül, akkor IOException kivétel keletkezik és a catch blokk hajtódik végre, ami ki fogja írni, hogy az adott portot nem figyeli semmi.

```
antal@NyenyecPC: ~/Asztal/prog2/kapu
Fájl Szerkesztés Nézet Keresés Terminál Súgó
antal@NyenyecPC:~/Asztal/prog2/kapu$ javac KapuSzkenner.java
antal@NyenyecPC:~/Asztal/prog2/kapu$ java KapuSzkenner 172.20.10.8
0 nem figyeli
1 nem figyeli
2 nem figyeli
3 nem figyeli
4 nem figyeli
5 nem figyeli
6 nem figyeli
7 nem figyeli
8 nem figyeli
9 nem figyeli
10 nem figyeli
11 nem figyeli
12 nem figyeli
13 nem figyeli
14 nem figyeli
15 nem figyeli
16 nem figyeli
17 nem figyeli
18 nem figyeli
19 nem figyeli
20 nem figyeli
21 nem figyeli
```

Módosíthatunk egy kicsit a programon annak érdekében, hogy megtudjuk milyen exception-t dob a try.

```
antal@NyenyecPC: ~/Asztal/lauda
Fájl Szerkesztés Nézet Keresés Terminál Súgó
sed)
1015 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1016 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1017 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1018 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1019 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1020 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1021 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1022 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1023 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1024 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
1025 nem figyelijava.net.ConnectException: Kapcsolat elutasítva (Connection refu
sed)
antal@NyenyecPC:~/Asztal/lauda$ 
```

Ha nem sikerült kapcsolatot létesíteni ConnectionException-t ad a program. Ez azt jelzi, hogy hiba történt

egy socket távoli címhez és porthoz történő csatlakoztatásakor. Általában a kapcsolatot távolról tagadják meg pl. egy folyamat sem figyeli az adott cím adott portjait. Ha sikerül kapcsolatot létrehoznia a programnak, akkor tudja, hogy a portot figyelik, ha pedig nem, akkor azt, hogy nem figyelik.

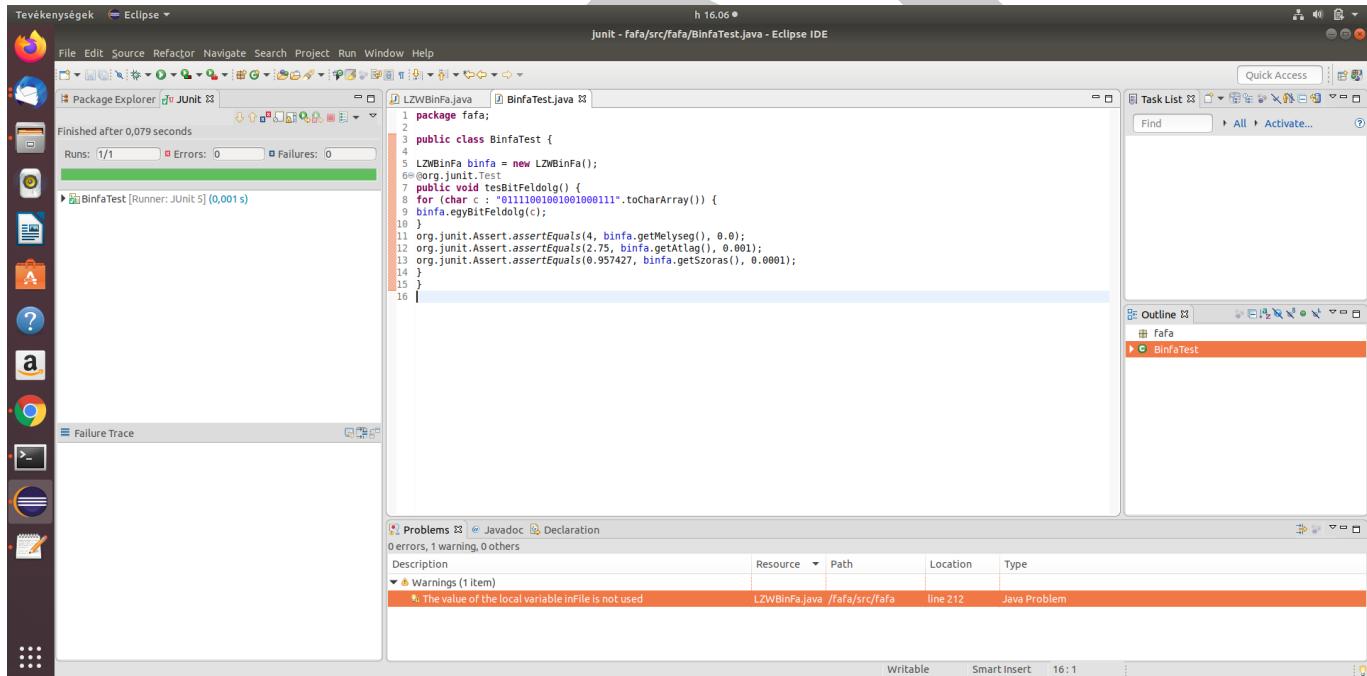
16.2. Junit Teszt

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...: A feladat az volt, hogy egy junit tesztbe dolgozzuk be a binfa kézzel számított mélységét és szórását. A posztban felhasznált jelsorozatot használjuk hozzá. A binfa objektum a BinFeldolg dolgozzuk fel karakterenként a kézzel megadott string karakter változatát. Ha fel-dolgozásra kerültek megnézzük, hogy a számolt érték és a tanárúr által számolt értékek megegyeznek-e, ha igen a program jól működik.

```
public static void assertEquals(double expected, double actual, double delta)
```

Ez a metódus megnézi, hogy a várt és a tényleges érték megegyezik-e, és lehet egy kis eltérés. Ha nem egyezik AssertionError-t kapunk és a teszt elbukott.



16.3. AOP

Megoldás forrása: <https://gitlab.com/baluka94/bhax/tree/master/lauda/aop>

Tanulságok, tapasztalatok, magyarázat...: A feladat az volt, hogy írunk bele egy átszövő vonatkoztatást az LZWBInFa Java átiratába. Ehhez használnuk kellet valamilyen aspektus orientált paradigmát. Az aspektus orientált programozás, egy olyan paradigma, amivel lehetőségünk lesz arra, hogy egy a programok áttekintetősége és újból való felhasználhatósága érdekében aspektusokat fűzhetünk a programhoz. Ez annyit tesz,

hogy a forráskód módosítása nélkül is képesek leszünk változásokat eszközölni az adott programon, hogy aztán a várt eredményt kapjuk a futtatáskor. Ezeket ún. aspektusnyelven írjuk, mely nagyban hasonlít az objektumorientált programozási nyelvekre. Az aspektust "hozzáfűzzük" a programhoz, így a forráskód átírása nélkül tudtuk megmásítani a program viselkedését. Az aspektus csupán egy módosított programrészről tartalmaz, ezért önmagában nem felhasználható. AspectJ-t használtam a megoldáshoz. A kódcsipeteket megadtuk egy külön fájlban, amiknek megmondhatjuk, hogy egy adott metódus előtt vagy után fussanak le. Az inorder és postorder módon való kiíratást valósítottam meg.

```
import java.io.FileNotFoundException;
import java.io.PrintWriter;
public aspect BinFa{
int melyseg = 0;
public pointcut callkiir(LzwBinFa fa, LzwBinFa.Csomopont n, PrintWriter os):call(void LzwBinFa.kiir(LzwBinFa.Csomopont, PrintWriter)) && args(n,os) && target(fa) && within(LzwBinFa);
after(LzwBinFa fa, LzwBinFa.Csomopont n, PrintWriter os):callkiir(fa,n,os) @@
)
public pointcut hivas(LzwBinFa.Csomopont n, PrintWriter os): call(void LzwBinFa.kiir(LzwBinFa.Csomopont, PrintWriter)) && args(n,os);
after(LzwBinFa.Csomopont n, PrintWriter os) : hivas(n, os){
try{
kiirPre(n, new PrintWriter("preorder.txt"));
}
catch(FileNotFoundException e) {
System.out.println(e);
}
melyseg = 0;
try{
kiirPost(n,new PrintWriter("postorder.txt"));
}
catch(FileNotFoundException e) {
System.out.println(e);
}
}
```

Importáljuk a PrintWriter-t és a FileNotFoundException-t, mivel használjuk ezeket. Majd megmondjuk a programban, hogy ha az LzwBinFa.kiir metódus meghívódik az eredeti programban, akkor elérjük az argumentumait miután ez megvolt próbálja meg lefuttatni a kiirPre() és kiirPost() függvényeket. Itt kaphatunk FNFE-t, ezért egy trycatch-be helyezzük és kezeljük.

```
public void kiirPost(LzwBinFa.Csomopont elem, java.io.PrintWriter os) {
if(elem != null) {
++melyseg;
kiirPost(elem.nullasGyermekek(), os);
kiirPost(elem.egyesGyermekek(), os);
for(int i = 0; i < melyseg; i++){
os.print("---");
}
os.print(elem.getBetu());
```

```
os.print("(");
os.print(melyseg -1);
os.println(")");
--melyseg;
}
}

public void kiirPre(LzwBinFa.Csomopont elem, java.io.PrintWriter os) {
if(elem != null) {
++melyseg;
for(int i = 0; i < melyseg; i++) {
os.print("---");
}
os.print(elem.getBetu());
os.print("(");
os.print(melyseg -1);
os.println(")");
kiirPre(elem.nullasGyernek(), os);
kiirPre(elem.egyesGyernek(), os);
--melyseg;
}
}
```

Végül a 2 függvényünk marad. Az eredeti programban inorder fabejárás történik, azaz először az adott elem baloldali gyerekét dolgozta fel, majd az elemet, végül pedig a jobb oldali gyereket. Ezzel szemben a postorder-nél a sorrend: bal oldali gyerek, jobb oldali gyerek, majd az elem maga. A postordernél pedig az elemet, majd bal és jobb oldali gyereket.

Futtatás után:

```
v 14.26 •
Megnyitás ▾
postorder... Mentés
ki.txt Mentés
Megnyitás ▾
preorder.txt
Mentés
7 bájt)
```

melyseg = 7
átlag = 5.363636363636363
szorás = 1.3618169680781094

melyseg = 7
átlag = 5.363636363636363
szorás = 1.3618169680781094

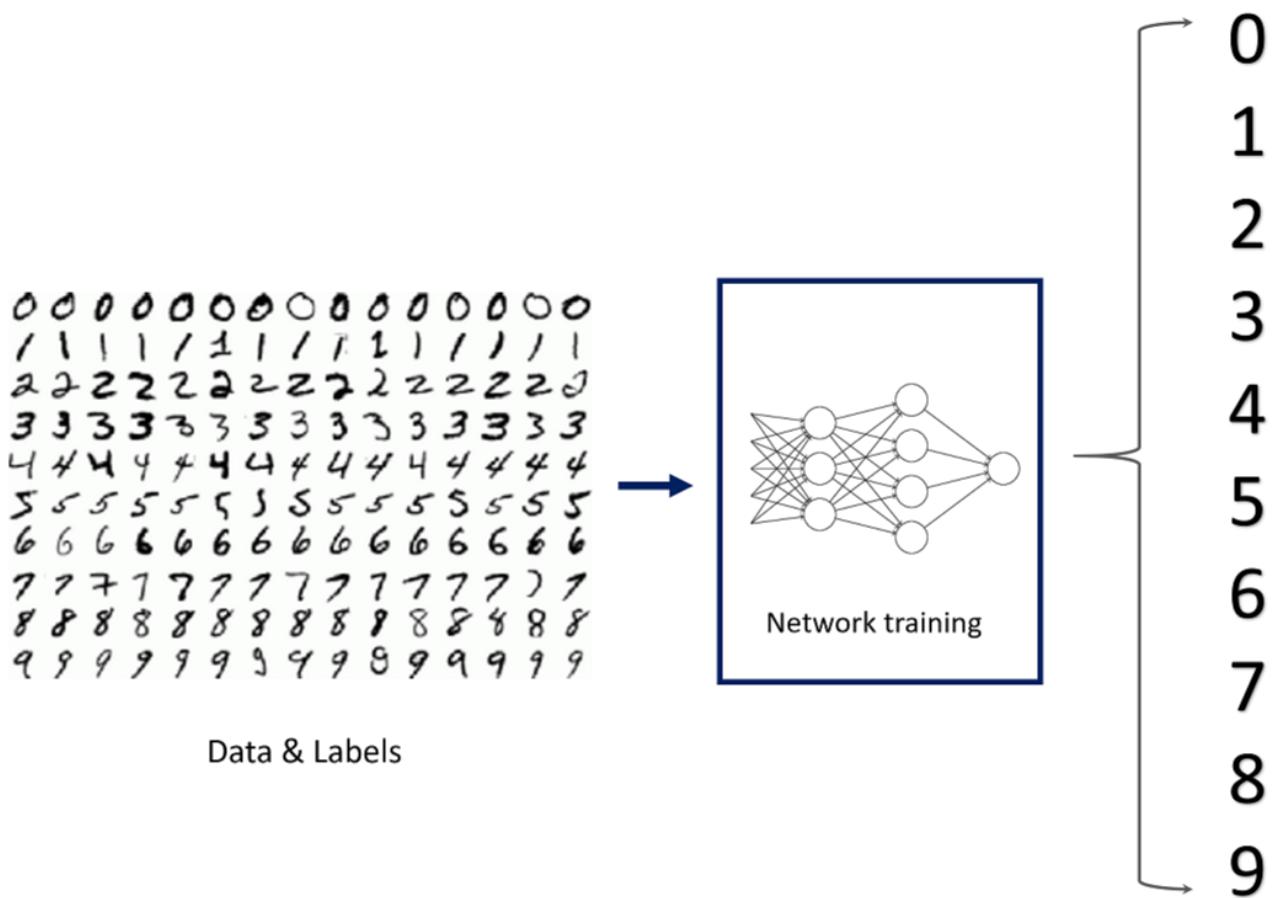
17. fejezet

Helló, Lauda!

17.1. MNIST

Megoldás forrása: https://gitlab.com/baluka94/bhax/blob/master/calvin/mnist/mnist_soft.py

Tanulságok, tapasztalatok, magyarázat... : A feladatban, a tensorflow segítségével kellett megtanítani egy mesterséges intelligenciának azt, hogy felismerjen egy kézzel írott 28x28 pixeles képen lévő számot. A program az MNIST könyvtárat használja a tanuláshoz. Az MNSIT egy adatbázis lényegében, amiben kézzel írott digitalizált arab számjegyek találhatóak meg és gyakran képfeldolgozó programok taníttatására használják, mint most az esetünkben. Széleskörben használják még tanításra és tesztelésre az ún. gépi tanulás (machine learning) területén. 60000 tanító és 10000 tesztelő képet tartalmaz.



A képek mérete a feldolgozásnál kicsi, hogy az osztályozásuk és felismerésük egyszerűbb legyen. A képek feldolgozására egy kialakított neurális hálózatot használunk. A programon néhány módosítást kellett végrehajtanunk, mivel a tensorflow azóta rengeteget változott. A függvényhívásokhoz csatolni kellett, hogy compat.v1. és működőképesek lettek. A 46. sorban lévő függvényhívást a következőre kellett módosítani:

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

Ez azt jelenti, hogy a saját képünk grayscale-es változatát használja a program. Ennek 2 oka van. Az első, hogy az mnist képei grayscale-ek, a másik pedig, ha ezt a változtatást nem tesszük meg, akkor hibát fogunk kapni. Ezután a 72. sorban változtatunk, ahol a paramétereket nevesítenünk kellett.

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

Nézzük a kódot:

```
tf.compat.v1.disable_eager_execution()
x = tf.compat.v1.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
y_ = tf.compat.v1.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=y, labels=y_))
train_step = tf.compat.v1.train.GradientDescentOptimizer(0.5).minimize(
```

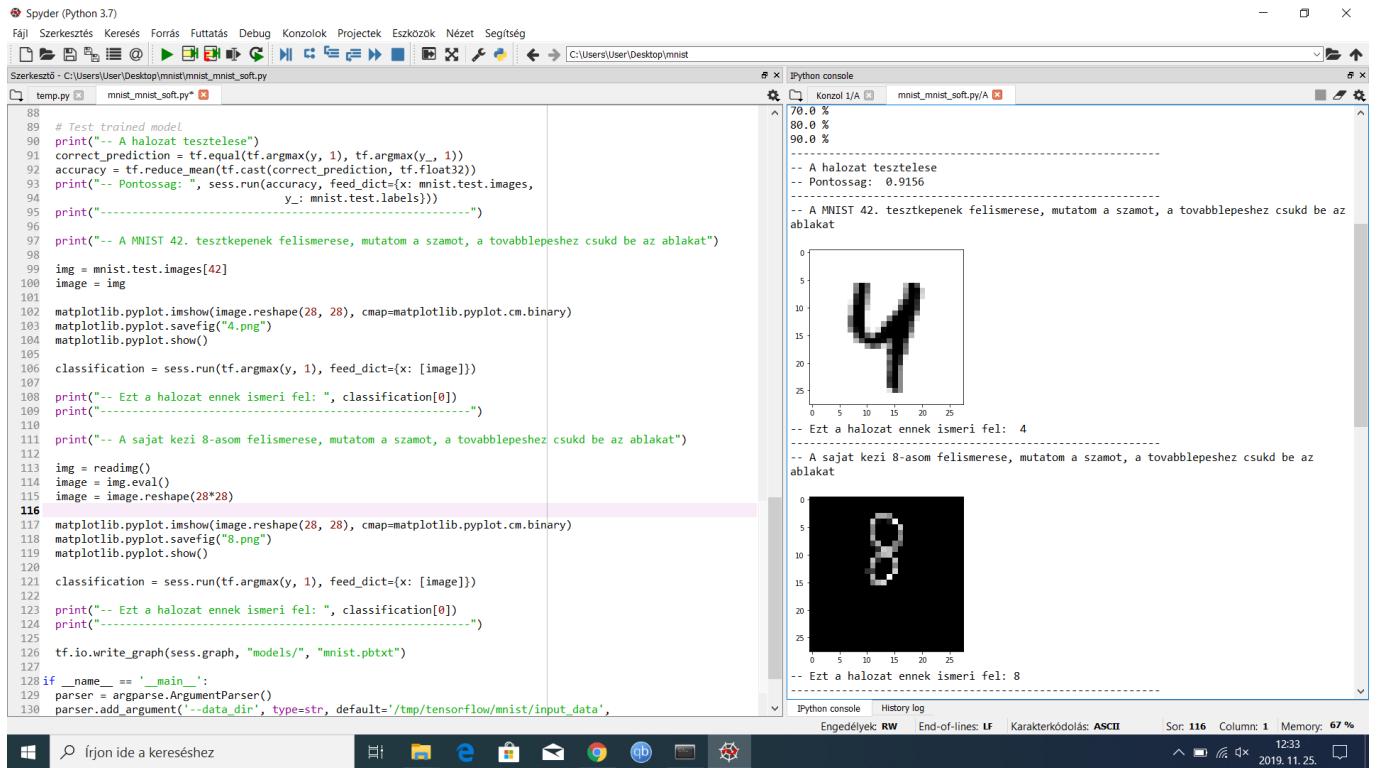
```
cross_entropy)
tf.compat.v1.initialize_all_variables().run()
print("-- A halozat tanítása")
for i in range(1000):
batch_xs, batch_ys = mnist.train.next_batch(100)
sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
if i % 100 == 0:
print(i/10, "%")
print("-----")
# Test trained model
print("-- A halozat tesztelése")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("-- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.test.images,
y_: mnist.test.labels}))
print("-----")
```

Eltároljuk a képet először, ami 28x28 azaz 784 pixel. Elkezdjük a program tanítását for ciklussal. Ezer képpel tanítjuk és 100 képenként kiírjuk, hogy hol járunk. Ezek után a pontosságát ellenőrizzük és ezt tudatjuk a felhasználóval. Majd megpróbálunk felismeretni vele egy mnist képet, majd az általunk kézzel rajzolt 8-ast.

```
img = mnist.test.images[42]
image = img
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm
    .binary)
matplotlib.pyplot.savefig("4.png")
matplotlib.pyplot.show()
classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image] })
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")
print("-- A saját kezi 8-asom felismerése, mutatom a számot, a
továbblepéshez csukd be az ablakat")
img = readimg()
image = img.eval()
image = image.reshape(28*28)
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm
    .binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()
classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image] })
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")
tf.io.write_graph(sess.graph, "models/", "mnist.pbtxt")
```

Mindkét esetben a képeket elmentjük, majd kirajzoljuk őket a matplotlib.pyplot.show() függvénytel. Ez után a program meghatározza, hogy melyek ismeri fel a számot és kiírjuk a konzolra az eredményt. Az

eredmény:

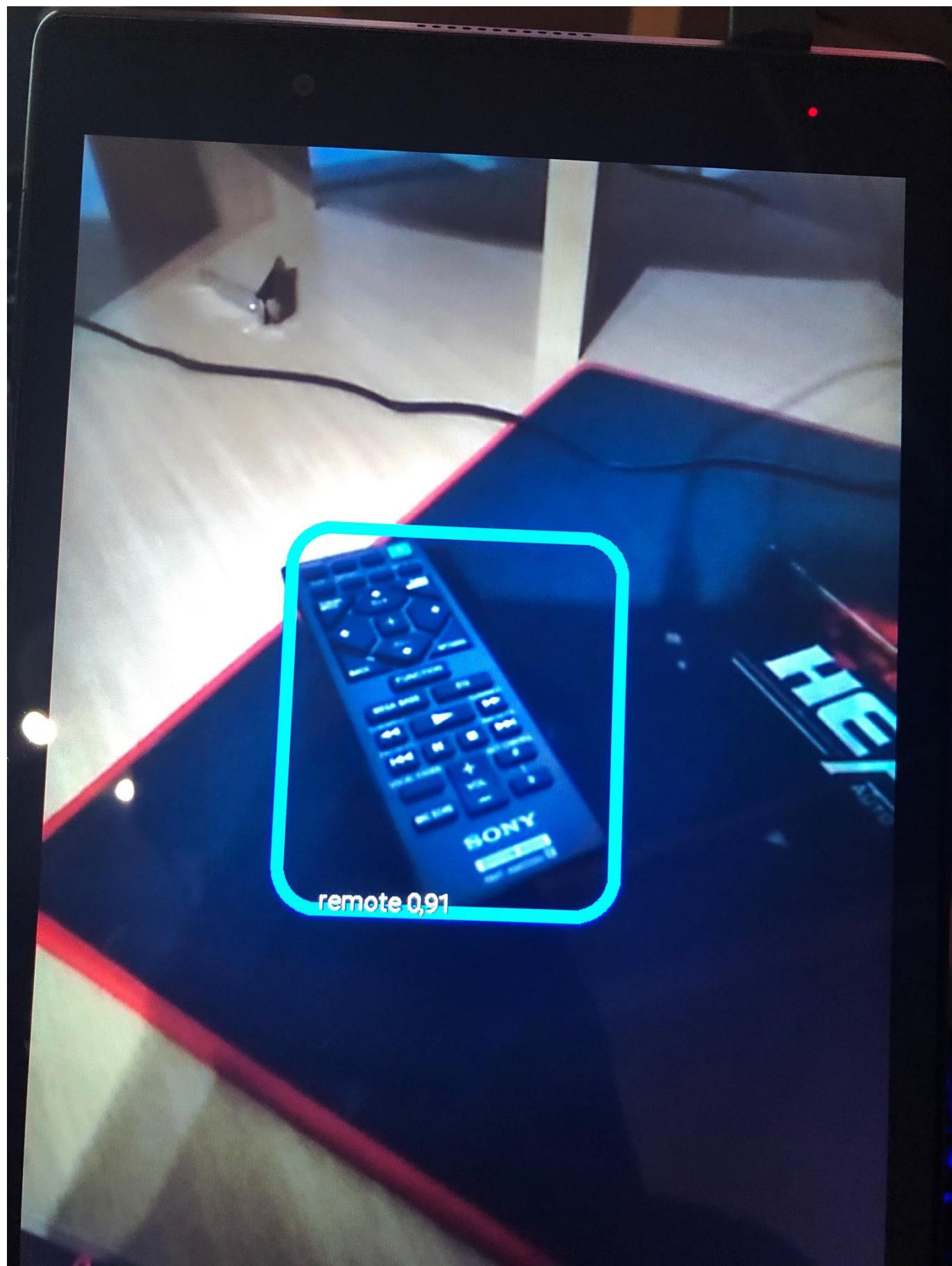


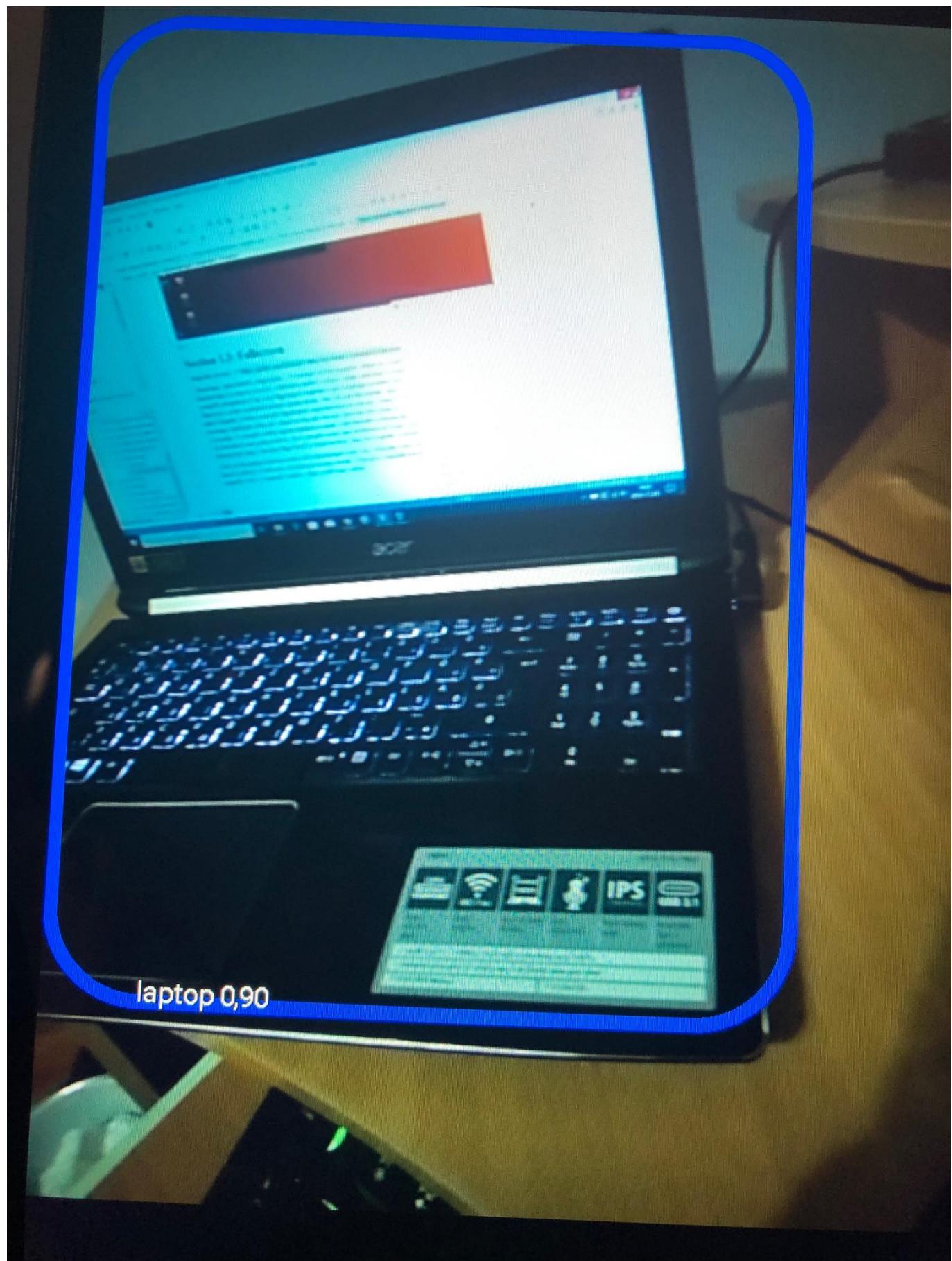
```
88 # Test trained model
89 print("-- A halozat tesztelése")
90 correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
91 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
92 print("-- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.test.images,
93                                     y_: mnist.test.labels}))
94 print("-----")
95 print("-- A MNIST 42. tesztképenek felismereése, mutatom a számot, a továbblepeshez csukd be az ablakat")
96
97 img = mnist.test.images[42]
98 image = img
99
100 matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
101 matplotlib.pyplot.savefig("4.png")
102 matplotlib.pyplot.show()
103
104 classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})
105
106 print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
107 print("-----")
108 print("-- A saját kezi 8-asom felismereése, mutatom a számot, a továbblepeshez csukd be az ablakat")
109
110 img = reading()
111 image = img.eval()
112 image = image.reshape(28*28)
113
114 matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
115 matplotlib.pyplot.savefig("8.png")
116 matplotlib.pyplot.show()
117 classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})
118 print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
119 print("-----")
120 tf.io.write_graph(sess.graph, "models/", "mnist.pbtxt")
121
122 if __name__ == '__main__':
123     parser = argparse.ArgumentParser()
124     parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/mnist/input_data',
125     help='Path to the directory containing the MNIST data')
126     args = parser.parse_args()
127
128     # Create a session
129     sess = tf.Session()
130
131     # Import the trained graph
132     saver = tf.train.import_meta_graph('models/mnist.pbtxt')
133     saver.restore(sess, 'models/mnist.pbtxt')
134
135     # Get the input and output tensors
136     x = tf.get_default_graph().get_tensor_by_name('Placeholder:0')
137     y = tf.get_default_graph().get_tensor_by_name('Softmax:0')
138
139     # Run the session
140     result = sess.run(y, feed_dict={x: np.reshape(np.array([image]), (1, 784))})
141
142     # Print the result
143     print(result)
```

17.2. Android telefonra a TF objektum detektálója

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...: A tensorflow object detect kipróbálása volt a feladat. Ez egy objektum észlelő program. A program elég nagy valószínűséggel képes megállapítani a felvett dolgokat, mint ahogy az a képeken is látszik. A távirányítót 91%-os pontossággal, a laptopot pedig 90%-ossal képes volt megállapítani. A programnak még vannak hiányosságai, de nagyon érdekes volt látni, hogy ilyesmit is létre lehet hozni.





17.3. Deep MNIST

Megoldás forrása: <https://gitlab.com/baluka94/bhax/blob/master/calvin/mnist/deepmnist.py>

Tutorált: Rácz András

Tanulságok, tapasztalatok, magyarázat...: A fentebb megoldott tensorflow-os feladatunk deepmnist-es változatának megírása volt a feladat. A programkódban a : 76,101,111,117,121,124,135,144,145,147,148,179 sorokban kellett módosításokat végezni. A sorokban a tf és a függvény neve közé .compat.v1-et kellett írni és már működő képes. Bele kellett helyezni, hogy felismerje a saját kézzel rajzolt képet, amit az előző programból kimásolhatunk.

A modellt a program hozza létre és mivel grayscale-es képeket fogad el, ezért átalakítjuk őket ennek megfelelően.

```
with tf.name_scope('reshape'):
    tf.compat.v1.disable_eager_execution()
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

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

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

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

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

    with tf.name_scope('adam_optimizer'):
        train_step = tf.compat.v1.train.AdamOptimizer(1e-4).minimize( ←
            cross_entropy)

    with tf.name_scope('accuracy'):
        correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
        correct_prediction = tf.cast(correct_prediction, tf.float32)
    accuracy = tf.reduce_mean(correct_prediction)

    graph_location = tempfile.mkdtemp()
    print('Saving graph to: %s' % graph_location)
    train_writer = tf.compat.v1.summary.FileWriter(graph_location)
    train_writer.add_graph(tf.compat.v1.get_default_graph())
```

Ezután a tanítás következik. 20000 képpel tanítjuk és 100 képenként kiíratjuk, hogy hol járunk, illetve a képek felismerésének pontosságát. Ez egy elég kis számmal fog kezdődni, de minnél több képet dolgoz fel, annál jobban fogja a képeket felismerni és ez az érték annál jobban megnő.

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

A tanítás után megmutatjuk neki a saját képünket, amit remélhetőleg fel is ismer. A tanítási idő itt megnőtt az előző feladatéhoz képest.

Futtatásról kép:



IV. rész

Irodalomjegyzék

DRAFT

17.4. Általános

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

17.5. C

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

17.6. C++

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

17.7. 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 UDPORG 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 érdeklődésekért és hasznos észrevételeikért.

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