

Univerzális programozás

Így neveld a programozód!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert ÁCs Földesi, Zoltán	2019. december 4.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai

Ajánlás

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

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

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	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	8
2.3. Változók értékének felcserélése	10
2.4. Labdapattogás	11
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	14
2.6. Helló, Google!	15
2.7. 100 éves a Brun téTEL	17
2.8. A Monty Hall probléma	18
3. Helló, Chomsky!	21
3.1. Decimálisból unárisba átváltó Turing gép	21
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	21
3.3. Hivatalos nyelv	23
3.4. Saját lexikális elemző	24
3.5. l33t.l	25
3.6. A források olvasása	27
3.7. Logikus	29
3.8. Deklaráció	29

4. Helló, Caesar!	31
4.1. int *** háromszögmátrix	31
4.2. C EXOR titkosító	32
4.3. Java EXOR titkosító	34
4.4. C EXOR törő	35
4.5. Neurális OR, AND és EXOR kapu	37
4.6. Hiba-visszaterjesztéses perceptron	40
5. Helló, Mandelbrot!	42
5.1. A Mandelbrot halmaz	42
5.2. A Mandelbrot halmaz a std::complex osztállyal	44
5.3. Biomorfok	46
5.4. A Mandelbrot halmaz CUDA megvalósítása	49
5.5. Mandelbrot nagyító és utazó C++ nyelven	52
5.6. Mandelbrot nagyító és utazó Java nyelven	54
6. Helló, Welch!	57
6.1. Első osztályom	57
6.2. LZW	59
6.3. Fabejárás	64
6.4. Tag a gyökér	65
6.5. Mutató a gyökér	67
6.6. Mozgató szemantika	68
7. Helló, Conway!	69
7.1. Hangyszimulációk	69
7.2. Java életjáték	73
7.3. Qt C++ életjáték	73
7.4. BrainB Benchmark	77
8. Helló, Schwarzenegger!	81
8.1. Szoftmax Py MNIST	81
8.2. Mély MNIST	84
8.3. Minecraft-MALMÖ	84

9. Helló, Chaitin!	85
9.1. Iteratív és rekurzív faktoriális Lisp-ben	85
9.2. Gimp Scheme Script-fu: króm effekt	86
9.3. Gimp Scheme Script-fu: név mandala	86
10. Helló, Gutenberg!	87
10.1. Programozási alapfogalmak	87
10.2. Programozás bevezetés	88
10.3. Programozás	90
III. Második felvonás	91
11. „Helló, Berners-Lee!”	93
11.1. Python	93
11.2. C++,java	94
12. Helló, Arroway!	97
12.1. OO szemlélet	97
12.2. Homokatózó	100
12.3. Gagyí	107
12.4. Yoda	108
12.5. Kódolás from scratch	109
13. Helló, Liskov!	113
13.1. Liskov helyettesítés sértése	113
13.2. Szülő-gyerek	115
13.3. Ciklomatikus komplexitás	117
13.4. Anti OO	118
13.5. Hello, Android!	119
14. Helló, Mandelbrot!	121
14.1. Reverse engineering UML osztálydiagram	121
14.2. Forward engineering UML osztálydiagram	121
14.3. Egy esettan	123
14.4. BPMN	126
14.5. TeX UML	127

15. Helló, Chomsky!	129
15.1. Encoding	129
15.2. Paszigráfia Rapszódia OpenGL full screen vizualizáció	130
15.3. Paszigráfia Rapszódia LuaLaTeX vizualizáció	131
15.4. Full screen	132
15.5. Perceptron osztály	134
16. Helló, Stroustrup!	136
16.1. JDK osztályok	136
16.2. RSA és összefoglalás	138
16.3. Változó argumentumszámú ctor	142
17. Helló, Gödel!	144
17.1. STL map érték szerinti rendezése	144
17.2. Gengszterek	146
17.3. GIMP Schme hack	147
17.4. Alternatív Tabella rendezése	152
18. Helló,!	156
18.1. SamuCam	156
18.2. BrainB	159
18.3. OOCWC Boost ASIO hálózatkezelése	161
18.4. OSM térképre rajzolás	162
19. Helló, Lauda	166
19.1. Port scan	166
19.2. Junit teszt	167
19.3. Android játék	169
19.4. AOP	174
20. Helló, Calvin	176
20.1. Minecraft MALMÖ	176
20.2. MNIST	178
20.3. TF telefonra	180

IV. Irodalomjegyzék	184
20.4. Általános	185
20.5. C	185
20.6. C++	185
20.7. Lisp	185

DRAFT

Előszó

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

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz 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?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk?

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán 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
- C kapcsán a [**KERNIGHANRITCHIE**] könyv adott részei.
- C++ kapcsán a [**BMECPP**] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a [The GNU C Reference Manual](#), mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipete! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [**BMECPP**] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.
- Kódjátszma, <https://www.imdb.com/title/tt2084970/>, benne a **kódtörő feladat** élménye.

- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

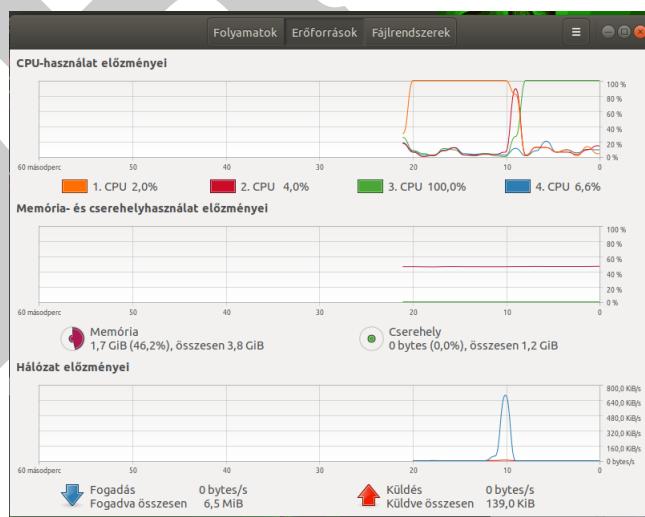
Írj olyan C végtelen ciklusokat, amelyek 0 illetve 100 százalékban dolgoztatnak egy magot és egy olyat, amely 100 százalékban minden magot!

Megoldás forrása:

Kód:

```
#include <stdio.h>

int main(){
while(1){
}
```



Magyarázat: 1.Egy szál 100%-osan futtatása: A main függvényben lévő while ciklus mindaddig lefog futni, amíg a while utáni zárójelben lévő feltétel igaz lesz. Jelen esetben a feltételnek '1' van beírva, ezt a gép minden igaznak tekinti(a 0-át pedig hamisnak). Ezért ez a ciklus mindenkorán futni fog amíg manuálisan ki nem iktatjuk. Ez a ciklus 1 szálat fog folyamatosan 100%-on futtatni. Kód:

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

int main(){
while (1){
sleep(1);
}
}
```

Folyamatnév	Felhasználó	% CPU	Azonosi Memória	Összes lemez	Összes lemez	Lemez olv.
Xorg	foldeiszoltan	2	2923	20,7 MiB	3,3 MiB	104,0 KiB
v2	foldeiszoltan	0	4802	64,0 KiB	—	—
update-notifier	foldeiszoltan	0	4235	4,9 MiB	1,6 MiB	5,7 MiB
systemd	foldeiszoltan	0	2773	1,6 MiB	28,4 MiB	420,0 KiB
sublime_text	foldeiszoltan	0	4295	24,9 MiB	2,3 MiB	—
ssh-agent	foldeiszoltan	0	3326	320,0 KiB	—	—
(sd-pam)	foldeiszoltan	0	2774	2,8 MiB	—	—
pulseaudio	foldeiszoltan	0	3461	3,7 MiB	416,0 KiB	8,0 KiB
plugin_host	foldeiszoltan	0	4324	13,2 MiB	6,1 MiB	—
nautilus-desktop	foldeiszoltan	0	3927	32,4 MiB	11,1 MiB	544,0 KiB
nautilus	foldeiszoltan	0	4034	26,8 MiB	4,8 MiB	264,0 KiB
kdeconnectd	foldeiszoltan	0	3946	6,9 MiB	24,8 MiB	—
ibus-x11	foldeiszoltan	0	3649	4,6 MiB	—	—
ibus-portal	foldeiszoltan	0	3653	436,0 KiB	—	—
ibus-engine-simple	foldeiszoltan	0	4051	648,0 KiB	8,0 KiB	—
ibus-dconf	foldeiszoltan	0	3647	652,0 KiB	—	—
ibus-daemon	foldeiszoltan	0	3643	1,7 MiB	8,0 KiB	4,0 KiB
gvfs-udisks2-volume-monitor	foldeiszoltan	0	3678	1,6 MiB	—	—
gvfs-ftp-volume-monitor	foldeiszoltan	0	3756	652,0 KiB	—	—
gvfs-gphoto2-volume-monitor	foldeiszoltan	0	3742	772,0 KiB	748,0 KiB	—
gvfs-goa-volume-monitor	foldeiszoltan	0	3685	552,0 KiB	—	—
gvfsd-trash	foldeiszoltan	0	3989	1,0 MiB	426,0 KiB	—

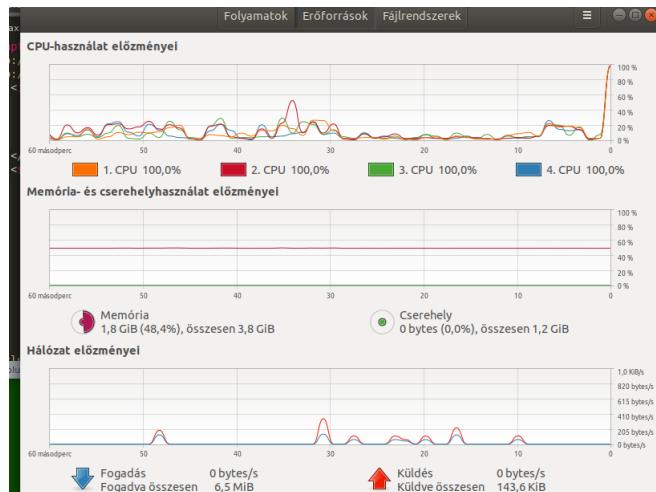
Magyarázat: 2.Egy szál 0%-on való futtatása: Az előzőhöz képest itt is egy végtelen while ciklusunk van a main függvényben, azonban itt megadunk egy sleep nevű utasítást is. A sleep működéséhez include-olunk kell az unistd.h nevezetű header file-t is. A sleep függvény utáni zárójelben megadhatjuk,hogy az adott szálat hány másodpercig altassa a program.Ezt használhatjuk még programok késleltetésére is. Kód:

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

int main () {

#pragma omp parallel

    while (1) {
}
}
```



Magyarázat: 3. minden szál futtatása Ahhoz, hogy minden szálat lefoglaljon a programunk, párhuzamossá kell tennünk a program futását. Ehhez include-oljuk az openmp(Open Multi-Processing) alkalmazásprogramozási felületet(API-t). Ezt az omp.h header file-al tehetjük meg. Ezáltal a programunk párhozamosan fog futni több szálon. A main függvényünk kibővült a #pragma omp parallel sorral. Ez a sor fogja a while ciklusban lévő utasításokat az összes szálra irányítani. Így a végtelen ciklusunk az összes szalon fog egyszerre futni. Ahhoz, hogy ez a program leforduljon használnunk kell a -fopenmp kapcsolót is.

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 pszeudódójára:

```
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.

Ha T1000-ben meghívjuk saját magát, és van benne végtelen ciklus akkor a Lefagy függvény igazat ad vissza ha nincs akkor pedig hamisat.A Lefagy2 pedig a Lefagy függvényre épül és ha a Lefagy igaz értéket ad vissza akkor a Lefagy2igazat ad vissza, ha a Lefagy hamisat,azaz nincs végtelen ciklus a programban akkor a Lefagy2 pedig belép egy végtelen ciklusba,tehát ha nem is volt benne végtelen ciklus mostmár lesz így ellentmondás jön létre.Tehát ilyen program tényleg nem létezhet.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Kód:

```
#include <stdio.h>

int main()
{
    int a=7;
    int b=1;

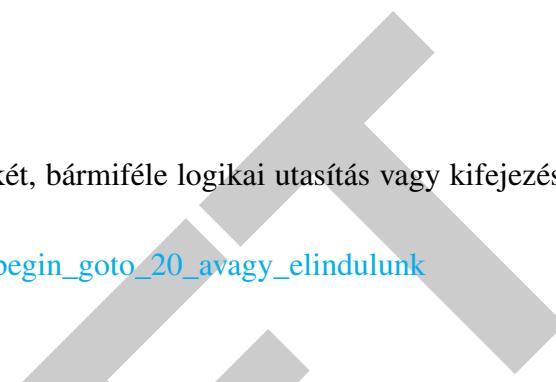
    printf("Valtozok csere elott\n");

    printf("%d\n",a);
    printf("%d\n",b);

    a=a+b;
    b=a-b;
    a=a-b;

    printf("Valtozok csere utan\n");
    printf("%d\n",a);
    printf("%d\n",b);

    return 0;
}
```



```
Foldesizoltan@Aspire:~/Asztal/prog1/Turing
Fájl Szerkesztés Nézet Keresés Terminal Súgó
Foldesizoltan@Aspire:~/Asztal/prog1/Turing$ ./valtcserkul
Valtozok csere elott
7
1
Valtozok csere utan
1
7
Foldesizoltan@Aspire:~/Asztal/prog1/Turing$
```

Magyarázat: Változók cseréje különbséggel: A main függvényünk elején deklarálunk két int típusú változót(a,b). Jelen esetünkben a=7 és b=1. A printf függvény segítségével tudunk a konzolra kiíratni. A printf utáni zárójelben idézőjelek közé írhatjuk be amit ki szeretnénk íratni. A %d kapcsoló azt adja meg, hogy az utána írt 'a' változót egész számként fogja kiírni. A \n kapcsoló pedig a sortörés. Maga a változó értékének cseréje egyszerű matematika. Az 'a' értéke kezdetben 7, 'b' értéke pedig 1. Első lépésként 'a' értékét egyenlővé tesszük 'a' és 'b' összegével, így a=7+1=8. Második lépésként 'b' értékét egyenlővé tesszük 'a' és 'b' különbségével, így b=8-1=7. Majd utolsó lépésként 'a' értékét egyenlővé tesszük 'a' és 'b' különbségével, azaz a=8-7=1. Amint látjuk 'a' értéke 1 lett 'b' értéke pedig 7, tehát megcserélődtek. Kód:

```
include <stdio.h>

int main()
{
    int a=7;
    int b=1;

    printf("Valtozok csere elott\n");

    printf("%d\n",a);
    printf("%d\n",b);

    a=a*b;
    b=a/b;
    a=b/a;

    printf("Valtozok csere utan\n");
    printf("%d\n",a);
    printf("%d\n",b);
    return 0;
}
```

```
foldesizoltan@Aspire:~/Asztal/progi/Turing$ ./valtcsereszor
Valtozok csere elott
7
1
Valtozok csere utan
1
7
foldesizoltan@Aspire:~/Asztal/progi/Turing$
```

Magyarázat: Ez a példa annyiban különbözik az előzőtől, hogy itt az összeadás és különbség helyett szorzást és osztást használunk. Az 'a' értéke kezdetben még csak 7, 'b' értéke pedig 1. Első lépésként 'a' értékét egyenlővé tesszük 'a' és 'b' szorzatával, a=7*1=7. Második lépésként 'b' értékét egyenlővé tesszük 'a' és 'b' hányadosával, b=7/1=7. Majd utolsó lépésként 'a' értékét egyenlővé tesszük 'b' és 'a' hányadosával azaz a=7/7=1. Ezek a változó cserék a régebbi időkben voltak hasznosak, ugyanis így nem kellett felesleges erőforrásokat lefoglalni egy segédváltozónak.

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 videót)

kon.)

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

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

Kód:

```
#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;

    for ( ; ; ) {

        getmaxyx ( ablak, my , mx );
        mvprintw ( y, x, "O" );

        refresh ();
        usleep ( 100000 );

        x = x + xnov;
        y = y + ynov;

        if ( x>=mx-1 ) {
            xnov = xnov * -1;
        }
        if ( x<=0 ) {
            xnov = xnov * -1;
        }
        if ( y<=0 ) {
            ynov = ynov * -1;
        }
        if ( y>=my-1 ) {
            ynov = ynov * -1;
        }
    }
}
```

```
    return 0;  
}
```

Magyarázat: A programunk futásához include-olnunk kell a curses.h nevezetű header file-t, melyben a különböző képernyőkezeléshez szükséges függvények találhatóak. A main függvényünk elején a WINDOW parancccsal adjuk meg az ablak ábrázolását, és a későbbiekben ablak néven érjük el. Az initscr függvénytel "tisztítjuk" le a konzolt. Ezután deklaráljuk az x és y koordinátákat melyek kezdőértékei 0 lesz. Utána az xnov és ynov deklarálása következik melyeknek értékei 1, ezekkel fogjuk növelni a koordinátákat, azaz léptetni a labdánkat. A pattogásunk alapja egy végtelen for ciklus, ezért ez addig fog futni amíg manuálisan ki nem lőjük. A getmaxyx függvény meghatározza a konzolunk maximális koordinátáit. Az mvprintw függvénytel fogunk x és y koordinátákra jelen esetben egy '0'-t írni. Az usleep mögötti zárójelbe beírt szám fogja meghatározni a pattogás gyorsaságát. Minél nagyobb számot írunk be annál lassabb lesz, ugyanis ez a parancs altatja a ciklusunkat a megadott ideig. Ezután az x és y értékét növeljük xnov-el és ynov-el (jelen esetben 1 el). Az ezután következő 4 db if utasítás segít nekünk abban, hogy labdánk a konzolon belül maradjon. Ha elérte a bal, vagy jobb oldalt, akkor az xnov változó értékét beszorozzuk -1 el, tehát előjelet fog váltni, ezáltal a következő körben csökkeni fog x értéke, tehát visszapattan a labdba. Ha pedig az ablak tetejét, vagy alját érte el akkor ugyanezt csináljuk meg csak az ynov változó értékével.

If nélkül:

Megoldás forrása: https://progpater.blog.hu/2011/02/13/megtalaltam_neo_t

```
#include <stdio.h>  
#include <stdlib.h>  
#include <curses.h>  
#include <unistd.h>  
  
int  
main (void)  
{  
    int xj = 0, xk = 0, yj = 0, yk = 0;  
    int mx = 80 * 2, my = 24 * 2;  
  
    WINDOW *ablak;  
    ablak = initscr ();  
    noecho ();  
    cbreak ();  
    nodelay (ablak, true);  
  
    for (;;) {  
        xj = (xj - 1) % mx;  
        xk = (xk + 1) % mx;  
  
        yj = (yj - 1) % my;  
        yk = (yk + 1) % my;  
  
        clear ();  
  
        mvprintw (0, 0,
```

```
" " ←  
-----  
");  
mvprintw (24, 0,  
" " ←  
-----  
");  
mvprintw (abs ((yj + (my - yk)) / 2),  
abs ((xj + (mx - xk)) / 2), "X");  
  
refresh ();  
usleep (150000);  
  
}  
return 0;  
}
```

A fenti kód ugyanúgy egy labdát fog pattogtatni a konzolon, csak ebben az ersetben nem használtunk semmien logikai utasítást. Ehhez a fordításhoz is szükséges a -lncurses kapcsoló. A WINDOW utasítással létrehozunk egy mutatót. Az initscr függvénytelőkészítjük az ablakunkat ahol pattogni fog a labdánk. A kód alapja egy végtelen for ciklus, vagyis a labdánk mindenkorban pattogni fog amíg manuálisan ki nem lőjük a programot. Az mvprintw függvényekkel kezeljük a koordinátákat. Először x=0-nál kirajzolunk kötőjeleket majd x=24-nél is, ezek lesznek az ablakunk határai. Ezután matematikai képletek segítségével kiszámoljuk a koordinátákat, és kiíratjuk őket.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

Kód:

```
#include <stdio.h>  
  
int main()  
{  
    int a=1;  
    int count=0;  
do  
count++;  
  
    while (a <= 1) ;
```

```
printf("%d", count);
printf("bites a szohossz");
    return 0;
} }
```

Magyarázat: A fenti programunk számítógépünk int típusú változójának méretét fogja megadni. A main függvényben deklarálunk egy 'a' változót és értékét 1 re állítjuk. Ennek segítségével fogjuk mérni a szóhosszt. Valamint deklarálunk egy count nevezetű változót is, melyet majd növelünk és a programunk végénk kiíratjuk, ez lesz az int hossza. A main fő része egy do while ciklus. Ez, a sima while-al ellentében hálultesztelős utasítás, tehát először fogja növelni a count értékét, és csak azután fogja letesztelni a while utáni feltételt. Ez azért fontos mert így tudjuk elérni, hogy az első shift is bele számítson az eredményünkbe. A szóhosszt az úgy nevezett bitshift operátor segítségével állapítjuk meg. Jelen példánkban a balra shiftelőt használjuk, de van jobbra shiftelő is. Ez az operátor annyit fog tenni, hogy a memoriában eltárolt biteket eggyel balra fogja eltolni és jobbra kiegészíti egy nullával. Tehát például az 'a' értéke egy, ez kettes számrendszerben van eltárolva a memoriában ennek bináris kódja 0001. A bitshift ezt fogja eggyel eltolni tehát 0010 lesz, ami már a 2-es szám. Majd 0100 lesz, ami a 4-es. És mindenegyes shiftnél növeljük a count értékét eggyel. Ezt mindaddig fogja így csinálni amíg az adott memóriacímben csak 0-ások lesznek, tehát az értéke 0 lesz.

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://progater.blog.hu/2011/02/13/bearazzuk_a_masodik_labort#more2657583

Kód:

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

void
kiir (double tomb[], int db)
{
    int i;

    for (i = 0; i < db; ++i)
        printf ("%f\n", tomb[i]);
}

double
tavolsag (double PR[], double PRv[], int n)
{
    double osszeg = 0.0;
    int i;

    for (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 (i = 0; i < 4; ++i)
    {
        PR[i] = 0.0;
        for (j = 0; j < 4; ++j)
            PR[i] += (L[i][j] * PRv[j]);
    }
    if (tavolsag (PR, PRv, 4) < 0.00000001)
        break;

    for (i = 0; i < 4; ++i)
        PRv[i] = PR[i];
}

kiir (PR, 4);

return 0;
}
```

Magyarázat:A PageRank a Google keresőmotorjának legfontosabb eleme.Minél többen hivatkoznak egy oldalra, az annál jobb. A fenti C program négy lap PageRank értékét számolja ki.A kódunkhoz szükséges a math.h header file include-olása,ebben matematikai függvények találhatóak, pl.sqrt.A kód elején található egy void típusú függvény.A void annyit takar, hogy a függvény nem fog visszatéríteni semmilyen értéket.A függvény neve után zárójelek között adjuk meg a függvény paramétereit, jelen esetben a függvény egy double típusú tömböt, illetve egy int típusú változót vár. Ez a függvény egy for ciklus segítségével fogja kiíratni a konzolra a kapott tömb elemeit.A következő függvény már double típusú,tehát egy double típusú

változót fog visszaadni. Paraméterként pedig két tömböt illetve egy egész számot vár. Ez a függvény fogja számolni a távolságot. A két tömb elemeit egyesével kivonja majd lényegében négyzetre emeli. Majd a függvény végén gyököt vonunk az sqrt segítségével, ezáltal biztos, hogy pozitív eredményt kapunk. A main függvényben lévő 'L' nevezetű mátrix mondja meg, hogy melyik oldalra hányan mutatnak linkekkel. A main függvény fő része egy végtelen for ciklus. A számítást egy másik for ciklussal végezzük el, ez egy sima mátrix szorzás. Ezután az így kapott eredményt átadjuk a tavolság függvénynek, és ha az így visszakapott érték kisebb mint 0.00000001 akkor megszakítjuk a ciklust és átadjuk az értékeket kiíratásra.

2.7. 100 éves a Brun tétele

Írj R szimulációt a Brun tétele demonstrálására!

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

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

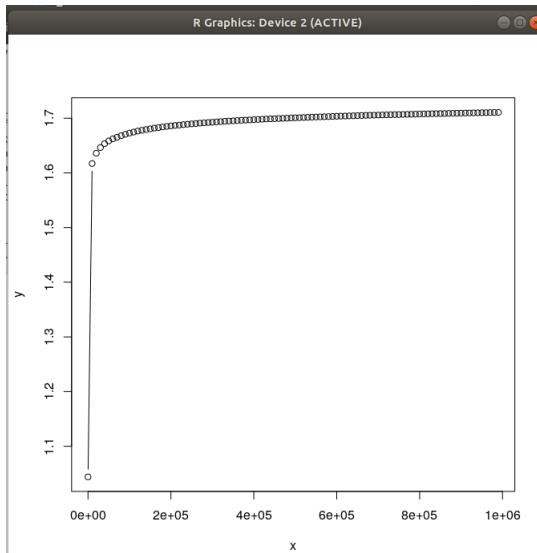
A Brun tétele kimondja, hogy az ikerprímek (olyan prímpárok melyeknek különbsége 2) reciprokösszege egy véges értékhez konvergál. A következő kód ennek a tételenek R-beli megvalósítása.

```
library(matlab)

stp <- function(x) {

  primes = primes(x)
  diff = primes[2:length(primes)] - primes[1:length(primes)-1]
  idx = which(diff==2)
  t1primes = primes[idx]
  t2primes = primes[idx]+2
  rt1plust2 = 1/t1primes+1/t2primes
  return(sum(rt1plust2))
}

x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```



Ahhoz,hogy kódunk működjön telepítenünk kell a matlab csomagot, ezt megtehetjük az intsall.packages("matlab") parancssal.Ezután ezt a csomagot a library(matlab) parancssal be is töltjük. Ezek után stp néven létrehozunk egy függvényt amely paraméterül kap majd egy számot.primes néven létrehozunk egy vektort amelyben eltároljuk a paraméterként kapott számig a prímszámokat. Ezután a diff vektorban kiszámoljuk a prím párok különbségét, ezt úgy tesszük meg,hogy először eltároljuk a primes vektor második tagjától a vektor végéig a prímeket, majd ezekből kivonjuk a primes vektor első tagjától az utolsó előtti tagjáig a prímeket.A következő sorban az idx vektorban eltároljuk azoknak a vektoroknak az indexét ahol a különbség kettő(tehátikerpímek).A t1primes vektorban eltároljuk az így kapott prímpárok első tagját, a t2primes-ban pedig a párok második tagját.Az rt1plus2 vektorba kiszámoljuk a két vektor reciprokösszegét, majd return-el ezeknek az összegüket íratjuk ki. Az utolsó három sorral pedig ki is rajzoltatjuk az így kapott eredményt.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Tutor:Kiss Máté

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

A Monty Hall paradoxon egy az USA-ban futó televíziós vetélkedőn alapul.A játékost megkéri a műsorvezető, hogy válasszon 3 csukott ajtó közül. A három ajtó közül az egyik mögött egy nagyon értékes nyeremény van, a másik kettő pedig valami nagyon értéktelen. A játékos választása után a műsorvezető a másik két ajtó közül kinyit egyet ami mögött biztosan nincsen az értékes nyeremény. Ezután a műsorvezető megkérdezi a játékost,hogy szertné-e a másik ajtót választani,vagy marad az eredeti döntésénél.És itt jön a paradoxon. Ugyanis az első választásnál a játékosnak $1/3$ esélye volt arra,hogy az értékés nyereményt választja.Most a józan ész azt sugallja,hogy mindenelyik ajtót választja, úgyis $50\%-50\%$ esélye van a nyereményre,de igazából $2/3$ esélye lenne a nyereményre ha átvált a másik ajtóra.Ezt talán könnyebb elfogadni ha elképzeljük ugyanezt csak 100 ajtóval. Tehát a játékos választ egyet, $1/100$ az esélye arra,hogy eltalálja a nyereményt és $99/100$ az esély arra,hogy valamelyik másik ajtó mögött van. Ezután a játékvezető kinyit 98 ajtót ami mögött nincs semmi és felteszi ugyanazt a kérdést:Váltunk-e ajtót?Talán így könnyebb

belátnunk,hogy sokkal nagyobb esélyünk van nyerni ha váltunk a másik ajtóra, ugyanis az az eredeti 99/100 esély oda koncentrálódik.

A következő kód ennek a problémának R szimulációja:

```
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) {

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


}

valtoztatesnyer = which(kiserlet==valtoztat)

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

Kódunk elején a kiserletek_szama vektorban mentjük el,hogy hányszor hajtuk végre a szimulációt.Ezután a kiserlet vektorba tároljuk el,hogy melyik ajtó mögött lesz a nyeremény, 1-3 ig fog egy számot random választani annyiszor ahányszor szimuláljuk.A játékos vektorban ugyanezt csináljuk, ez fogja megadni a játékos választását.A musorvezeto vektor azt az ajtót fogja tárolni amit a műsorvezető fog kinyitni. Ezt úgy határozzuk meg, hogy indítunk egy for ciklust ami végigmegy az összes létrejött eseten. Ezután egy if segítségével először megnézzük azt az esetet amikor a játékos eltalálta a nyereményt rejtő ajtót, azaz a

kísérlet és a játékos vektor egyenlő. Ha ez az eset áll fent akkor a maradék két ajtó közül random választ egyet a program. Ha pedig nem egyezik meg a választása a játékkosnak a nyereményt rejtő ajtóval akkor, azt a két ajtó kiveszük a választásból és a maradék egy ajtót fogja kinyitni a műsorvezető. Ezután a nem-valtoztatesnyer vektorban eltároljuk azokat az eseteket amikor akkor nyerne a játékos ha nem változtat. A változtat vektorban pedig azokat az eseteket amikor akkor nyerne amikor változtat. Ezután kiíratjuk ezeket.

DRAFT

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>

int main(){
int sz=0;
int a=0;
printf("Adj meg egy számot");
scanf("%d", &sz);
printf("Unárisba atváltva:");
for(int i=0;i<sz;i++)
{
printf("%d", 1);
}

return 0;
}
```

A fent látható program, egy decimális számot fog bekérni a konzolról és azt a számot unáris számrendszerbe fogja kiírni. Az unáris számrendszerbe úgy váltunk át, hogy annyiszor fogunk kiírni egy db egyest amennyi a szám értéke. Tehát pl ha beírjuk, hogy 5 akkor 11111-et fog kiírni. Ezt egy egyszerű for ciklussal tesszük meg. A Turing gép úgy végezné ezt az átváltást, hogy a szám végéről indulva, ha 0-át lát, akkor kilencre vált, és mindenki vonna egy számot, tehát 8,7,6,5,4,3,2,1,0 és a kivont egyeseket eltárolja, majd a folyamat végén kiírja őket.

3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

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

Forrás:<https://slideplayer.hu/slide/2108567/>

S, X, Y legyenek változók
a, b, c legyenek konstansok

S->abc, S->aXbc, Xb->bX, Xc->Ybcc, bY->Yb, aY->aaX, aY->aa

S (S->aXbc)
aXbc (Xb->bX)
abXc (Xc->Ybcc)
abYbcc (bY->Yb)
aYbbcc (aY->aa)
aabbcc

de lehet így is:

S (S->aXbc)
aXbc (Xb->bX)
abXc (Xc->Ybcc)
abYbcc (bY->Yb)
aYbbcc (aY->aaX)
aaXbbcc (Xb->bX)
aabXbcc (Xb->bX)
aabbXcc (Xc->Ybcc)
aabYbbccc (bY->Yb)
aabYbbccc (bY->Yb)
aaYbbbccc (aY->aa)
aaabbccc

A, B, C legyenek változók
a, b, c legyenek konstansok

A->aAB, A->aC, CB->bCc, cB->Bc, C->bc

A (A->aAB)
aAB (A->aC)
aaCB (CB->bCc)
aabCc (C->bc)
aabbcc

de lehet így is:

A (A->aAB)
aAB (A->aAB)
aaABB (A->aAB)
aaaABBB (A->aC)
aaaaACBBB (CB->bCc)
aaaabCcBB (cB->Bc)
aaaabCBcB (cB->Bc)
aaaabCBBc (CB->bCc)
aaaabbCcBc (cB->Bc)

```
aaaabbCBcc (CB->bCc)
aaaabbbCccc (C->bc)
aaaabbbbcccc
```

A generatív grammaтика különböző transzformációs szabályokat tartalmaz. Ezekhez szükségünk van egy alap szimbólumra, és ezen a szimbólumon tetszés szerint bármelyik szabályt bármennyiszer elvégezhetünk. Az a lényege, hogy egy fajta szimbólumot egy féleképpen tudunk létrehozni, ha több féleképpen is előtud állni akkor ez a nyelv már nem generatív.

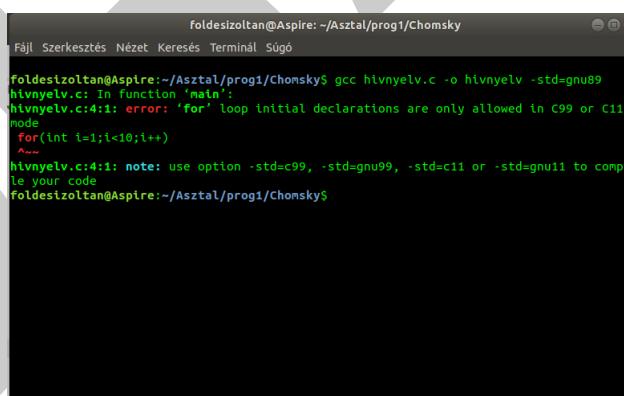
Először megadunk konstansokat és változókat is. Ezután leírjuk a megfelelő nyelvtani szabályokat. Ezt nyilak segítségével tehetjük meg. Bal oldalon megadjuk hogy miből, jobb oldalon meg hogy mi lesz belőle. Ezt rövidebben is láthatjuk alatta. A második generatív nyelv létrehozásánál lényegében elég volt a változók nevét átírni a változók neveit, és már is egy másik generatív nyelvet kapunk amely ugyanazt a nyelvet generálja.

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.

```
#include <stdio.h>

int main(){
for(int i=1;i<10;i++)
printf("lefut");
}
```



1989-ben adták ki a C c89-es szabványát és 1999-ben a c99-et. Ezekben a szabványokban különböző szemantikai szabályok érvényesek. A fenti kódban egy ilyen különbségre láthatunk példát. Első ránézésre valószínű fel sem tűnik benne semmi érdekes. Egy sima for ciklus benne egy utasítással. Azonban a for ciklust a c89-es szabvány szerint így nem adhatjuk meg, ugyanis ebben még tilos volt a zárójelek között deklarálni a ciklusváltozót. Ahhoz, hogy ezt lássuk, a megadott szabvány szerint kell lefordítanunk a programot. Ezt a -std=gnu89 kapcsolóval tehetjük meg. Ez a c89-es szabvány szerint fogja lefordítani a programunkat. Így meg is kapjuk azt a hibakódot miszerint c99 szerint kellenne fordítanunk a programot mert amit próbálunk csinálni az eszerint a szabvány szerint nem megengedett.

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!

Kód:

```
% {  
#include <string.h>  
int szamok_szama =0;  
%}  
%%  
[0-9]+ { ++szamok_szama; }  
  
%%  
  
int  
main()  
{  
yylex();  
printf("%d szam\n", szamok_szama);  
return 0;  
}
```

Erre a bemenetre



Ezt a kimenetet adja:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
földesizoltan@Aspire:~/Asztal/prog1/chomsky/lex$ gcc lex.yy.c -o lex -lfl  
földesizoltan@Aspire:~/Asztal/prog1/chomsky/lex$ ./lex <sor  
  
11 szam  
földesizoltan@Aspire:~/Asztal/prog1/chomsky/lex$
```

Magyarázat:A fent látható kód, a bemenetben megjelenő számokat fogja összeszámolni egy lexer segítségével.Egy lexer a bemenetben megjelenő szövegeket vizsgálja, és megadhatunk neki különböző mintákat,karaktereket amiket keres.A programot három fő részre lehet bontani, ezeket a fő részeket a % jelek határolják. Az első fő részben adhatjuk meg a különböző definíciókat,jelen esetben egy egész típusú változót

hoztunk létre. A második részben adhatjuk meg,hogy mit keressen a lexer, és ha talált olyan karaktert,mit tegyen vele.Jelen esetünkben az egész számokat fogja keresni, és ha talál egyet, akkor a változónk értékét növeli eggyel. A harmadik részben pedig megadjuk, hogy a program végén írassa ki a változó értéket. Ezt a programot először egy .l kiterjesztésbe kell elmentenünk. Fordítása két lépésben történik. Először flex valami.l, ez a parancs létrehoz egy valami.yy.c kiterjeszésű fájlt,majd ezt a fájlt kell c fordítóval lefordítanunk a -lfl kapcsoló segítségével.

3.5. I33t.I

Lexelj össze egy l33t cipher!

Megoldás forrás: https://gitlab.com/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProgChomsky/I337d1c7.l

Tutorált:Kiss Máté

```
%{
#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', {"1", "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"}}  
  
};  
  
%}  
%%  
. {  
  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
{  
  
        if(1337d1c7[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int) (100.0*rand() / (RAND_MAX+1.0));  
  
            if(r<91)  
                printf("%s", 1337d1c7[i].leet[0]);  
            else if(r<95)  
                printf("%s", 1337d1c7[i].leet[1]);  
            else if(r<98)  
                printf("%s", 1337d1c7[i].leet[2]);  
            else  
                printf("%s", 1337d1c7[i].leet[3]);  
  
            found = 1;  
            break;  
        }  
  
        if(!found)
```

```
printf("%c", *yytext);

}

%%

int
main()
{
    srand(time(NULL)+getpid());
    yylex();
    return 0;
}

}
```

Magyarázat:A leet egy főleg az interneten elterjedt rejtnyelv. Különböző számokkal és jelekkel helyettesítenek betűket, az alapján h mihez hasonlítanak.A fenti kód egy ilyen szöveget alakít olvashatóvá, vagy sima szöveget alakít át.Ennek a programnak is a lexer az alapja csak úgy mint az előző feladatnál.

A kód elején megadtunk egy struktúrát, ez tartalmazni fog egy c változót, ez lesz a betű amit át szeretnénk majd alakítani. Illetve tartalmazni fog még egy négy pointerból álló tömböt.Ezután feltöljük a tömböt. A kód második részében adjuk meg,hogy mit csináljon a karakterekkel. A pont a bal oldalon azt jelenti, hogy bármilyen karaktert észlel a mellette lévő kód részlet lefog futni.Ha bármilyen karaktert észlel akkor megfogja keresni a tömbben. Ezután egy random számot fog generálni. Ha az a random szám kisebb lesz mint 91 akkor a helyettesítő karakter tömb első karakterével fogja helyettesíteni az adott karaktert. Ha kisebb mint 98 akkor a másodikkal és így tovább. Tehát általában az első helyettesítővel fogja helyettesíteni. A main függvényben csak a random számgenerátort inicializáljuk illetve elindítjuk a karakter beolvasást.

3.6. A források olvasása

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

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

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



Bugok

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

- i.

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

Ha a SIGINT jelkezelése nem volt figyelmen kívül hagyva akkor most se legyen. Ha figyelmen kívül volt hagyva akkor a jelkezelő függvény kezelje.

ii.

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

Ez egy for ciklus.A ciklusváltozónk az 'i' melynek kezdőértéke nulla.A ciklus addig fog futni amíg az i kisebb lesz mint 5, és minden ciklus végén az 'i'-t növeljük eggyel.

iii.

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

Ez a for ciklus első ránézésre olyan mint az előző, azonban van egy különbés. Az előző ciklusnál ++i volt most pedig i++. Ennek a for ciklusnál nincs nagy jelentősége,mindkettő növelni fogja eggyel i értékét. Azonban vannak olyan helyzetek amikor nagy jelenősége lehet.Pl i=1 v=i++ és v=++i. Az elsőnél v értéke 1 lesz miközben i értéke kettő, mert hamarabb fogja v felvenni i értékét minthogy növelné eggyel i értékét. A másodiknál pedig előbb növeli, aztán teszi egyenlővé, szóval kettő lesz v értéke.

iv.

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

Egy tömböt tölt fel i értékeivel. Jelen esetben egy öt elemű tömböt hoz létre.

v.

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

Ebben a for ciklusban az i kisebb mint n mellett van még egy feltétel ,és a ciklus csak akkor fog lefutni ha minden feltétel igaz lesz .A második feltétel nem lesz jó,mert nem logika értéket fog visszaadni(== kellene).

vi.

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

A printf segítségével kiíratunk két egész számot melyeket egy egy függvény fog meghatározni.

vii.

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

Kiíratunk kt egész típusú számot, az egyik egy függvény fogja meghatározni.

viii.

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

Ugyanaz mint az előző csak itt az f függvény egy memóriacímét fog kapni

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 \text{ prim})) \leftrightarrow
```

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

```
$ (\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

Tutor: Kiss Máté

Ennek a feladat megoldásához a következő logikai összekötőjeleket kell ismernünk: \neg=negáció(tagadás), \vee=vegyeletek, \wedge=összekötő, \exists=létezik, \forall= minden, \supset=superset, \subset=subset, \neg\subset=\supset\neq, \neg\supset=\subset\neq. A kvantorok \exist=létezik, és a minden =\forall. Az első: Bármelyik x számhoz, tartozik egy olyan y prím szám, ami x-nél kisebb. Második: Bármely x szám mellett van olyan y prím és attól 2-vel nagyobb prím ami nagyobb mint az x. Harmadik: Ha van y prím szám, akkor bármelyik x nála kisebb lesz. Negyedik: Ha van olyan y amitől bármelyik x nagyobb akkor az x biztosan nem prím.

3.8. Deklaráció

Vezess be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referencia
- egészek tömbje
- egészek tömbjének referencia (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 int típusú, azaz egész számot vezet be, melynek neve a lesz

- ```
int *b = &a;
```

A 'b' tárolni fogja 'a' memóriacímét, azaz 'b' 'a'-ra mutató mutató lesz.

- ```
int &r = a;
```

Itt az r nem vátlozó hanem egy memóriacím, mégpedig 'a' memóriacíme.

- ```
int c[5];
```

Egy egész típusú 5 elemű tömb deklarálása. Az indexelés 0-tól kezdődik.

- ```
int (&tr)[5] = c;
```

c 5 elemű tömg, minden eleméhez referenciát rendel.

- ```
int *d[5];
```

d 5 elemű tömb összes tagja egy mutató lesz.

- ```
int *h();
```

int típusú pointert visszaadó függvény.

- ```
int *(*l)();
```

Egész típusra mutató pointert visszaadó függvény

- ```
int (*v(int c))(int a, int b)
```

Egészet visszaadó, két egészet kapó és függvényre mutató pointert visszaadó függvény.

- ```
int (*(*z)(int))(int, int);
```

Függvénymutatót visszaadó, ami egy olyan függvényre mutat ami egészet ad vissza és két egészet kér be.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozasCaesar/tm.c?fbclid=IwAR1E0hLaxtbHsS3uRASStdKaeBQPqZY6yht_srjKI4RJzSG-UVQRxAL_vFfA

Kód:

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

int main()
{
    int nr=5;
    double **tm;

    if ((tm = (double **)malloc(nr*sizeof(double)))==NULL)
    {
        return -1;
    }

    for(int i=0; i<nr; i++)
    {
        if((tm[i]=(double *) malloc ((i+1) * sizeof (double)))==NULL)
        {
            return -1;
        }
    }

    for(int i=0; i<nr; i++)
        for(int j=0; j<i+1; j++)
```

```
tm[i][j]=i*(i+1)/2+j;

for(int i=0; i<nr; i++)
{
    for(int j=0; j<i+1; j++)
        printf("%f,", tm[i][j]);
    printf("\n");
}
tm[3][0]=42.0;
(*tm+3)[1]=43.0;
*tm[3]+2)=44.0;
*(*tm+3)+3)=45.0;

for(int i=0; i<nr; i++) //megint kiiratjuk
{
    for(int j=0; j<i+1; j++)
        printf("%f,", tm[i][j]);
    printf("\n");
}

for(int i=0; i<nr; i++)
    free(tm[i]);

free(tm);

return 0;
}
```

A fent látható kód egy alsó háromszög mátrixot fog kiíratni. Az nr változóban adhatjuk meg, hogy hány soros legyen a mátrix, jelen esetünkben 5 lesz. Ezután a **tm el egy pointernek foglalunk le helyet. Majd a malloc függvény segítségével annyiszor foglalunk le double-nyi méretet ahány sorunk van. A malloc alapesetben egy pointert ad vissza, de mi típuskényszerítéssel double **-ot kapunk vissza. Ha a malloc valami hiba során nem tudja lefoglalni a megfelelő mennyiségű memóriaterületet akkor egy NULL pointert fog visszaadni, ezáltal az if igaz lesz és kilép a programunk, ha nem történik ilyen akkor tovább lép. Ezután indítunk egy for ciklust amely 0-tól, jelen esetben négyig fog futni. Ez a for ciklus fogja lefoglalni minden sornak a megfelelő méretet. Az első sornak 1-et a másodiknak 2-öt és így tovább. Itt megint csak a malloc függvényt használjuk. Ezután még két for ciklus segítségével feltöljük a mátrixot. Majd a végén kiíratjuk a mátrixot.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/exor/>

Kód:

```
#include<unistd.h>
#include<string.h>

#define MAX_KULCS 100
#define BUFFER_MERET 256

int main (int argc, char **argv) // argc - stringek száma, ami az argv-re ↪
    mutat, az **argv pedig tárolni fogja a parancssori argumentumokat
{

    char kulcs[MAX_KULCS];
    char buffer[BUFFER_MERET];

    int kulcs_index=0;
    int olvasott_bajtok=0;

    int kulcs_meret=strlen (argv[1]);
    strncpy(kulcs,argv[1],MAX_KULCS);

    while((olvasott_bajtok=read(0, (void *) buffer, BUFFER_MERET)))
    {
        for(int i=0; i<olvasott_bajtok; i++)
        {

            buffer[i]=buffer[i]^kulcs[kulcs_index];
            kulcs_index=(kulcs_index+1)%kulcs_meret;
        }

        write(1, buffer, olvasott_bajtok);
    }
}
```

A fenti kód a kizáró vagy(EXOR) segítségével fogja titkosítani a megadott tiszta szövengüket.A program fordítása után, a futtatási parancs után megadjuk a kulcsunkat(ez alapján fogja titkosítani a szöveget) majd "" segítségével átadjuk a filet amit titkosítani szeretnénk és ">" segítségével kiíratjuk egy másik fileba a titkosított szöveget.A kód elején a #define segítségével MAX_KULCS és BUFFER_MÉRET néven megadunk két konstanst.A main elején létrehozzuk a szükséges változókat. A kulcs_meretet egyenlővé tesszük az strlen függvény segítségével az argv[1] méretével. Az argv tömb tárolja a parancssori argumentumokat, az első tagja igazából a 2. tagja lesz mert 0-tól indul az indexelés, tehát ez a kulcsunk mérete lesz.A strncpy függvény a kulcs változóba másolja át a megadott kulcsunkat.Ezután egy while ciklus következik, ez addig fog futni amíg a fájlunkból adatot tud olvasni,közben eltároljuk az olvasott bájtok számát. Ezekután indítunk egy for ciklust ami az olvasott bájtok számáig fog futni.Ebben a ciklusban zajlik maga titkosítás.A bufferben lévő bájtokat össze exorálja a kulcs bájtjaival és így egy másik karaktert kapunk.A végén pedig kiíratjuk a titkosított szövegünket.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Forrás:https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html?fbclid=IwAR0z9V_UxdHR3gu

```
public class ExorTitkosito {

    public ExorTitkosito(String kulcsSzoveg,
                          java.io.InputStream bejovoCsatorna,
                          java.io.OutputStream kimenoCsatorna)
                          throws java.io.IOException {

        byte [] kulcs = kulcsSzoveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBajtok = 0;

        while((olvasottBajtok =
               bejovoCsatorna.read(buffer)) != -1) {

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

                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;
            }
            kimenoCsatorna.write(buffer, 0, olvasottBajtok);
        }
    }

    public static void main(String[] args) {

        try {
            new ExorTitkosito(args[0], System.in, System.out);

        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

Ez a program az eddigiekkel ellentétben java nyelven íródott. Java nyelvű kódot a javac parancssal tudunk fordítani. A java egy erősen objektumorientált magasszintű programozási nyelv. Amint láthatjuk sokkal rövidebb és könnyebben olvashatóbb mint C-s társa. Javaban minden class-nak tekinthető, még a main függvény is. Vannak public és private osztályok. A public osztályokat a programon belül bármi eltud érni, a private classokat pedig csak az adott class tudja. Az exortitkosito class fogja a titkosítást végezni. Hárrom argumentuma van, a kulcs, és a kimenő és bemenő csatorna. Ha kevesebbet kap akkor a throw segítségével hibát fog dobni. Ezután beolvassa a bajtöket és össze exorozza a kulccsal, majd ez kiírja. A mainben lévő try és catch függvényel hibákat kaphatunk el.

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/exor/>

Kód:

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE

#include<stdio.h>
#include<unistd.h>
#include<string.h>

int tiszta_lehet(const char titkos[], int titkos_meret) //tiszta szöveg ←
    lehet, függvény nézi, kap egy állandó karakter tömböt, meg egy méretet
{
    //tiszta szöveg valószínű, hogy tartalmazza a gyakori magyar szavakat
    return strcasestr(titkos, "hogy") && strcasestr(titkos, "nem") && ←
        strcasestr(titkos, "az") && strcasestr(titkos, "ha"); //megkeresi hogy ←
        van e benne hogy, nem, az, ha
}

void exor(const char kulcs[], int kulcs_meret, char titkos[], int ←
    titkos_meret)
//exor eljárás, kap egy kulcs tömböt, egy kulcs méretet, a titkos tömböt és ←
    a titkos méretét
{
    int kulcs_index=0; //deklaráció

    for(int i=0; i<titkos_meret; ++i) //a ciklus a titkos méretig fut
    {
        titkos[i]=titkos[i]^kulcs[kulcs_index]; //exorral nézi a kulcsokkal a ←
            dolgokat, ya know this bad boi
        kulcs_index=(kulcs_index+1)%kulcs_meret;
    }
}

int exor_tores(const char kulcs[], int kulcs_meret, char titkos[], int ←
    titkos_meret)
//exortores kulcs tömb, kulcs méret, titkos tömb, méret
```

```
{  
    exor(kulcs, kulcs_meret, titkos, titkos_meret); //exortörés go for it  
  
    return tiszta_lehet(titkos, titkos_meret); //tiszta lehet megnézi  
}  
  
int main(void)  
{  
//deklarációk  
    char kulcs[KULCS_MERET];  
    char titkos[MAX_TITKOS];  
    char *p=titkos;  
    int olvasott_bajtok;  
  
//titkos fajt pufferelt berantása //HÜLP  
while((olvasott_bajtok=  
    read(0, (void *) p,  
        (p-titkos+OLVASAS_BUFFER<  
        MAX_TITKOS) ? OLVASAS_BUFFER:titkos+MAX_TITKOS-p)))  
    p+=olvasott_bajtok;  
  
//maradek hely nullazása a titkos bufferben  
for(int i=0; i<MAX_TITKOS-(p-titkos);++i)  
    titkos[p-titkos+i]='\0';  
  
//osszes kulcs eloallitasa  
for(int ii='0';ii<='9';++ii)  
    for(int ji='0';ji<='9';++ji)  
        for(int ki='0';ki<='9';++ki)  
            for(int li='0';li<='9';++li)  
                for(int mi='0';mi<='9';++mi)  
                    for(int ni='0';ni<='9';++ni)  
                        for(int oi='0';oi<='9';++oi)  
                            for(int pi='0';pi<='9';++pi)  
    {  
        kulcs[0]=ii;  
        kulcs[1]=ji;  
        kulcs[2]=ki;  
        kulcs[3]=li;  
        kulcs[4]=mi;  
        kulcs[5]=ni;  
        kulcs[6]=oi;  
        kulcs[7]=pi;  
  
        if(exor_tores(kulcs,KULCS_MERET,titkos,p-titkos))  
            printf("Kulcs: [%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",ii,ji,ki,li  
                  ,mi,ni,oi,pi,titkos);  
        //ujra EXOR-ozunk, így nem kell egy második buffer  
        exor(kulcs,KULCS_MERET,titkos,p-titkos);
```

```
    }
    return 0;
}
```

A kód elején define segítségével vannak deklarálva konstansok, ezekre a későbbiekben könnyebb lesz hivatkozni. Ezután a main függvény előtt vannak deklarálva különböző eljárások és függvények, amiket majd a programunk során használni fogunk. A tiszta_lehet függvény az első. Ez paraméterként kap egy karakter tömböt és egy méretet, majd ebben a tömbben a 'hogy', 'nem', 'az' és 'ha' szavakat keresi és ezt adja vissza is. Alatta az exor nevezetű eljárás található, azért eljárás és nem függvény mert nincs visszatérítési értéke. Ez az eljárás paraméterként kap egy kulcs tömböt, kulcs méretet illetve a titkos tömböt és titkos méretet. Ezután egy for ciklus segítségével a titkos szöveget bájtonként össze exorálja a kulccsal. Ez alatt található az exor_tores függvény amely meghívja az exor eljárást. Ezután a tiszta_lehet függvény és ennek fogja visszatéríteni az értékét. Ezután következik a main függvény. A deklarálások után egy while ciklussal beolvassuk a bufferbe a titkos szöveget, ezután egy for ciklussal a bufferben lévő maradék helyet nullákkal töltjük fel. Ezután nyolc egybeágazott for ciklussal melyek mindegyike 0-tól 9-ig fut, előállítjuk az összes lehetséges kulcsot. Ez a program maximum 8 számjegyű kulcsot tud feltörni. Ezután meghívjuk az exor_tores függvényt és ha igazat ad vissza kiíratjuk a kulcsot és a tiszta szöveget.

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

A következő kódokkal neurális hálókat fogunk betanítani az OR, AND és EXOR műveletekre. Kezdjük az OR-al:

```
library(neuralnet)

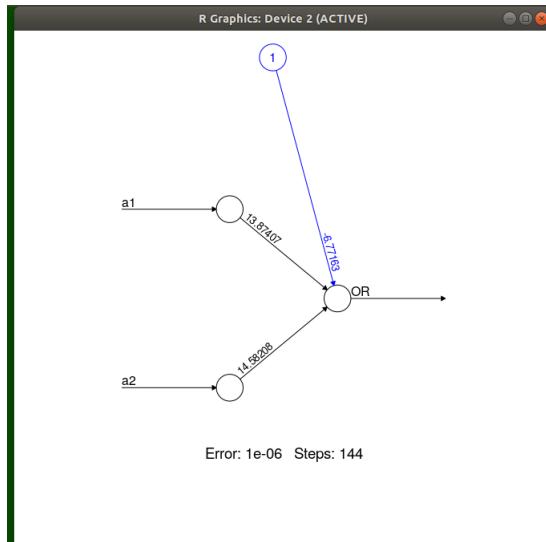
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR      <- c(0,1,1,1)

or.data <- data.frame(a1, a2, OR)

nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE, ←
                  stepmax = 1e+07, threshold = 0.000001)

plot(nn.or)

compute(nn.or, or.data[,1:2])
```



Először is szükségünk lesz a neuralnet package-re, ezt betölthjük. Majd meghatározunk 3 vektort. Itt megadjuk hogyha a1 0 és a2 is 0 akkor az OR is 0 lesz..és így tovább. Ezután ezekből adatot hozunk létre, majd feltölthjük a neurális hálót. Ezután a háló megtanulja és pontosan kiszámolja az OR értékeit. A megadott értékekből megállapítja a súlyokat. És a program végén a compute al számolja ki a végleges számokat.

```

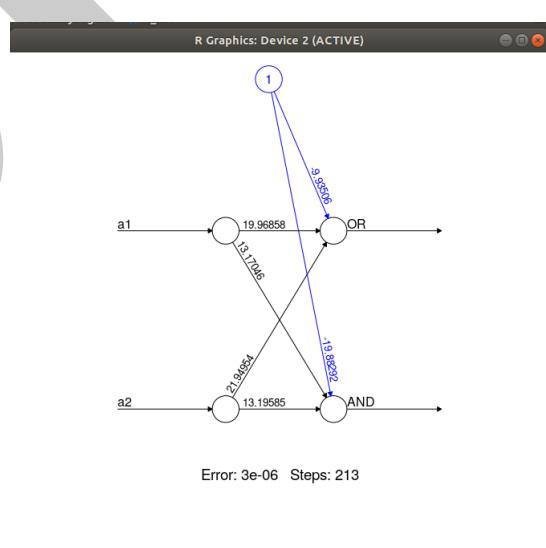
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR      <- c(0,1,1,1)
AND     <- c(0,0,0,1)

orand.data <- data.frame(a1, a2, OR, AND)

nn.orand <- neuralnet(OR+AND~a1+a2, orand.data, hidden=0, linear.output= FALSE,
                      stepmax = 1e+07, threshold = 0.000001)

plot(nn.orand)

compute(nn.orand, orand.data[,1:2])
  
```



Ez a kód pedig az AND logika utasítást fogja megtanulni. Lényegében ugyanaz mint az előző, csak az

OR mellé megadtuk az AND művelet eredményeit is, és ez alapján adja ki az eredményeket. Láthatjuk azt is, hogy nyugodtan ábrázolhatunk több kimenetet is egy ábrán.

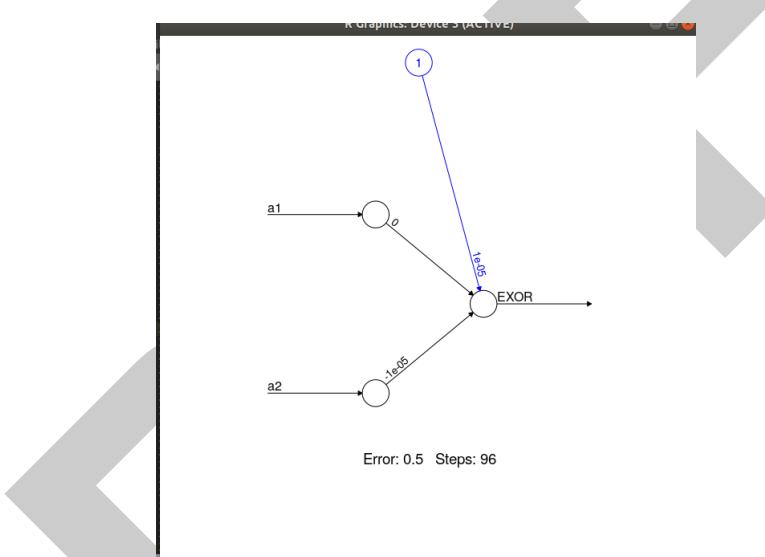
```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR   <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=0, linear.output=FALSE, ←
    stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

compute(nn.exor, exor.data[,1:2])
```



Ebben a kódban az EXORT szeretnénk megtanítani a neuronnak. EXOR-nál 0,1,1,0 kellene kapnunk azonban, láthatjuk, hogy mindenhol 0,5-öt kaptunk, azaz a program nem tudta megtanulni.

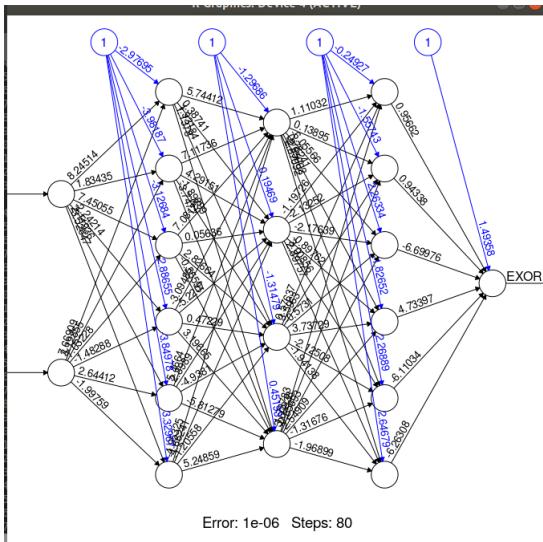
```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR   <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ←
    output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

compute(nn.exor, exor.data[,1:2])
```



Amint látjuk e fenti program már jól működik. Ugyanis a tanításnál megadtunk pár rejtett hálózatot és így sokkal hatékonyabban és pontosabban tud tanulni.

4.6. Hiba-visszaterjesztéses perceptron

C++

Kód:

```
#include <iostream>
#include "mlp.hpp"
#include "png++/png.hpp"
//g++ mlp.hpp main.cpp -o perc -lpng -std=c++11

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

    Perceptron* p = new Perceptron(3, size, 256, 1);

    double* image = new double[size];

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

    double value = (*p) (image);

    std::cout << value << std::endl;
```

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

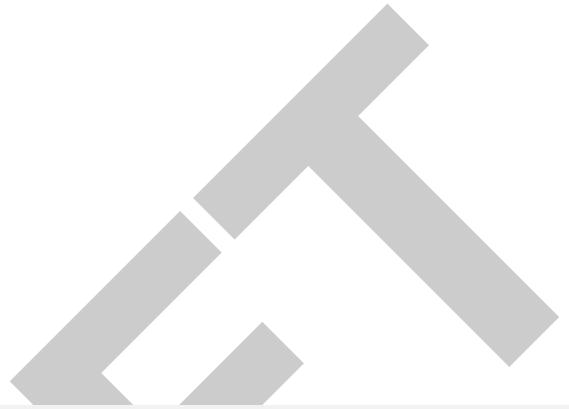
Megoldás forrása: <https://youtu.be/XpBnR31BRJY>

A program futásához szükségünk lesz a libpng csomagra.Ezzel a programmal a mandel.png RGB kódjának a piros részét adjuk át,és a program egy három rétegű hálót alkot belőle.A futáshoz szükségünk van az mlp.hpp fájlra is,ugyanis ez tartalmazza a perceptron classt.A main-ben először lefoglaljunk a szükséges helyeket.Ezután a megadott kép méretét eltároljuk egy változóban.Alatta pedig létrehozunk egy perceptron típusú változót, amiben megadjuk,hogy hány rétegű legyen a háló.Majd két egybe ágyazott for ciklussal betöljtük a kép piros részét és ráállítunk egy pontert.Ezután kiíratjuk a megfelelő eredményt, majd felszabadítjuk a memóriaterületet.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz



```
// Copyright - no copyright

#include <png++/png.hpp>

#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35

void GeneratePNG(int tomb[N][M]) {
    png::image<png::rgb_pixel> image(N, M);

    for (int x = 0; x < N; ++x) {
        for (int y = 0; y < M; ++y) {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y]);
        }
    }
    image.write("kimenet.png");
}

struct Komplex {
    double re, im;
};

int main() {
    int tomb[N][M];

    int i, j, k;
    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;
```

```
struct Komplex C, Z, Zuj;

int iteracio;

for (i = 0; i < M; ++i) {
    for (j = 0; j < N; ++j) {
        C.re = MINX + j * dx;
        C.im = MAXY - i * dy;

        Z.re = 0;
        Z.im = 0;
        iteracio = 0;

        while (Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < 255) {
            Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
            Zuj.im = 2 * Z.re * Z.im + C.im;
            Z.re = Zuj.re;
            Z.im = Zuj.im;
        }
        tomb[i][j] = 256 - iteracio;
    }
}
GeneratePNG(tomb);

return 0;
}
```

A fenti kód a következő png képet fogja generálni:



Ez a mandelbrot halmaz síkbeli ábrázolása. A mandelbrot halmaz komplex számokból áll. Ezekre a számokra igaz az, hogy $X(n+1)=Xn^2+c$. És ez nem tart a végtelenbe, azaz korlátos. A forráskód fordításához szükség van egy -lpng kapcsolóra, ugyanis a kód tartalmazza a png++ header file-t, ez szükség a png generáláshoz. A kódunk elején deklarálunk pár konstans számot, majd a png generáló eljárást is. Ez paraméterül kap egy NxM-es két dimenziós tömböt. Egy 500x500 -as képet fog létrehozni, két for ciklus segítségével és a tömbben lévő értékek szerint vagy fehér lesz az adott pixel vagy fekete. Az eljárás végén write-al hozzuk létre a png-t. Ezután létrehozzuk a Komplex struktúrát amiben egy valós és egy komplex szám lesz. A main elején deklaráljuk a szükséges változókat, és létrehozunk 3 Komplex osztályú változót is. Ezután a megfelelő képletet használva feltöljük a mátrixot a megfelelő értékekkel és átadjuk a GeneratePNG eljárásnak a

mátrixot.

5.2. A Mandelbrot halmaz a std::complex osztályal

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double a = -1.9;
    double b = 0.7;
    double c = -1.3;
    double d = 1.3;

    if ( argc == 9 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        a = atof ( argv[5] );
        b = atof ( argv[6] );
        c = atof ( argv[7] );
        d = atof ( argv[8] );
    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ←
                     " << std::endl;
        return -1;
    }

    png::image< png::rgb_pixel > kep ( szelesseg, magassag );

    double dx = ( b - a ) / szelesseg;
    double dy = ( d - c ) / magassag;
    double reC, imC, reZ, imZ;
    int iteracio = 0;

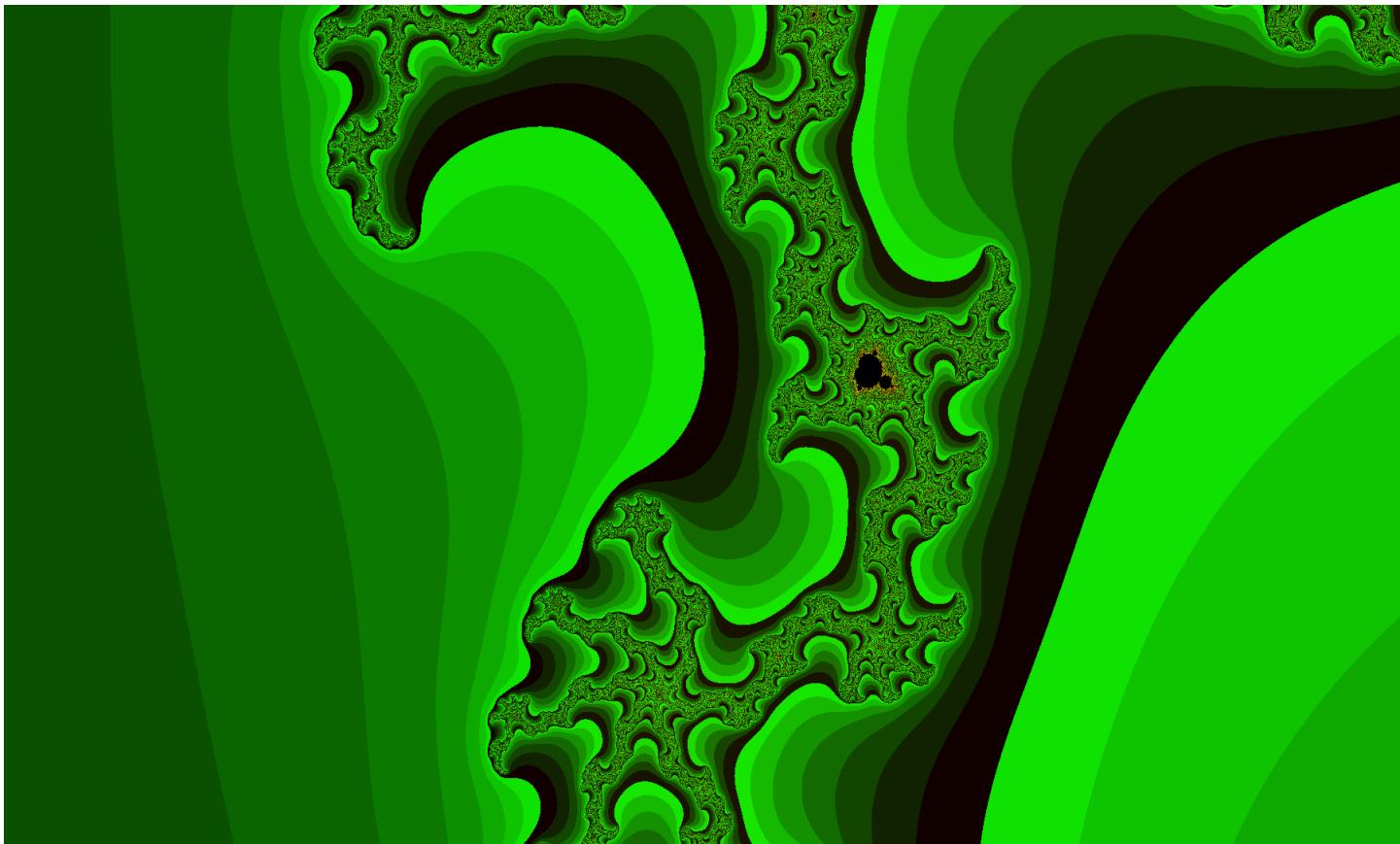
    std::cout << "Szamitas\n";

    for ( int j = 0; j < magassag; ++j )


```

```
{  
  
    for ( int k = 0; k < szelesseg; ++k )  
    {  
  
        reC = a + k * dx;  
        imC = d - j * dy;  
        std::complex<double> c ( reC, imC );  
  
        std::complex<double> z_n ( 0, 0 );  
        iteracio = 0;  
  
        while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )  
        {  
            z_n = z_n * z_n + c;  
  
            ++iteracio;  
        }  
  
        kep.set_pixel ( k, j,  
                        png::rgb_pixel ( iteracio%255, (iteracio*iteracio ←  
                            )%255, 0 ) );  
    }  
  
    int szazalek = ( double ) j / ( double ) magassag * 100.0;  
    std::cout << "\r" << szazalek << "%" << std::flush;  
}  
  
kep.write ( argv[1] );  
std::cout << "\r" << argv[1] << " mentve." << std::endl;  
}
```





Ez a kód nem sokban különbözik az előzőtől. Bekerült a complex header file. Ezáltal nem kell struktúrát használnunk, hogy kezeljük a komplex számokat, hanem ez a header file fogja kivitelezni azt. A kód elején deklaráljuk a szükséges változókat. Futtatáskor meg kell adnunk a kép magasságát és szélességét, azt hogy milyen néven hozza létre a képet, és az n a b c d értékeit. Ezt követően a program beállítja a megfelelő értékekre a változókat. Ha viszont rosszul adtuk meg a dolgokat (pl kevés számot adtunk meg) akkor kiíratja a használatot. Ezután a szükséges méretet lefoglaljuk a majd létrejövő képünknek. Maga a számolás a while ciklusban történik. A program futása közben folyamatosan tájékoztat arról, hogy hány százaléknál jár a generálás és arról is ha sikeresen lementette a képet.

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbgrZy76E>

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

```
#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 szelesseg magassag n a b c ←
                  d reC imC R" << std::endl;
    return -1;
}

png::image<png::rgb_pixel> kep ( szelesseg, magassag );

double dx = ( xmax - xmin ) / szelesseg;
double dy = ( ymax - ymin ) / magassag;

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

std::cout << "Szamitas\n";

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

    for ( int x = 0; x < szelesseg; ++x )

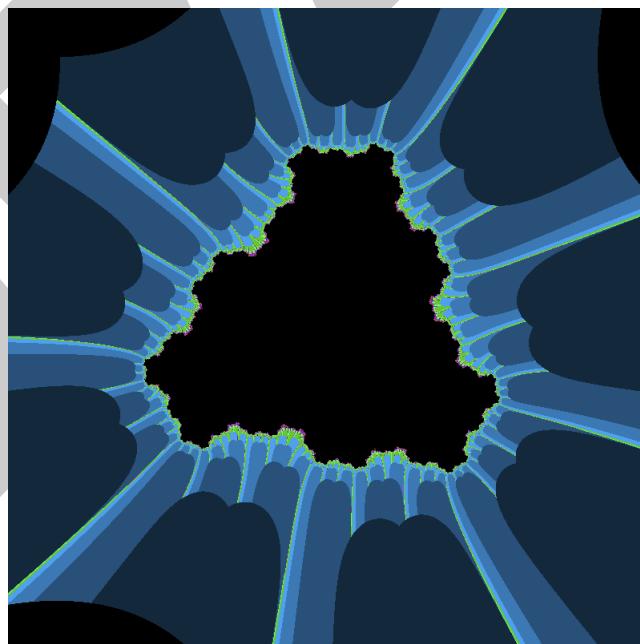
        double reZ = xmin + x * dx;
        double imZ = ymax - y * dy;
        std::complex<double> z_n ( reZ, imZ );
```

```
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 ) magassag * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
}

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



Ez a program lényegében Julia halmazokat fog létrehozni. Julia halmazokból végtelen mennyiségű van. Ennek a programnak az alapja az előző mandelbrotos program. A mandelbrot halmaz tartalmazza az összes Julia halmazt. A mandelbrotos programmal ellentétben itt a c nem változó lesz, hanem állandó, és a konzolról kérjük be. A kód elején megint csak létrehozzuk a megfelelő változókat. Majd bekérjük azokat és ha megfelelők akkor a megfelelő változóhoz rendeljük azokat. Lefoglaljuk a helyet a képunknek. For ciklu-

sukkal végig megyünk az összes pixelen és a megfelelő egyenlet segítségével kiszámoljuk az értékeket. Ez a program is jelzi, hogy hány százaléknál jár és hogy lementette-e már a képet.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: https://progpater.blog.hu/2011/03/27/a_parhuzamossag_gyonyorkodtet?fbclid=IwAR0m3eiY

```
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>

#include <sys/times.h>
#include <iostream>

#define MERET 600
#define ITER_HAT 32000

_device_ int
mandel (int k, int j)
{

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

float dx = (b - a) / szelesseg;
float dy = (d - c) / magassag;
float reC, imC, reZ, imZ, ujreZ, ujimZ;

int iteracio = 0;

reC = a + k * dx;
imC = d - j * dy;

reZ = 0.0;
imZ = 0.0;
iteracio = 0;

while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;
```

```
++iteracio;

}

return iteracio;
}

/*
_global_ void
mandelkernel (int *kepadat)
{

int j = blockIdx.x;
int k = blockIdx.y;

kepadat[j + k * MERET] = mandel (j, k);

}
*/



_global_ void
mandelkernel (int *kepadat)
{

int tj = threadIdx.x;
int tk = threadIdx.y;

int j = blockIdx.x * 10 + tj;
int k = blockIdx.y * 10 + tk;

kepadat[j + k * MERET] = mandel (j, k);

}

void
cudamandel (int kepadat[MERET] [MERET])
{

int *device_kepadat;
cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

// dim3 grid (MERET, MERET);
// mandelkernel <<< grid, 1 >>> (device_kepadat);

dim3 grid (MERET / 10, MERET / 10);
dim3 tgrid (10, 10);
mandelkernel <<< grid, tgrid >>> (device_kepadat);

cudaMemcpy (kepadat, device_kepadat,
```

```
        MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
cudaFree (device_kepadat);

}

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

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

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

int kepadat [MERET] [MERET];

cudamandel (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));
    }
}
kep.write (argv[1]);

std::cout << argv[1] << " mentve" << std::endl;

times (&tmsbuf2);
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
      + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

delta = clock () - delta;
```

```
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;  
}
```

Ebben a feladatban az első mandelbrotos feladatot fogjuk megoldani a CUDA segítségével.A CUDA segítségével el tudjuk azt érni, hogy a számításokat ne csak a processzerünk végezze,hanem besegítsen a videókártyánk is. A CUDA működéséhez NVIDIA videókártyára lesz szükségünk,ugyanis a CUDA-t az NVIDIA fejlesztette ki.Így a programunk sokkal gyorsabban fogja legenerálni a 600x600-as képünket.

A program fordításához szükségünk lesz az nvidia-cuda-toolkit-re.Ezután a telepítés után nvcc parancsal tudjuk majd a fordítást végrehajtani.A kód elején a DEFINE segítségével meghatározunk két állandót, az egyik a kép mérete lesz a másik pedig az iterációs határ.A mandel függvényben végezzük el a szükséges számításokat melyek majd létrehozzák a mandelbrot halmazt.Itt nem használjuk a complex header file-t, ami magától tudná kezelni a komplex számokat.A függvény előtt észrevehetjük,hogy nem simán a visszatérítési típus van megadva,hanem már megadtuk a device szócskát,amivel jelezzük,hogy a GPU befog szállni a számolásba.Mind a device illetve a global szócska arra lesz jó,hogy megadjuk,hogy ezekhez a számításokhoz szálljon be a GPU is. Ezt majd a fordítóprogram intézi el nekünk.A mandelkernel függvény fogja kiszámolni a z értékeket.A cudamandel függvényben lefoglalunk egy 600x600-as tömbnek memóriát. Majd a grid-ben megadjuk,hogy hány blokkot szertnénk létrehozni, a tgridben pedig,hogy egy blokkhoz hány szál tartozzon. Ezután ezeket átadjuk a mandel függvénynek.Ha ez megvan, felszabadítjuk a területet.A main függvény nem sokban különbözik az előző feladathoz képest,csupán annyiban,hogy futási időt is mérünk.

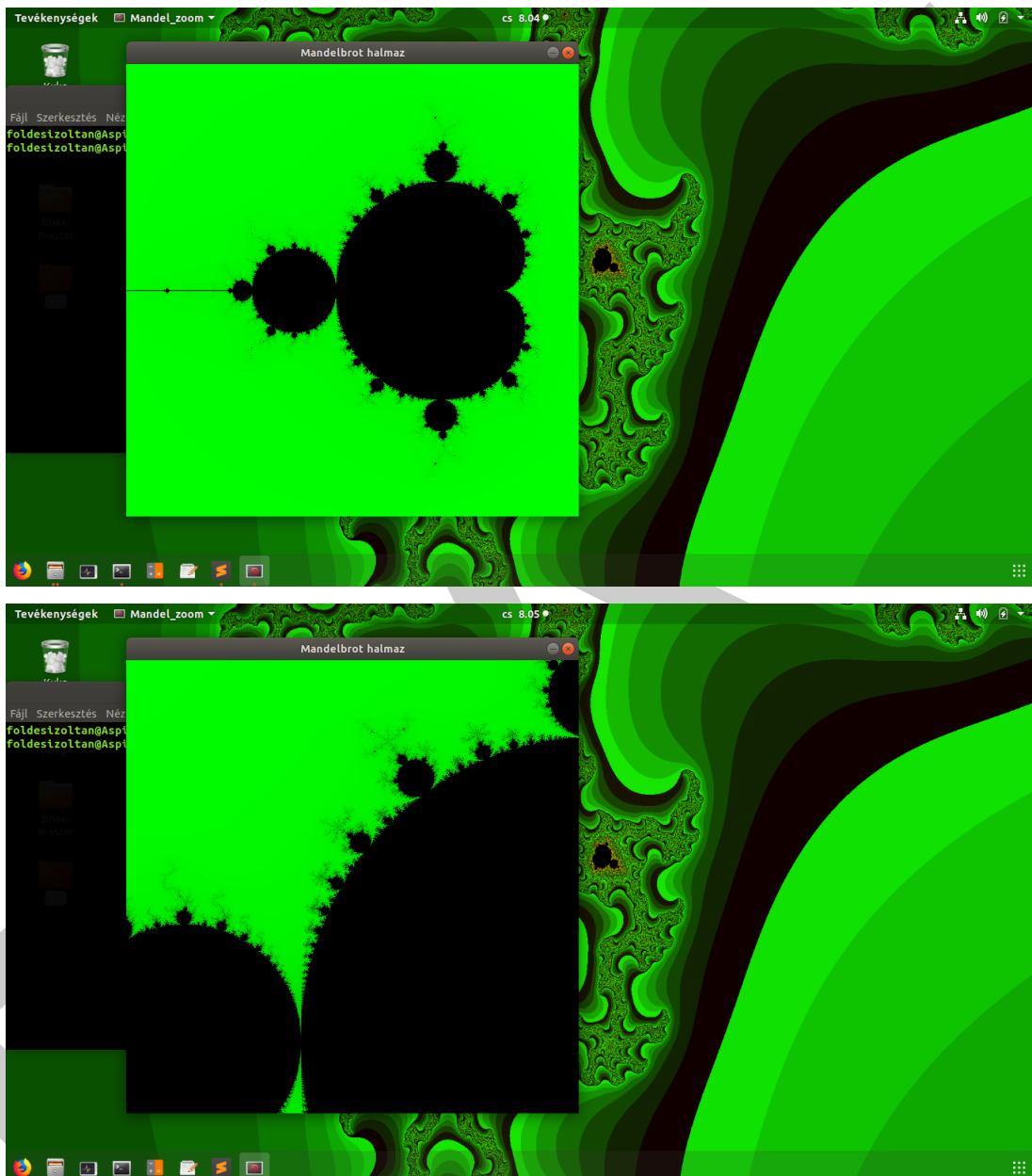
Ez a technológia sokszor használatos képfeldolgozásnál illetve videórenderelésnél.

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!

```
#include <QApplication>  
#include "frakablak.h"  
  
int main(int argc, char *argv[])  
{  
    QApplication a(argc, argv);  
  
    FrakAblak w1;  
    w1.show();  
  
    /*  
     FrakAblak w1,  
     w2(-.08292191725019529, -.082921917244591272,  
     -.9662079988595939, -.9662079988551173, 600, 3000),  
     w3(-.08292191724880625, -.0829219172470933,  
     -.9662079988581493, -.9662079988563615, 600, 4000),  
     w4(.14388310361318304, .14388310362702217,  
     .6523089200729396, .6523089200854384, 600, 38655);  
    w1.show();  
    w2.show();
```

```
w3.show();  
w4.show();  
*/  
return a.exec();  
}
```



A fent látható kód működéséhez szükséges telepíteni a libqt4-dev-et. Valamint szükséges összesen 5 fájl. A fájlokat az előbb említett csomaggal fogjuk összefűzni. Először qmake -project parancssal létfogunk hozni egy .pro kiterjesztésű fájlt, ennek a fájlnak az utolsó sorába be kell írnunk a QT += widgets mondatot. Ezek után qmake *.pro parancssal létrejön egy makefile. Majd a make parancssal létrehozzuk a futtatható fájlt. Ezután futtatva a kódot megjelenik a Mandelbrot halmaz, amiben kijelölve egy területet bele tudunk nagyítani szinte a végtelenségig. Ez látható az első és a második képen.

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

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

```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {  
    private int x, y;  
    private int mx, my;  
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,  
        int szélesség, int iterációsHatár) {  
        super(a, b, c, d, szélesség, iterációsHatár);  
        setTitle("A Mandelbrot halmaz nagyításai");  
        addMouseListener(new java.awt.event.MouseAdapter() {  
            public void mousePressed(java.awt.event.MouseEvent m) {  
                x = m.getX();  
                y = m.getY();  
                mx = 0;  
                my = 0;  
                repaint();  
            }  
            public void mouseReleased(java.awt.event.MouseEvent m) {  
                double dx = (MandelbrotHalmazNagyító.this.b  
                    - MandelbrotHalmazNagyító.this.a)  
                    /MandelbrotHalmazNagyító.this.szélesség;  
                double dy = (MandelbrotHalmazNagyító.this.d  
                    - MandelbrotHalmazNagyító.this.c)  
                    /MandelbrotHalmazNagyító.this.magasság;  
                new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+  
                    x*dx,  
                    MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,  
                    MandelbrotHalmazNagyító.this.d-y*dy-my*dy,  
                    MandelbrotHalmazNagyító.this.d-y*dy,  
                    600,  
                    MandelbrotHalmazNagyító.this.iterációsHatár);  
            }  
        });  
        addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {  
            public void mouseDragged(java.awt.event.MouseEvent m) {  
                mx = m.getX() - x;  
                my = m.getY() - y;  
                repaint();  
            }  
        });  
    }  
    public void pillanatfelvétel() {  
        java.awt.image.BufferedImage mentKép =  
            new java.awt.image.BufferedImage(szélesség, magasság,  
                java.awt.image.BufferedImage.TYPE_INT_RGB);  
        java.awt.Graphics g = mentKép.getGraphics();  
        g.drawImage(kép, 0, 0, this);  
    }  
}
```

```
g.setColor(java.awt.Color.BLUE);
g.drawString("a=" + a, 10, 15);
g.drawString("b=" + b, 10, 30);
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iterációsHatár, 10, 75);
if(számításFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyítás_");
sb.append(++pillanatfelvételSzámláló);
sb.append("_");
sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");
try {
    javax.imageio.ImageIO.write(mentKép, "png",
        new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
    e.printStackTrace();
}
}
public void paint(java.awt.Graphics g) {
    g.drawImage(kép, 0, 0, this);
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
}
public static void main(String[] args) {
    new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
```

Ebben a feladatban az előző Qt-s projektetírtuk meg java-ban.A lényege annyi,hogy létrehozza a mandelbrot halmazt és bele tudunk nagyítani,illetve a nagyított képet az n betű megnyomásával élesíteni tudjuk.Ebben a programban már be van importálva egy másik program a MandelBrothalmaz.java, ezt az extend szócskával

érjük el, ez olyan mintha C-ben vagy C++-ban include segítségével hozzácsatoltuk volna. A program figyeli amikor megnyomjuk az egeret, és akkor lekérdezi a koordinátákat, majd azt is figyeli mikor engedjük fel, és a két koordináta különbségéből kiszámolja, hogy hova szerettünk volna belenagyítani. A kódnak van egy kis hibája, ugyanis minden nagyításkor új ablakot nyit. Az új koordinátánál, számol szélességet és magasságot és újra rajzolja a képet.



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.

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

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*u1-1;
            w=v1*v1+v2*v2;
        }
        while (w>1.0);
        nincsTarolt=false;
    }
    else
    {
        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*u1-1;
            w=v1*v1+v2*v2;
        }
        while (w>1.0);
        nincsTarolt=true;
    }
    return w;
}
```

```
        }
    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;
}
```

A fenti kóddal random számokat tudunk generálni. Ezt egy osztályval valósítottuk meg. Ennek az osztálynak van egy public illetve egy private része. A public részt a programon belül bárhonnan eltudjuk érni, a private rész viszont csak az osztályon belül lesz elérhető. A class elején van megadva a konstruktor és a destruktör. A konstruktor akkor fut le ha egy új példányt szeretnénk létrehozni ilyen típusú osztályként. A destruktör pedig törlésnél fog lefutni, ám ez az esetünkben nem lesz fontos. A kovetkező függvényel fogjuk a random számokat generálni, azonban ennek a matematikai háttérével most nem fogunk foglalkozni. A konstruktor meghívásakor a nincsTarolt változó igaz lesz illetve az strand függvény meghívódik. A main függvényben létrehoztunk egy pg nevű változót amely Polargen típusú lesz, tehát itt meghívódik majd a konstruktor. Valamint egy for ciklussal 10 random számot generálunk le.

A java kód a következőképpen néz ki:

```
public class PolarGenerator
{
    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
        if(nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do{
                u1 = Math.random();
                u2 = Math.random();
```

```
    v1 = 2 * u1 -1;
    v2 = 2 * u2 -1;
    w = v1*v1 + v2*v2;
} while (w>1);

double r = Math.sqrt((-2 * Math.log(w) / w));
tarolt = r * v2;
nincsTarolt = !nincsTarolt;
return r * v1;
}

else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}

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

Láthatólag sokkal rövidebb lett a kódunk mint c++-ban. Annyiban különbözik, hogy javaban minden egy nagy class része, még a main függvény is. Ezenkívül maga a generáló függvény teljesen ugyanaz, és maga a program is ugyanúgy működik.

6.2. LZW

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

https://progater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat?fbclid=IwAR0p5MQomtcQIdfTeZvPInhgRxu-CCsxGOx453MSrGk

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

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;
}

extern void kiir (BINFA_PTR elem);
extern void ratlag (BINFA_PTR elem);
extern void rszoras (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);

int
main (int argc, char **argv)
{
    char b;

    BINFA_PTR gyoker = uj_elem ();
    gyoker->ertek = '/';
    gyoker->bal nulla = gyoker->jobb_egy = NULL;
    BINFA_PTR fa = gyoker;

    while (read (0, (void *) &b, 1))
    {
        if (b == '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);  
  
extern int max_melyseg, atlagosszeg, melyseg, atlagdb;  
extern double szorasosszeg, atlag;  
  
  
printf ("melyseg=%d\n", max_melyseg-1);  
  
atlagosszeg = 0;  
melyseg = 0;  
atlagdb = 0;  
ratlag (gyoker);  
  
atlag = ((double)atlagosszeg) / atlagdb;  
  
  
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);  
}
```

```
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));

        }
    }
}
```

```
}

}

int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            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;
    }
}

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

A fent látható program a bemenetére kapott 0-ásokból és egyesekből egy bináris fát fog alkotni. A bináris fára igaz az, hogy minden gyermek csomópontjának maximum 0,1 vagy kettő gyermekje lehet. A kód fordításához szükségünk lesz a -lm kapcsolóra, ennek segítségével tudjuk majd a matematikai függvényeket használni. Kódunk elején deklaráljuk a binfa struktúrát, a typedef segítségével. Ez tartalmazni fog egy egész értéket, illetve két mutatót. Az egyik mutató a bal oldali gyermekre fog mutatni a másik pedig a jobb oldalira. Az alatta lévő függvényteljesítményével az új elemeknek fogjuk lefoglalni a megfelelő nagyságú memóriát, ezt a malloc segítségével tesszük meg. Ez a lefoglalt területre mutató mutató fog visszaadni, itt vizsgáljuk azt is ha NULL pontert ad vissza, ugyanis ekkor nem tudta lefoglalni a területet és ekkor egy errorr dobunk. Ezután négy függvényprototípust láthatunk. Ezután következik a main ahol először lefoglaljuk a gyökérnek a területet értékét pedig egy / jelre állítjuk. Valamint a bal és jobb mutatóját NULL pointerre állítjuk. Majd visszaállunk a gyökérre. Ezután elkezdjük beolvasni a számokat a bemenetről. Ha nullást kapunk a bementről

akkor először megnézzük,hogy van-e a fának baloldali gyermeké, ha nincs akkor létrehozunk egyet ,érteke nullás lesz a pointerek pedig NULL pointerek, és a gyökér baloldali pointerét ráállítjuk. Ha azonban már van baloldalni gyermeké akkor addig lépkedünk balra, ameddig nem lesz egy üres hely. Ha egyest kapunk a bementről akkor ugyanezeket a lépéseket tesszük meg csak a jobb oldalra.A main végén a kiir függvénytel fogjuk kiíratni a fánkat. Ez a függvény preorder bejárás szerint fogja kiíratni a fát, a fabejárásokról a következő feladatnál beszélünk részletesebben. A szabadit függvénytel rekurzívan fogjuk felszabadítani az összes lefoglalt területet.

6.3. Fabejárás

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

Tutorált: Kiss Máté

Az előző programunk tökéletes lesz a fabejárások bemutatására. Csak a kiir függvényt fogjuk változtatni. Egy fát három féleképpen tudunk bejárni.Az előző kiir függvényben inorder szerint jártuk be a fát. Az az először a bal oldai részfát jártuk be inorderesen majd a gyökeret és azután a jobb oldalit inorderesen. Most nézzük meg preorderesen a kiir függvényt.

```
void kiir(BINFA_PTR elem)
{
    if (elem !=NULL)
    {
        ++melyseg
        if (melyseg>max_melyseg)
            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
    }
}
```

Itt láthatjuk,hogy először a gyökeret járjuk be, majd a baloldali részfát preorderesen, majd utoljára a jobb oldali részfát preorderesen.Az utolsó bejárási mód a postorder bejárás.Nézzük a kiir függvényt :

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

```
{  
    ++melyseg  
    if (melyseg>max_melyseg)  
        max_melyseg = melyseg;  
    kiir(elem->bal_nulla);  
    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);  
    --melyseg  
}  
}
```

Itt először a baloldali részfát járjuk be postorderesen majd a jobb oldali részfát ugyancsak postorderesen, és majd a legvégén a gyökeret. Észrevehetjük, hogy az összes fabejárás rekurzívan történik.

6.4. Tag a gyökér

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

```
class LZWBinFa  
{  
public:  
    LZWBinFa (char b = '/') : betu (b), balNulla (NULL), jobbEgy (NULL) {}  
    ~LZWBinFa () {};  
    void operator<<(char b)  
    {  
        if (b == '0')  
        {  
            // van '0'-s gyermeke az aktuális csomópontnak?  
            if (!fa->nullasGyermek ()) // ha nincs, csinálunk  
            {  
                Csomopont *uj = new Csomopont ('0');  
                fa->ujNullasGyermek (uj);  
                fa = &gyoker;  
            }  
            else // ha van, arra lépünk  
            {  
                fa = fa->nullasGyermek ();  
            }  
        }  
        else  
        {
```

```
if (!fa->egyesGyermek ())
{
    Csomopont *uj = new Csomopont ('1');
    fa->ujEgyesGyermek (uj);
    fa = &gyoker;
}
else
{
    fa = fa->egyesGyermek ();
}
}

class Csomopont
{
public:
    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0) {};
    ~Csomopont () {};
    Csomopont *nullasGyermek () {
        return balNulla;
    }
    Csomopont *egyesGyermek ()
    {
        return jobbEgy;
    }
    void ujNullasGyermek (Csomopont * gy)
    {
        balNulla = gy;
    }
    void ujEgyesGyermek (Csomopont * gy)
    {
        jobbEgy = gy;
    }
private:
    friend class LZWBinFa;
    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &);
    Csomopont & operator=(const Csomopont &);
};

if (b == '0')
{
    if (!fa->nullasGyermek ())
    {
        Csomopont *uj = new Csomopont ('0');
        fa->ujNullasGyermek (uj);
        fa = &gyoker;
    }
}

int main ()
```

```
{  
    char b;  
    LZWBInFa binFa;  
    while (std::cin >> b)  
    {  
        binFa << b;  
    }  
    binFa.kiir ();  
    binFa.szabadit ();  
    return 0;  
}
```

Ez a program lényegében ugyanaz mint az előző faépítő algoritmus, csak ez c++-ban íródott. A struktúra helyett osztályt, az az class-t használunk. A kód elején megadjuk az LZWBInfa osztályunkat. Ebben beállítjuk a gyökeret illetve a gyermeket melyek eleinte NULL pointerek lesznek. Majd ezután nézzük meg, hogy egyes vagy nullás jön-e a bemenetről és attól függően, hogy melyik jön ugyanazt csináljuk mint az előző programunkban. Itt annyi a különbség, hogy egyből a fába kerül az adott elem. Ha új csomópontot kell létrehoznunk akkor a new szóval tudjuk azt megtenni. Ugyanis létrehoztunk még egy Csomopont nevezetű osztályt. A nullasGyermek és egyesGyermek függvények az adott gyermekre mutató pointert fognak visszatérési értékként adni. Az ujEgyesgyemek és ujNullasgyermek pedig paraméterként kap egy gyermeket és arra fog mutatót állítani. A main függvényünkben csak beolvassuk az adatokat a bemenetről és átadjuk az osztályoknak. Ezután kiiratjuk és a végén az összes helyet felszabadítjuk.

6.5. Mutató a gyökér

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

Az előző bináris fákban a gyökerelém, tagként szerepelt. Azonban most ezt átfogjuk írni egy mutatóra. Első lépésként megkeressük a protected részben a Csomopont gyokeret és mutatóvá írjuk át azt, ezt megtehetjük úgy, hogy egy csillagot teszünk elé.

```
protected:  
    Csomopont *gyoker;  
    int maxMelyseg;  
    double atlag, szoras;
```

Ezután ha fordítjuk a programot, nagyon sok hibát fog kiírni. Ezeket kijavíthatjuk, ha minden elem valamelyen gyermeknél a pontokat kicseréljük nyilakra. Valamint a gyokernel a referenciajeleket ki kell törölnünk. Ha ezután próbáljuk fordítani a programunkat, le fog fordulni, azonban az első futtatásnál szegmentálási faultot fog kidobni, ugyanis olyan memóriacímre szeretnénk hivatkozni amihez nincs jogunk. Szóval le kell foglalnunk a helyet a konstruktőrral, illetve a destruktőrral ki is kell majd azt törölnünk.

```
class LZWBInFa  
{
```

```
public:  
  
    LZWBinFa ()  
    {  
        gyoker= new Csomopont ('/');  
        fa = gyoker;  
    }  
    ~LZWBinFa ()  
    {  
        szabadit (gyoker->egyesGyermekek());  
        szabadit (gyoker->nullasGyermekek());  
        delete(gyoker);  
    }
```

Ezután a programunk ugyanúgy fog működni mint ahogy eddig is működött.

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

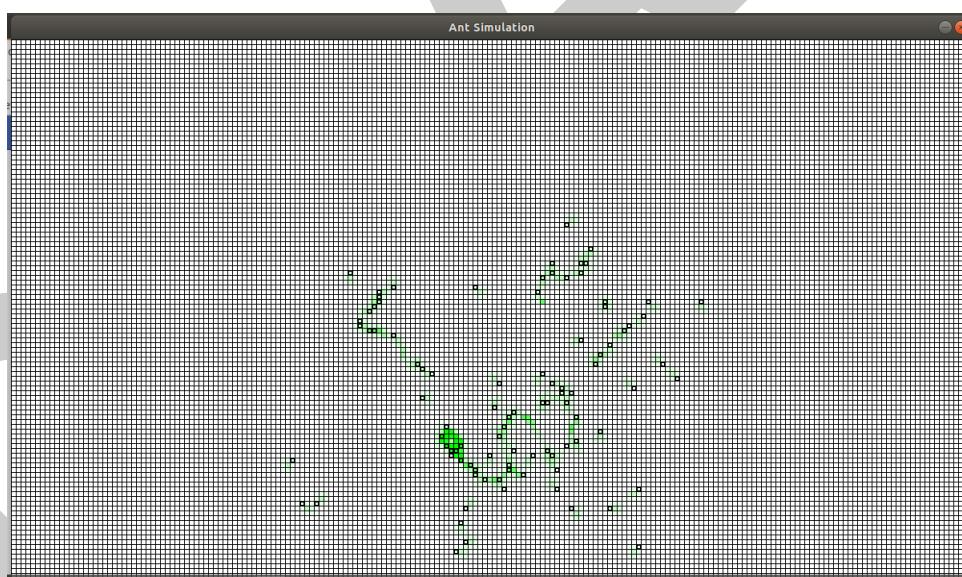
Helló, Conway!

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/nbatfai/bhax/tree/master/attention_raising/Myrmecologist



Ezt a feladatot a Qt segítségével oldjuk meg. Több fájlunk lesz, ezekből majd összeáll a programunk. A program egy hangyszimuláció lesz. Egy képernyőt felosztunk kis négyzetekre és minden kis négyzet ami ki lesz szinezve egy hangyának felel meg.

Az ant.h fájlból lesz megadva az Ant osztály, ezek lesznek a hangyák. minden hangyának lesz egy x és egy y koordinátája, illetve egy iránya, amit egy random számgenerálással fogunk megadni. Fontos megjegyezni, hogy ezek az adatok publikusak lesznek, azaz más függvények is elfogják érni ezeket.

```
#include "antwin.h"
```

```
#include <QDebug>

AntWin::AntWin ( int width, int height, int delay, int numAnts,
                int pheromone, int nbhPheromon, int evaporation, int ←
                cellDef,
                int min, int max, int cellAntMax, QWidget *parent ) : ←
                QMainWindow ( parent )

{
    setWindowTitle ( "Ant Simulation" );

    this->width = width;
    this->height = height;
    this->max = max;
    this->min = min;

    cellWidth = 6;
    cellHeight = 6;

    setFixedSize ( QSize ( width*cellWidth, height*cellHeight ) );

    grids = new int**[2];
    grids[0] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[0][i] = new int [width];
    }
    grids[1] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[1][i] = new int [width];
    }

    gridIdx = 0;
    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j ) {
            grid[i][j] = cellDef;
        }

    ants = new Ants();

    antThread = new AntThread ( ants, grids, width, height, delay, numAnts, ←
        pheromone,
                                nbhPheromon, evaporation, min, max, ←
                                cellAntMax);

    connect ( antThread, SIGNAL ( step ( int ) ),
              this, SLOT ( step ( int ) ) );

    antThread->start();
}
```

```
}

void AntWin::paintEvent ( QPaintEvent* )
{
    QPainter qpainter ( this );

    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i ) {
        for ( int j=0; j<width; ++j ) {

            double rel = 255.0/max;

            qpainter.fillRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight,
                                QColor ( 255 - grid[i][j]*rel,
                                          255,
                                          255 - grid[i][j]*rel ) );

            if ( grid[i][j] != min )
            {
                qpainter.setPen (
                    QPen (
                        QColor ( 255 - grid[i][j]*rel,
                                  255 - grid[i][j]*rel, 255 ),
                        1 )
                );

                qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                    cellWidth, cellHeight );
            }

            qpainter.setPen (
                QPen (
                    QColor ( 0, 0, 0 ),
                    1 )
            );

            qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight );
        }
    }

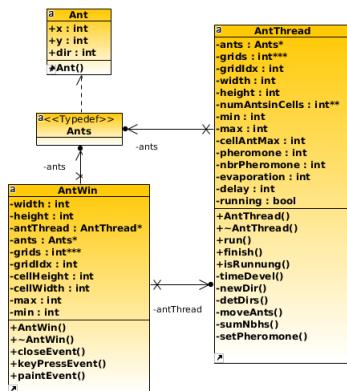
    for ( auto h: *ants) {
        qpainter.setPen ( QPen ( Qt::black, 1 ) );
        qpainter.drawRect ( h.x*cellWidth+1, h.y*cellHeight+1,
```

```
        cellWidth-2, cellHeight-2 ) ;  
    }  
  
    qpainter.end() ;  
}  
  
AntWin::~AntWin()  
{  
    delete antThread;  
  
    for ( int i=0; i<height; ++i ) {  
        delete[] grids[0][i];  
        delete[] grids[1][i];  
    }  
  
    delete[] grids[0];  
    delete[] grids[1];  
    delete[] grids;  
  
    delete ants;  
}  
  
void AntWin::step ( const int &gridIdx )  
{  
  
    this->gridIdx = gridIdx;  
    update();  
}
```

Az antwin.cpp-ben hozzuk létre magát az ablakot. minden kis négyzet 6x6 pixel lesz. For ciklusok segítségevel kirajzoltatjuk ezeket a négyzeteket és elkezdjük feltölteni őket hangyákkal. A kód végén felszabadítjuk a memóriát.

Az antthread.cpp fájlban adjuk meg a hangyák viselkedését. A sumNBHS függvényel tudjuk számonkövetni az egyes hangyák szomszédszámát. A newDir-el új irányt adunk a hangyáknak. Itt figyeljük a szomszédai viselkedését, ugyanis a hangyák feromon alapján követik egymást. A feronomot láthatjuk is futás közben először nagyon élesen majd egyre halványodik. A moveAnts függvényel tudjuk mozgatni a hangyákat.

Az osztálydiagrammban az osztályokat és azok kapcsolatait lehet ábrázolni. Ennek a programnak a következő az osztálydiagramma.



7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/Qt/Sejtauto>

Ebben a feladatban ismét egy Qt-s projektet készítünk el. Most egy sejtautomatát fogunk készíteni C++-ban. Ezeket a sejtautomatákat modellezésre használják. Egy ilyen program leggyoribb formája két dologból áll. Egy négyzetrács illetve a négyzetrácsban találhatóak a sejtek. A program futása során, ahogy telik az idő a sejtek változtatják állapotukat.

Mi ebben a feladatban az egyik legismertebb ilyen sejtautomatával fogunk foglalkozni, a John Conway által kifejlesztett életjátékkal. Ennek a programnak háttere olyan lesz mint egy négyzetrácsos füzet, és minden sejtnak nyolc darab szomszédja lesz. Ebben a modellben a sejteknek két állapota lehet, vagy élők vagy pedig halottak. A sejtek az idő műlásával megadott szabályok szerint fognak állapotot változtatni. Ezek a szabályok a következők:

- Egy olyan helyre ahol halott sejt van, de van 3 élő szomszédja, egy új sejt születik
- Egy olyan sejt amely él, és van legalább két élő szomszédja, az továbbra is élni fog
- Az összes többi sejt halott lesz

A program több fájlból fog állni.

```
#include "sejtablak.h"
```

```
SejtAblak::SejtAblak(int szelesseg, int magassag, QWidget *parent)
    : QMainWindow(parent)
{
    setWindowTitle("A John Horton Conway-féle élatjáték");

    this->magassag = magassag;
    this->szelesseg = szelesseg;

    QDesktopWidget *d = QApplication::desktop();

    int sz = d->width();
    int m = d->height();

    cellaSzelesseg = sz/this->szelesseg;
    cellaMagassag = m/this->magassag;

    setFixedSize(QSize(this->szelesseg * cellaSzelesseg, this->magassag * ↵
                      cellaMagassag));

    racsok = new bool**[2];
    racsok[0] = new bool*[this->magassag];
    for(int i=0; i<this->magassag; ++i)
        racsok[0][i] = new bool [this->szelesseg];
    racsok[1] = new bool*[this->magassag];
    for(int i=0; i<this->magassag; ++i)
        racsok[1][i] = new bool [this->szelesseg];

    racsIndex = 0;
    racs = racsok[racsIndex];

    for(int i=0; i<this->magassag; ++i)
        for(int j=0; j<this->szelesseg; ++j)
            racs[i][j] = HALOTT;

    sikloKilovo(racs, 5, this->magassag-20);

    eletjatek = new SejtSzal(racsok, this->szelesseg, this->magassag, 80, ↵
                             this);
    eletjatek->start();

}

void SejtAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);

    cellaSzelesseg = size().width()/this->szelesseg;
    cellaMagassag = size().height()/this->magassag;
    bool **racs = racsok[racsIndex];
    for(int i=0; i<magassag; ++i) { // végig lépked a sorokon
        for(int j=0; j<szelesseg; ++j) { // s az oszlopok
```

```
// Sejt cella kirajzolása
if(racs[i][j] == ELO)
    qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                      cellaSzelesseg, cellaMagassag, Qt::black) ←
                      ;
else
    qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                      cellaSzelesseg, cellaMagassag, Qt::white) ←
                      ;
qpainter.setPen(QPen(Qt::gray, 1));

qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
                  cellaSzelesseg, cellaMagassag);
}

}

qpainter.end();
}

SejtAblak::~SejtAblak()
{
    delete eletjatek;

    for(int i=0; i<magassag; ++i) {
        delete[] racsok[0][i];
        delete[] racsok[1][i];
    }

    delete[] racsok[0];
    delete[] racsok[1];
    delete[] racsok;
```

A fenti kód a sejtablak.cpp. Ezzel hozzuk létre magát a sejtteret. Beállítjuk az ablak nevét, ezt fogja kiírni a program futásakor. A kód elején hivatkozunk a sejtablak.h file-ra, ebben található a sejtablak osztály. Ebben a kódban létrehozzuk a 100x75-ös ablakot. Valamint létrehozzuk a 6x6-os cellákat. Kezdetben minden cella üres lesz. Majd elindítjuk az életjátékot és ha egy sejt élő lesz akkor feketére festi az adott cellát, ha halott akkor pedig fehér marad. A siklokilovo függvény segítségével fognak a sejtek mozogni.

```
#include "sejtszal.h"

SejtSzal::SejtSzal(bool ***racsok, int szelesseg, int magassag, int ←
                    varakozas, SejtAblak *sejtAblak)
{
    this->racsok = racsok;
    this->szelesseg = szelesseg;
    this->magassag = magassag;
    this->varakozas = varakozas;
    this->sejtAblak = sejtAblak;

    racsIndex = 0;
```

```
}
```



```
int SejtSzal::szomszedokSzama(bool **racs,
                                 int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;

    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)

            if(!((i==0) && (j==0))) {

                int o = oszlop + j;
                if(o < 0)
                    o = szelesseg-1;
                else if(o >= szelesseg)
                    o = 0;

                int s = sor + i;
                if(s < 0)
                    s = magassag-1;
                else if(s >= magassag)
                    s = 0;

                if(racs[s][o] == allapot)
                    ++allapotuSzomszed;
            }

    return allapotuSzomszed;
}
```



```
void SejtSzal::idoFejlodes() {

    bool **racsElotte = racsok[racsIndex];
    bool **racsUtana = racsok[(racsIndex+1)%2];

    for(int i=0; i<magassag; ++i) { // sorok
        for(int j=0; j<szelesseg; ++j) { // oszlopok

            int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);

            if(racsElotte[i][j] == SejtAblak::ELO) {

                if(elok==2 || elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            } else {

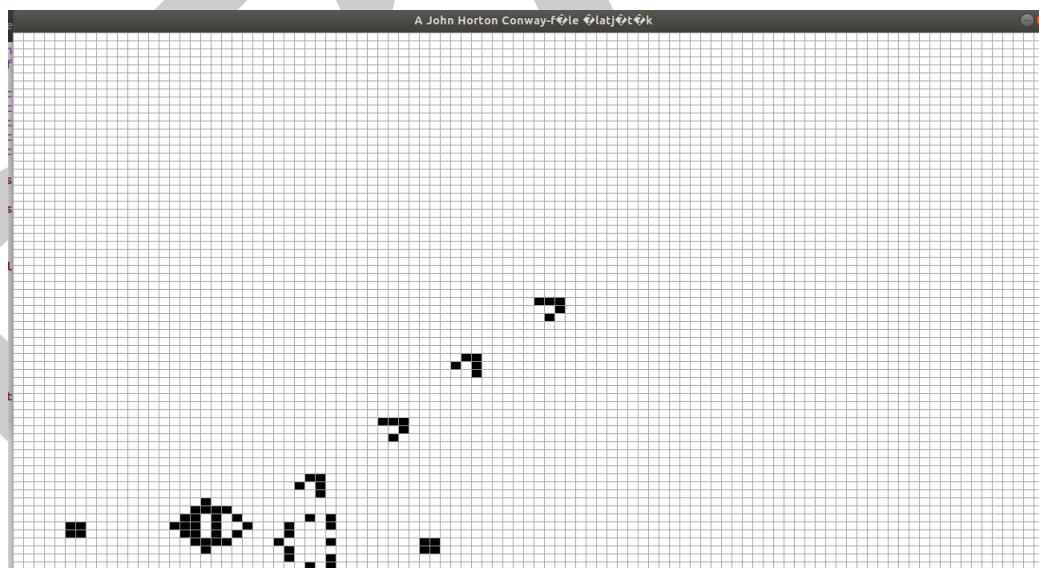
```

```
        if(elok==3)
            racsUtana[i][j] = SejtAblak::ELO;
        else
            racsUtana[i][j] = SejtAblak::HALOTT;
    }
}
racsIndex = (racsIndex+1)%2;
}

void SejtSzal::run()
{
    while(true) {
        QThread::msleep(varakozas);
        idoFejlodes();
        sejtAblak->vissza(racsIndex);
    }
}
```

Ez a másik fontos fájl, a `sejtszal.cpp`. Végigmegyünk for ciklusokkal az adott sejt összes szomszédján(magát kihagyva) és a Conway szabályok szerint figyeljük az állapotukat. Ha két vagy több élő szomszédja van akkor élő marad ha három élő szomszéda van akkor pedig halott marad.

A `main.cpp` ezeket a fájlokat fogja össze, igazából minden lényeges rész ezekben található meg. A fordítást a Qt-vel végezzük.



7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: <https://github.com/nbatfai/esport-talent-search>

Tutor: Kiss Máté

Ez a projekt is egy Qt-s projekt lesz. Most egy kis koncentráció teszteléssel foglalkozunk. Ez a program azt fogja vizsgálni, hogy mennyi ideig tudjuk az egerünket egy megadott dolgon tartani, miközben az az adott doleg mozog illetve folyamatosan jelennek meg ugyanúgy kinéző dolgok a képernyőn. Tehát még arra is figyelnünk kell, hogy jó kis kört kövessünk. A kód az eredményt egy külön mappába fogja elmenteni, így figyelemmel kísérhetjük a fejlődésünket.

Ez a program is több file-ból fog állni. Nézzük először a BrainBWin.cpp-t. Először is kiíratjuk az ablakot, ezen belül lesz verziószámunk, illetve egy órát is jelzünk. A kód folyamatosan figyeli a signalokat, illetve követi az egérmozgását. Az egérmozgását csak akkor fogja figyelni ha levan nyomva a bal egérgomb. Ha nagyon messze kerül a kurzorunk a karaktertől akkor egy változóhoz hozzáfog adni egyet, ezt akkor fogja elmenteni ha több mint 12-szer fordul ilyen elő, ez majd beleszámít a végső pontunkba. A programban az S betű lenyomásával menteni tudunk, a P mint pause betűvel pedig szüneteltetni tudjuk a futását.

```
#include "BrainBThread.h"

BrainBThread::BrainBThread ( int w, int h )
{
    dispShift = heroRectSize+heroRectSize/2;

    this->w = w - 3 * heroRectSize;
    this->h = h - 3 * heroRectSize;

    std::srand ( std::time ( 0 ) );

    Hero me ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100,
              this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
              100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 9 );

    Hero other1 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  5, "Norbi Entropy" );
    Hero other2 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  3, "Greta Entropy" );
    Hero other4 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
                  this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
                  5, "Nandi Entropy" );
    Hero other5 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
                  ) - 100,
```

```
    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        7, "Matyi Entropy" );
```

```
heroes.push_back ( me );
heroes.push_back ( other1 );
heroes.push_back ( other2 );
heroes.push_back ( other4 );
heroes.push_back ( other5 );
```

```
}
```

```
BrainBThread::~BrainBThread()
{
```

```
}
```

```
void BrainBThread::run()
{
    while ( time < endTime ) {

        QThread::msleep ( delay );

        if ( !paused ) {

            ++time;

            devel();

        }

        draw();
    }

    emit endAndStats ( endTime );
}
```

```
void BrainBThread::pause()
{

    paused = !paused;
    if ( paused ) {
        ++nofPaused;
    }
}
```

```
void BrainBThread::set_paused ( bool p )
```

```
{  
  
    if ( !paused && p ) {  
        ++nofPaused;  
    }  
  
    paused = p;  
  
}
```

A másik fontos fájl a BrainBThread.cpp. Ebben létrehozzuk az 5 hőst, az egyik mi leszünk. Ezeknek mindenek meghatározzuk a koordinátájukat. A run eljárással fogjuk indítani a program futását, valamint ebben a kódban figyelünk arra is, hogy az adott képfrissítések ne legyenek túl gyorsak az emberi szemnek. A BrainBThread.h-ban van megadva a mi hősünk, Samu néven, nekünk ezen a hősön kell tartani a program futása során a kurzorunkat. A mérést nehezíti, hogy a másik négy hőst a mi hősünkhez nagyon közel tesszük le, ezért nehéz lehet követni.



8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó: <https://youtu.be/j7f9SkJR3oc>

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa...
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

A fenti feladat megoldásához szükségünk lesz a tensorflow telepítésére. Ezt a következőképpen tehetjük meg:

```
sudo apt install python3-dev python3-pip  
sudo pip3 install -U virtualenv # system-wide install  
virtualenv --system-site-packages -p python3 ./venv  
source ./venv/bin/activate # sh, bash, ksh, or zsh  
pip install --upgrade pip  
pip install --upgrade tensorflow  
#ellenőrizzük, hogy helyesen települt-e  
python -c "import tensorflow as tf; tf.enable_eager_execution(); print(tf. ↵  
    reduce_sum(tf.random_normal([1000, 1000])))"
```

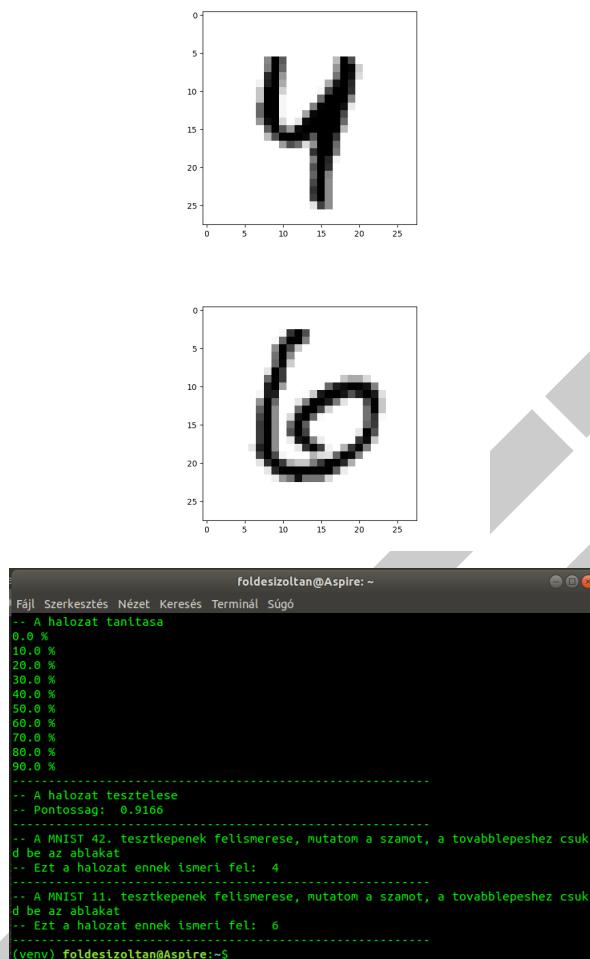
A tensorflow egy ingyenes és nyílt forráskódú könyvtár. Különböző dolgokra lehet használni, köztük gépi tanulásra, azon belül neurális hálókhöz is. Mi ebben a feladatban arra fogjuk használni, hogy segítségével kézzel írt számokat felismerjen a gépünk. Ehhez szükségünk van egy adatbázisra, ez az MNIST lesz. Az MNIST-ben 60.000 kép van és 10.000 teszt kép, melyek kézzel írt arab számokat ábrázolnak.

```
from __future__ import absolute_import  
from __future__ import division  
from __future__ import print_function  
  
import argparse  
  
# Import data  
from tensorflow.examples.tutorials.mnist import input_data
```



```
print("-----")  
  
print("-- A MNIST 42. tesztképének felismereése, mutatom a számot, a ←  
      továbblepeshez csukd be az ablakat")  
  
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 hálózat ennek ismeri fel: ", classification[0])  
print("-----")  
  
print("-- A MNIST 11. tesztképének felismereése, mutatom a számot, a ←  
      továbblepeshez csukd be az ablakat")  
  
img = mnist.test.images[11]  
image = img  
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 hálózat ennek ismeri fel: ", classification[0])  
print("-----")  
  
if __name__ == '__main__':  
    parser = argparse.ArgumentParser()  
    parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ ←  
        mnist/input_data',  
                      help='Directory for storing input data')  
    FLAGS = parser.parse_args()  
    tf.app.run()
```

A kód elején beimportáljuk az MNIST adatbázisból a képeket, majd a neurális hálóhoz létrehozzuk a megfelelő változókat, és az y-ban tároljuk az egyenletet, amely 0 vagy 1 értéket vehet fel. Az x változóban fogjuk tárolni a képeket. A cross entropy függvénytel azt vizsgáljuk, hogy a becsült illetve a valós értéknek mennyi a különbsége, tehát ez a szám minél kisebb annál pontosabb lesz a program. Ezután lépésekben megtörténik a tanítás, majd a végén teszteljük két képpel a létrejött programot.



Amint látjuk a program futásakor százelékosan jelzi a tanítás állapotát, illetve, hogy az első képet ténylegesen 4-esnek a másodikat pedig ténylegesen hatosnak ismerte fel.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndi8>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Tutorált: Kiss Máté

Megoldás video:

Megoldás forrása:[https://hu.wikipedia.org/wiki/Lisp_\(programoz%C3%A1si_nyelv\)#P%C3%A9ldaprogramok](https://hu.wikipedia.org/wiki/Lisp_(programoz%C3%A1si_nyelv)#P%C3%A9ldaprogramok)

Ebben a feladatban a lisp programozási nyelvvel fogunk megismерkedni. Egy rekurzív illetve egy nem rekurzív faktoriálist számláló függvényt fogunk létrehozni. A lisp telepítéséről számos videót találunk az interneten. A lisp nyelvet egy általános célú programnyelvnek szánták, azonban a mesterséges intelligencia kutatás nyelveként is használták. A lisp legfőbb adatstruktúrája a láncolt lista. Szintaxtisa a prefix jelölés jellemző. A teljesen zárójeles forma könnyíti a programkódok könnyen elemezhetőek, azonban emberi olvasásánál könnyen el lehet veszni a sok zárójelben. Nézzük is a függvényeket. Kezdjük a rekurzív függvényeket.

```
(defun faktorial (n)
  (if (= n 0)
      1
      (* n (faktorial (- n 1))))))
```

Amint látjuk az összes kifejezés teljesen zárójelezett. A defun szócskával tudjuk deklarálni a függvényt, mögötte zárójjelben pedig láthatjuk, hogy egy n változót fog kapni paraméterként. Ezután egy if - el megnézzük, hogy a kapott szám 0-e, ha igen akkor egyet fog visszaadni a függvény. Ezt is prefix módon adjuk meg tehát a (= n 0) azt jelenti, hogy n=0. Ha nem 0 akkor pedig n-t megszorozzuk a faktorial n-1-el, ezeket is prefix módon írjuk be. A faktorial szóval hívjuk meg saját magát, és ez mindaddig fogja magát meghívni amíg nem nulla lesz, és akkor már tudjuk az eredményt, hogy 1. Ezután visszafele fog dolgozni a függvény és a végén megkapjuk az eredményt. Most nézzük meg ugyanezt nem rekurzívan.

```
(defun faktorial (n)
  (loop for i from 1 to n
        for fakt = 1 then (* fakt i)
        finally return fakt))
```

Ugyanúgy létrehozzuk a függvényt. Majd a loop for szócskával egy ciklust indítunk ami 1-től fog a kapott számig menni. Majd ezután egy változót indítunk ami minden egyes körben egyenlő lesz az akkori érté-

ke*az adott i értékével.Így kapjuk meg a faktoriális értékét, majd a függvény végén a return-el visszaadjuk ennek a változónak az értékét,ez lesz az eredmény.

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

Az SMNIST kísérletet megcsináltam lvl.9-ig ezért ezt a feladatot passzoltam.

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Az SMNIST kísérletet megcsináltam lvl.9-ig ezért ezt a feladatot passzoltam.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

A könyv elején megismerkedünk az alapfogalmakkal. A számítógépek programozási nyelvének 3 szintjét különböztetjük meg: gépi nyelv, assembly szintű nyelv illetve magas szintű nyelv. Mi a magas szintű nyelvekkel foglalkozunk részletesebben. Az ezen a nyelveken megírt programot forrásprogramnak nevezzük. Ezeknek vannak szintaktikai szabályai, ezek az adott kód nyelvtani szabályai. Valamint vannak a szemantikai szabályok a tartalmi, értelmezési illetve jelentésbeli szabályok tartoznak ide. Ezeket a forrásszövegeket kétféleképpen tudjuk gépi kód díszítési forma vagy interpreteres módszerrel. A fordítóprogram lexikális, szintaktikai majd szemantikai elemzést végez, és ez után generál kódot. Ezzel létrejön egy tárgykód, ez már gépi nyelvű, de még nem futtatható. Ez a kód átkerül egy kapcsolatszerkesztő programnak, és így áll elő egy futtatható program. Az interpreteres módszer anniban különbözik, hogy az nem alkot tárgykódot hanem egyből végrehajtja a forráskód feladatait, így egyből kapunk eredményt. Egy program saját szabványát hivatkozási nyelvnek nevezzük. Ebben vannak definiálva a szemantikai illetve a szintaktikai szabályai az adott nyelvnek. Napjainkban már programok írásához grafikus integrált környezetet használnak.

A programnyelveket három féleképpen tudjuk csoportosítani: Imperatív, deklaratív illetve máselvű nyelvek. Az imperatív nyelvek algoritmikusak, utasításokból állnak a változó a legfőbb programozói eszköz és szorosan kötődnek a Neumann-architektúrához. Ezeknek alcsoportjai az eljárásorientált illetve az objektumorientált nyelvek. A deklaratív nyelvek nem kötődnek annyira a Neumann-architektúrához, nem algoritmikusak, a programozó csak a problémát adja meg, nincs lehetőség memóriamegletre. Alcsoportjai Funkcionális illetve logikai nyelvek. minden más nyelv a máselvű nyelvek közé sorolható.

A kifejezések segítségével, különböző értékekből, új értéket tudunk meghatározni. Ezeknek két összetevőjük van, érték és típus. A kifejezések operandusból, operátorokból illetve kerek zárójelekből állhatnak. Az operátorok a műveleti jelek, a zárójelek segítségével a végrehajtás sorrendjén tudunk változtatni, az operandusok pedig az értékek. Vannak egy operandusú, két operandusú és három operandusú kifejezések. Operátor sorrend szerint van infix, postfix és prefix ábrázolás. Az a folyamat amikor a kifejezés értéke és típusa kiszámolásra kerül kiértékelésnek nevezzük. Az infix alak nem egyértelmű, az operátorok erősségeit egy precedencia táblázatban adják meg. Az infix teljesen bezárójelezett alakja teljesen egyértelmű. A kifejezés típusát két féleképpen lehet meghatározni. Ha ugyan olyan típusú operandusok vannak a kifejezésben akkor típusegyenértékűség lesz, ha nem akkor pedig típuskényszerítés. A konstansok értéke fordítási időben dőlnek el. A C egy kifejezésorientált nyelv. Típuskényszerítő elvét vallja. A könyv itt bemutatja az operátorokat.

Az adattípusok konkrét programozási eszközök, mindegyiknek van egy neve, ez az azonosítójuk. Egy adattípushatároz meg: tartomány, műveletek, reprezentáció. Az adattípusok tartománya azokat az értékeket tartalmazza amit felvehet. A reprezentáció megadja, hogy hánnyájú lehetséges az adott típus. minden típusos nyelv rendelkezik alap típusokkal, de van olyan is ahol a programozó definiálhat saját típusát. Ezzel jobban lehetővé téve a modellezést. Saját típus létrehozáskor a fentebb említett három dolgot kell definiálnia a programozónak. Vannak egyszerű illetve összetett adattípusok. Az egyszerűeket már nem lehet tovább bontani, azonban az összetett összes eleme valamilyen egyszerű típusra bontható. Egyszerű típusok: Egész és valós (lebegőpontos ábrázolás), ezek numerikus típusok ezeken numerikus és összehasonlító műveleteket lehet végrehajtani. Karakteres típusok tartományába a karakterláncok tartoznak, a sztring pedig karakterszorozatból áll, szöveges és hasonlító műveleteket lehet végezni. Egyes nyelvek ismerik a logikai típusat, ez igaz vagy hamis lehet, logikai illetve összehasonlító műveletek végezhetőek el vele. Összetett típusok: Idefiníciók a tömb és a rekord. A tömb elemei ugyanolyan típusúak. A tömböt mint típusat meghatározza: dimenzióinak száma, indexkészletének típusa és tartománya és elemeinek a típusa. A C nem ismeri a több dimenziós tömböt ezért úgy képzeljük el, hogy egy tömb elemei tömbök lesznek. A rekord elemei különböző típusúak lehetnek. A mutató egy adott tárterületre fog mutatni. A nevesített konstans három dologból áll: név, típus és érték. Ezt olyankor használjuk ha egy értéket sokszor szeretnénk használni a programban. Az egész program futása során állandó lesz. A változók négy részből állnak: név, attribútumok, cím és érték. A név egy azonosító, a legfőbb attribútum a típus. Többféleképpen lehet deklarálni: Explicit (a program végzi valami eredményeként), implicit (programozó végzi), automatikus (a fordítóprogram végzi). A cím a tárterületet adja meg ahol tárolva van. Többféleképpen lehet tárterületet hozzárendelni, ezt a könyv részletesen taglalja. Értéket adhatunk értékkadó utasítással, kezdőértékkadással.

Ezután megismerkedünk a C-ben lévő típusokkal majd láthatunk pár példát deklarációra. A typedefet is taglalja a könyv valamint a struktúra megadását is.

Ezekután az utasításokról olvashatunk részletesen. A fordító ezek segítségével készíti el a tárgyprogramot. Két csoportja van: deklarációs és végrehajtható, az utóbbit használja a tárgykód elkészítéséhez. Ezeknek 9 csoportja van: értékkadó utasítás, üres utasítás, ugró utasítás (go to), elágaztató utasítás (if, if else, switch), cikluszervelő utasítások (while, for, do while), vezérlésátadó (continue, break, return), I/O illetve egyéb utasítások. Ezeket a könyv részletesen írja le illetve példákat is mutat rá.

10.2. Programozás bevezetés

[KERNIGHAN RITCHIE]

Megoldás videó: <https://youtu.be/zmfT9miB-jY>

A könyv vezérlési szerkezetek című fejezetében a különböző ilyen szerkezeteket írja le a könyv részletesen. Ezekkel a vezérlési szerkezetekkel határozhatjuk meg, hogy a program a különböző műveleteket milyen sorrendben hajtsa végre. Egy kifejezés, pl. egy deklaráció, vagy függvény meghívás akkor válik utasítássá ha egy pontos vesszőt teszünk utána. Kapcsos zárójelek segítségével ún. blokkokat hozhatunk létre. Ezekben a blokkokban több utasítás lehet. Ezt pl. az if-nél használjuk. Az if-else utasítással döntéseket hozhatunk. Az if után zárójelek között kell megadunk a vizsgált kifejezést, és ha az igaz lesz (nem nullát kapunk vissza) akkor az if utáni utasítást végrehajtja a program, ha hamis lesz akkor pedig tovább ugrik az else ágra. Az else ágat el lehet hagyni, illetve az if után blokk is állhat. Az else-if segítségével több if-et is megadhatunk, és a program futása során az összeset megfogja vizsgálni, és ha egyik sem lesz igaz, csak akkor ugrik majd az else ágra. A switch utasítással többirányú elágazást hozhatunk létre a programban. A switch szó után zárójelek között megadjuk a vizsgált kifejezést, majd case szó után megadunk még egy kifejezést, és ha az

a kifejezés igaz lesz akkor az utána lévő utasítást hajtja végre a program. Akármennyi case-t létrehozhatunk. Tehát például ha egy dolgozatot szeretnénk osztályozni akkor a dolgozat pontszámát beírjuk a switch mögé, a case szavak után pedig a különböző pontthatárokat és azután, hogy az hanyas érdemjegynek felel meg. A program ki fogja választani, hogy melyik intervallumba illik be a pontszám és kiírja az érdemjegyet. Az utasítás végén megadhatunk egy default értéket, ez akkor fog végrehajtódni ha egyik esetre se igaz az adott kifejezés. Tehát például ha valakinek több pontja lenne mint a max pontszám akkor a default utasítás hajtódna végre. A break utasítással bármelyik utasításból kiléphetünk manuálisan. A while illetve for utasításokkal ciklusokat hozhatunk létre a programban. A while után zárójelek között kell megadnunk egy kifejezést, ez a ciklus mindaddig fog futni amíg a a kifejezés igaz lesz (akkor áll meg ha hamissá válik). Ha igaz, akkor a megadott utasítást végrehajtja. A for utasítással is ciklusokat tudunk létrehozni a programban. Ilyenkor a for szócska után zárójelben 3 kifejezést kell megadnunk. A zárójelben az első kifejezésben megadjuk azt ahonnan indulunk (vagy értékadással vagy egy függvényel), a második kifejezés általában egy relációs kifejezés, ezt fogja vizsgálni a program (ha igaz akkor fog lefutni) a harmadik kifejezéssel általában a ciklusváltozóval csinálunk valamit (vagy növeljük vagy csökkentjük). A for ciklust nagyon egyszerűen át tudjuk írni while ciklusra. Az első kifejezést kell a while előtt írnunk, a második kifejezés a while utáni zárójelbe kerül, a 3. kifejezés pedig utasításként a while utáni blokkba kerül. Az, hogy melyik ciklust használjuk az adott programban, teljesen a programozóra van bízva. Általában attól használjuk amelyik kézenfekvőbbnek tűnik. Ha a for ciklus utáni zárójelek közé két pontosvesszőt írunk, akkor egy végtelen ciklust hozunk létre. Ilyenkor a 2. kifejezést minden igaznak fogja kiértékelni a program és a break utasítással vagy a return utasítással tudunk manuálisan kilépni a ciklusból. A while és for ciklusok feltételeit a program minden ciklus futása előtt ellenőrzi le, így előfordulhat az, hogy az adott ciklus egyszer sem fut le (ha a feltétel a kezdetektől fogva hamis). Ezekkel ellentétben a do-while ciklus ún. hátróltesztelős ciklus, azaz egyszer biztosan lefog futni, mivel csak a ciklus futása végén ellenőrzi a feltétel igazságát. Ezt a ciklust sokkal ritkábban használják, de ritkán nagyon hasznos. A break utasítással az előbb már átbeszélt utasításokból idő előtt ki tudunk lépni. A continue utasítás a switch utasításban nem használható. A while esetén ez az utasítás azt fogja eredményezni, hogy egyből végrehajtja a feltételvizsgálatot. Általánosan pedig a következő utasításra ugrik. A goto utasítással egy címkerére ugorhatunk, általában nem használják. De ha igen akkor a leggyakoribb felhasználási módszere, ha több egybeágynak ciklusból szeretnénk kilépni, ugyanis itt nem használható a break utasítás.

Ezek az utasítások leírásuk sorrendjében hajtódnak végre illetve különböző sorrendbe sorolhatóak. Hat fő csoportba tudjuk sorolni az utasításokat. A címkezett utasításoknál előtagként egy címkét írhatunk az utasítás előtt. Ez a címke egy azonosítóból áll. A következő csoport a kifejezés utasítás, a legtöbb utasítás ilyen. Ide tartozik a függvény hívás vagy az értékadás, pontosvesszővel zárjuk le. Az összetett utasítások csoportjába tartozik a blokk. Ezáltal több utasítást egy utasításként tudunk kezelni. Ezt a kapcsos zárójelekkel lehető meg. A kiválasztó utasítások közé tartozik az if illetve a switch utasítás. Az ebbe a csoportba tartozó utasítások már megváltoztatják a végrehajtási sorrendet. Az if-el található else utasítás minden a blokkban lévő utolsó if-el van párban. A switch utasítás után bármennyi case állhat, default viszont csak egy. Az iterációs utasítások csoportjába tartozik a while, do-while illetve for utasítások. Ezek egy ciklust definiálnak. A vezérlésátadó csoportba a goto, continue, break illetve return utasítások tartoznak. Ezekkel feltétel nélkül átadhatjuk a vezérlést. A goto működéséhez szükséges egy címke, amire át fog tért a vezérlés. A continue utasítással egy ciklusban újra megvizsgálásra kerül a feltétel. A break-el a következő utasításra ugordhatunk. A returnel visszatérítési értéket adhatunk meg.

10.3. Programozás

[BMECPP]

A könyv részletesen fejti ki az objektumorientáltságot.A C++ mint már tudjuk objektumorientált nyelv, ezáltal összetebben problémákat lehet vele megoldani azonban ez teljesítménycsökkenéssel járhat.A C++-ban osztályokat használunk, aminek minden elemét egyednek hívunk.Fontos megemlíteni az adatrejtést, ez azt jelenti,hogy az osztályon belüli bizonyos adatok más programrésznek nem elérhetőek.A könyv tárgyalja a tagfüggvények különböző megadását illetve a tagváltozókról is beszél.Az adatrejtést is kifejti ezután a szerző, elmagyarázza,hogy azért fontos a különböző adatok védelme,hogy azokat más ne tudja megváltoztatni,ugyanis lehet olyan függvény ami azzal az értékkel számol.Ezután szó van a konstruktorról illetve a destruktorról. A konstruktor egy új példány létrehozásakor hívódik meg, és ez foglalja le számára a memóriát. A destruktur pedig egy példány törlésekor felszabadítja a memóriát.A dinamikus adatkezelésről is olvashatunk. Memóriát lefoglalni a new operátorral, a delete operátorral pedig a lefoglalt memóriát törölhetjük.A másolókonstruktorról is olvashatunk, ami egy adott példányt fog lemásolni, ez akkor fog meghívódni ha egy függvényt értékszerinti paraméteradással hívunk meg,így nem fog változni az eredeti példány.Létezik sekély(bitenként másol) illetve mély másolás.Majd a friend függvényeket írja le a könyv, a friend függvények úgyanúgy hozzáférnek a tagfüggvényekhez és a tagváltozók, mintha az osztály tagjai lennének.Friend osztályok is léteznek,ezeknek is ugyanaz a joguk lesz mintha az adott osztályba tartoznának.A statikus tagváltozók az osztály minden objektumában ugyanazt az értéket veszik fel.Az operátorok kiértékelési sorrendje az ún. precedenciatáblázatban van leírva, ezt a sorrendet zárójelek segítségével tudjuk befolyásolni.C++-ban az operátorok túlterhelésével,saját operátorokat hozhatunk létre.Itt a szerző különböző példákkal magyarázza el az operátor túltöltését.

A 10.fejezetben a kivitelkezelésről olvashatunk. C-ben a hibákról, a függvények által visszatérített hibakódokkal illetve globális változókkal tudunk tájékozódni. C++-ban azonban létrehozhatunk ún. kivitelkezelést. Ezáltal sokkal átláthatóbb illetve könnyebben kezelhető lesz a hibakezelésünk.Ebben a fejezetben megismérjük a C-ben lévő hibakezelés problémáit. Illetve arra is kapunk segítséget,hogy hogyan kereljük el a memóriszivárgást a kivitelkezelés során.C-ben ha egy függvényhívás láncot nézünk és valamelyik függvény hibát észlel(például nem tud megnyitni egy fájlt) akkor a visszatérési értékéből tudjuk meg a hibát, ezt a kódot visszaadja egész az első láncszemig, a main függvényig. A main függvényben pedig kezeljük azt.Ez nagyon körülményes illetve nagyon könnyű átsiklani a hibákon, így könnyen előfordulhat az,hogy hibás adattal fogunk dolgozni.Ezeket a hibákat tudjuk kiküszöbölni a C++-os kivitelkezeléssel. Kivitelkezeléskor ugyanis a futás azonnal a hibakezeléses részlegre fog kerülni. Ezzel a módszerrel nem csak hibát, hanem bármilyen kivételes esetet tudunk kezelni.A 190.oldali példában egy try-catch blokkot találunk.Ha megfelelő számot írunk be akkor a try blokkban lévő majdnem összes utasítás lefog futni, majd a végén kiírja a program,hogy done. Ilyenkor csak a throw utasítás nem fut le. Ha azonban nem megfelelő értéket írunk be akkor a throw utasítással egy kivételt dobunk, jelen esetben const char* típusú stringet, ilyenkor a vezérlés azonnal átkerül az ugyan ilyen kivételt 'elkapó' catch-re, és végrehajtódi a megadott utasítás.Ha egy függvény hívási láncot nézünk akkor ilyenkor egyből a megfelelő catch ágra fog kerülni a vezérlés, és nem a main függvényre. Ha egy kezeletlen kivétellel találkozik a program akkor az abort függvény hívódik meg amely a programból való kilépést eredményezi.Több try-catch blokkot egybe tudunk ágyazni illetve egy kivételt többször is dobhatunk.A 197.oldalon a verem visszacsévélésére látuk egy példát. Ez az a jelenség mikor egy kivétel dobásakor és elkapásáig a lokális változók felszabadulnak.Ebből az a tanulság,hogy kivétel dobás és elkapás között kód futhat le.

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

11. fejezet

„Helló, Berners-Lee!”

11.1. Python

A könyv megadott oldalain a Python nyelv megismerésével foglalkozunk .A python egy magasszintű, dinamikus ,objektumorientált és platformfüggetlen nyelv. Leginkább prototípus készítésére és különböző tesztelésekre használják.A python egy köztes nyelv,azaz nincs szükség fordításra, ezért használják előszörrel tesztelésekre,ugyanis kevesebb időt vesz igénybe.Ettől függetlenül nagyon bonyolult problémákat lehet leírni vele.Sokkal rövidebb kódok hozhatóak létre ugyanarra a problémára mint például C++ vagy java nyelveken. A kódot sorbehúzással szerkesztjük, nincs szükség zárójelekre vagy kulcsszavakra. Egy utasítás a sor végéig tart, nincs szükség pontosvesszőre.A behúzást tabulátorral vagy szóközökkel is létrehozhatjuk ,azonban ezekkel konzisztensnek kell lennünk, mert megzavarhatjuk az értelmezést. Az értelmező a kódot soronkénti tokenekre bontja. Ezek a tokenek lehetnek azonosítók, kulcsszavak, operátorok, delimeterek vagy literálok.Ezek után a könyvben egy táblázatot láthatunk ami felsorolja a nyelvben lefoglalt kulcsszavakat .Pythonban az összes adatot objektumok ábrázolnak és a rajtuk végezhető műveletek függnek az adott objektumok típusától.Nincs szükség megadni az adatok típusát ezt a rendszer futási időben kitalálja az adott adathoz rendelt értékek alapján. A számoknak van egész, lebegőpontos illetve komplex ábrázolása is.Az enneselek vagy angolul tuples objektumok gyűjteményei. Ezeknek az objektumoknak nem muszáj ugyanolyan típusúaknak lennieük. Ezeket sima zárójelek közé írjuk.

```
#példa ennesekre  
  
#egy elemű ennes ('a','b','c')  
#tuple kulcsszó tuple('a','b','c')
```

A szögletes zárójelekkel megadott dolgo pedig a lista . Ebben különböző típusú elemek lehetnek .Kapcsos zárójelekkel a szótárakat adhatjuk meg .Ez egy rendezetlen halmaz.

```
['a','b','c']
```

Pythonban a NULL neve `None`.A változók az egyes objektumokra mutató referenciák.Ha egy változó már más objektumra mutat akkor a az automatik garbage collector hívódik meg. Ez a felesleges memóriaterület fogja felszabadítani.Érétkadásra az egyenlőségjelet használjuk. Változóna kértékül adhatunk objektumot típushoz függvényt.A del kulcsszó egy változó hozzárendelését fogja törlni, a tényleges törlést a garbage collector fogja elvégezni.Ugyanúgy léteznek itt is globális illetve lokális változók. Ha egy függvényben deklarálunk egy változót akkor az alapesetben lokális lesz, azonban ha a függvény elején deklaráljuk és élé

írjuk a GLOBAL utasítást akkor globális változóként fog létrejönni.A Python rövidre zárt kiértkelést végez, erre példa:

```
a<b<=c==d ami ezt fogja jelenetni: a<b és b<=c és c==d
```

Különböző típusok közt konvertálhatunk illetve sztringből is képezhetünk számot.A sztringeket listákat illetve az enneseket együtt szekvenciáknak nevezzük. Ezeken a szekvenciákon különböző műveleteket végezhetünk el.A len függvény a szekvencia hosszát, a min illetve a max pedig a szélsőértékeket fogják visszaadni.Az in illetve not in el pedig arra kaphatunk választ,hogy egy adott elem benne van vagy nincs az adott szekvenciában.A könyvben ezután láthatjuk a listákon elvégezhető műveleteket,egy táblázatba fogalva.Ugyanilyen táblázat található a szótákról is. Ezután a könyv áttér a nyelvi eszközökre. Először a print metódus láthatjuk,amivel a konzolra tudunk kiíratni.Ezután az elágazásról esik szó,itt else elif utasításokkal érhető el.Ugyanúgy rendelkezésre áll a for illetve a while ciklus. A for ciklus mellé kapunk egy range függvényt ami menet közben egy listát fog készíteni és alapestben egyessével lépked.Azonban lehetőség van megadni két értéket is,ami a kezdő és vég értéket fogja megadni,majd ezután akár azt is megadhatjuk,hogy mennyit lépjén.Címkeket is elhelyezhetünk a kódunkban a label kulcsszóval,ezeket majd a goto parancssal tudjuk majd elérni.Pythonban is van lehetőségünk függvényeket létrehozni. Ezeket a def kulcsszóval tudjuk definiálni.Ezekre a függvényekre tekinthetünk értékekkel is,mivel továbbadhatóak.A paraméterek érték szerint adódnak át, kivéve a mutable típusok(listák,szótárak).Egy visszatérítési értékük van de visszatérhetnek szekvenciákkal is.Ezekután az osztályokról illetve az objektumokról olvashatunk.A python támogatja mind a klasszikus és objektumorientált eljárásokat is.Létrehozhatunk osztályokat melyeknek példányai az objektumok lesznek.Az osztályok örökölhetnek más osztályuktól.Az osztályban lévő függvényeket tagfüggvényeknek nevezzük.A tagfüggvényeket ugyanúgy definiálhatjuk mint a globális függvényeket, azonban van egy kötelező első paraméterük, a self.A self értéke minden az objektum lesz mely a függvényt meghívta.Az __init__ szócskával megadott metódusa konstruktur tulajdonságú lesz. Majd a modulokról ír a könyv.Ezek a modulok a fejlesztés megkönnyítése érdekében lettel létrehozva.Ezeken a modulokon belül is a mobilhoz létrehozott modulokat nézzük meg.A appuifw a felhasználói felület kialakításáért felelős.A messaging modul az SMS és MMS kezelését könnyíti meg.A sysinfoval a telefon adatát kérhetjük le.A camera modullal a kamerát kezelhetjük.Az audio modullal pedig a hangfelvételeket illetve ezek lejátszását kezelhetjük.Kivételkezelésre is van lehetőségünk.A try kulcsszó után írhatjuk azt a kódblokkot ahol előfordulhat a váratlan helyzet, és ha ez bekövetkezik akkor except blokk veszi át a vezérlést. Valamint megadhatunk még egy else ágat is ezek után. Az utolsó két blokk helyett írható egy finally blokk is.Ez például egy hiba esetén bezárhat egy fájlt.Ezután a könyvben példákat láthatunk a fent leírtakhoz.

11.2. C++,java

Ebben az olvasónaplóban a megadott két könyv szerint fogjuk összehasonlítani a java és a C++ programozási nyelveket. A két nyelv szintaxisa nagyon hasonló.

Java és C++ között már a program fordítása során is van különbség.A java az alkalmazásokat bájkódra fogja alakítani,majd ez egy virtuális gép fogja futtatni,míg a C++ kódból natív kód lesz. Megjegyzésekkel ugyanúgy rakhattunk a kódba mint C++-ban,azonban itt már rendelkezésre áll dokumentációs megjegyzés. Ezt a per jel után két csillaggal tudjuk megadni.Ezeket a program majd kiszűri a fordítást során és egy HTML-ként fog kimenteni.A java karakterkészlete már 16 bites,ellentétben a C++-al aminek csak 8 bites karakterkészlete van.

A java fájlunk nevének ugyanannak kell lennie mint a kódban lévő fő osztály, ugyanis a futtató ezt az osztályt fogja keresni, ha ez nem így van akkor nem fog fordulni a kódunk. C++-ban és javaban is rendelkezésre állnak ugyanazok az alap változótípusok (integer, double stb.). Javaban a tömb típus egy igazi típus lesz a C++-ban lévő tömbbel ellentétben, ugyanis C++-ban a tömb csak egy mutatótípus. Azonban ez nem primitív típus, ezek objektumhivatkozást tartalmaznak. Javaban nincsenek többdimenziós tömbök, de megoldható a létrehozásuk ugyanis egy tömb tartalmazhat tömböket is. Ezt a következőképpen tudjuk például megadni:

```
int[][] iaa = new int[3][];
for (int i=0; i<iaa.length; i++) {
    iaa[i] = new int[i+1];
    for (int j=0; j<iaa[i].length; j++) iaa[i][j]=0
}
```

A kifejezések kiértékelése minden nyelvben ugyanúgy történik. Először a legbelül zárójelben lévő kifejezés, ha nincs zárójel akkor először a nagyobb prioritású operátor, ha pedig egyenjogú operátorok vannak egymás mellett akkor balról jobbra fog kiértékelődni a kifejezés. Javaban nem lesznek pointerek illetve referencia jelek, ugyanis javaban minden referenciaként van átadva. Míg C++ pointerezni, referenciazni kell. A Java magasabb szintű programozási nyelv mint a C++. Mindkét nyelvben van lehetőségünk konstansokat létrehozni, javaban ehhez a final kulcsszót használjuk míg C++-ban pedig a const kulcsszóval.

Mindkét nyelvben megtalálhatóak ugyanazok a vezérlési szerkezetek, mint az if, if else, for ciklus, while ciklus, switch.. Azonban a Java-ból kivették a goto utasítást. Ezt a megbízható és biztonságos programok írásának érdekében tették meg. Az objektumorientált fogalmakkal folytatjuk. Javaban a az objektumok a legkisebb önálló egységek, tehát javaban minden objektum. C++-ban az objektumok adatstruktúrák.

Az osztályon belül az adott dolog összes jellemzője megtalálható. Szinte minden leírható egy osztállyal pl. bankszámla, vállalat dolgozói stb. Az osztályokat lehet példányosítani, egy adott példány vagy egyed rendelkezni fog az adott osztály tulajdonságaival. Ezeket az egyedeket vagy példányokat objektumoknak is nevezzük. Javaban az osztályokat egy két részből álló osztálydefiníció írja le. Az első rész azokat a változókat tartalmazza mellyel az objektum állapota írható le. A második rész pedig az osztályon elvégzhető metódusokat tartalmazza. Mind javaban és C++-ban írhatunk konstruktorokat ezek automatikusan meghívódnak ha létrehozunk egy új egyedet. A konstruktor nevének ugyanannak kell lennie mint magának az osztálynak. Olyan mint egy metódus, viszont nincs visszatérítési értéke. Ha egy objektum törlődik, akkor meghívódik az adott osztály destruktora, ez a memóriát fogja felszabadítani. Ezt C++-ban nekünk kell megírni, azonban javaban van egy automatikus szemétfelüjtő mechanizmus. Azonban vannak olyan esetek ahol tudnunk kell, hogy egy objektum megsemmisül, ezt a javaban a finalize metódus adja meg. Ez akkor hívódik meg mielőtt még a szemétfelüjtő felszabadítaná a memóriát. Az osztályváltók olyan változók amelyek magához az osztályhoz tartoznak. Ezeken az osztályváltókban az összes példány osztózik. Az objektumok példányosítással jönnek létre, ezt a new operátorral tudjuk elvégezni. Az operátor mögé megadjuk, hogy melyik osztályt szeretnénk példányosítani és mögé zárójelek közé megadjuk az adott példány paramétereit. Azonban a paraméter szerinti példányosítás csak akkor tud működni ha saját konstruktort hoztunk létre. Ez a konstruktor a megfelelő metódusokat meghívva beállítja a megfelelő értékeket. Javaban létezik az öröklődés aminek legegyszerűbb este amikor egy osztályt egy másik létező osztály kiterjesztésével definiálunk. Ez a kiterjesztés több dolgot is jelenthet pl. új műveletek vagy új változók bevezetését. Az osztályban ezt az extends szóval jelezzük. Ebben az esetben az alap osztály lesz a szülőosztály a bővíttet pedig a gyermek-osztály. Így létrejönnek leszármazottak illetve ősök. A bővíttet osztály rendelkezni fog az újonnan megadott tagokkal illetve a szülő tagjaival is, tehát örökli a tagokat (innen származik a kifejezés). A gyermek azonban a szülő konstruktoraiból nem örökli. Ezért a programozónak gondolkoznia kell, hogy elég ha a szülő kon-

ruktorát hajtja végre,vagy van-e valami új változó amit inicializálni kell a leszármazott osztályban.Minden olyan helyen ahol az adott gyermek ősei használhatóak, ott a gyermek is használható lesz,mivel ugyanúgy rendelkezik azokkal a változókkal és fügvényekkel,ezt polimorfizmusnak nevezzük.Erre nézzünk egy példát:

```
class Madar {  
public void repul() {};  
};  
  
class Prog {  
public void fgv ( Madar madar) {  
    madar.repul();  
}  
};  
  
class Sas extends Madar {};  
class Pingvin extends Madar {};  
  
class Liskov{  
    public static void main(String[] args)  
    {  
        Prog kod = new Prog();  
        Madar mad = new Madar ();  
        kod.fgv (mad);  
  
        Sas sas = new Sas();  
        kod.fgv (sas);  
  
        Pingvin pingvin = new Pingvin();  
        kod.fgv(pingvin);  
    }  
}
```

Erről a példáról részletesebben fogunk foglalkozni a Liskov feladatcsokor szűlő-gyermek feladatánál.

A java nyelv másik fontos eleme az osztályokon kívül az interface.Ez nem része az osztályoknak. Az interface,már C-ben is létezett,azonban ott protokollnak hívták.Ezt vette át a Java kisség módosítva.Az interface egy új referenciatípus.Absztrak metódusok deklarációjának és konstans értékek összessége.Az interfacekben található metódusok csak deklarálódnak,azaz megvalósítás nélkül szerepelnek.Egy interface használata implementáció keresztül történik.Tehát ahhoz,hogy az interfacen belüli metódusokat használhassuk,implementálni kell az adott interfacet.Ezek között az interfacek között is van öröklődés.Ezt a kapcsolatot itt is kiterjesztésnek nevezzük.

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

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

Ebben a feladatban a polártranszformációs generatort fogjuk megírni javaban illetve C++-ban. A teljes programok forrásként megtalálható githubon, ide csak programrészletek lesznek beszúrva.

Ez az algoritmus 10 db random számot fog generálni polártranszformáció segítségével. Kezdjük a javas kóddal. A PolarGenerator osztályban két darab változó található, egy logikai illetve egy double típusú. A logikai nincsTarolt változóval azt jelezzük, hogy van-e tárolt változó vagy sem, a tarolt double változóban pedig a tárolt változó értékét fogjuk tárolni. Ezután található az osztály konstruktora amely a logikai változónkat igazra fogja állítani, ami azt jelenti majd, hogy nincs tárolt változónk.

```
public class PolarGenerator{
    boolean nincsTarolt = true;
    double tarolt;
    public PolarGenerator()
    {
        nincsTarolt = true;
    }
}
```

Ezután egy metódus következik következő néven, ez fogja a random számokat generálni. Ez a metódus azt fogja nézni, hogy van-e tárolt változó. Ha van tárolt változó akkor azt fogja visszaadni értékül. Ha nincs akkor pedig két értéket fog kiszámolni, és az egyiket elfogja tárolni a másikat pedig értékül fogja visszaadni.

Ezután van leírva a main függvényünk, ezzel indul a programunk. Először is létrehozunk egy objektumot a PolarGenerator classbol, majd egy for ciklus segítségével 10-szer meghívjuk a következő metódust, ezáltal létrehozva a 10 darab random számot.

```
public double kovetkezo()
{
    if(nincsTarolt)
    {
```

```
double u1, u2, v1, v2, w;
do{
    u1 = Math.random();
    u2 = Math.random();
    v1 = 2* u1 -1;
    v2 = 2* u2 -1;

    w = v1*v1 + v2*v2;
} while (w>1);
double r = Math.sqrt((-2 * Math.log(w) / w));
tarolt = r * v2;
nincsTarolt = !nincsTarolt;
return r * v1;
}
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
public static void main(String[] args)
{
    PolarGenerator g = new PolarGenerator();
    for (int i = 0; i < 10; ++i)
    {
        System.out.println(g.kovetkezo());
    }
}
```

Nézzük meg ugyanezt a kódot C++ nyelven. Különbségek fognak adódni ugyanis C++ nyelven nem minden osztály még javaban minden osztálynak tekintünk. C++-ban is osztályokkal fogjuk megoldani a feladatot. Név szerint a PolarGen osztállyal. Ennek az osztálynak lesz egy public illetve egy private része. A public részhez a kdon belül bármi hozzá tud érni, míg a private részhez csak az adott osztályon belül tartozó dolgok tudnak hozzáférni. A public részhez három dolog tartozik. Az első rész a konstruktor, melynek ugyanaz a neve mint magának az osztálynak. Ez a konstruktor akkor fog meghívódni amikor az osztályt példányosítjuk és jelen esetben a nincsTarolt változót igazra fogja állítani. A második rész a destruktur amit a ~ jelrel jelölünk, ez akkor hívódik meg amikor egy példányt törlünk. Ez a javas kódunkban nem volt, mivel javabban egy automatikus szemétygyűjtő van, ami felszabadítja a területet ha egy példányra már nincs több hivatkozás. Ezen kívül a public részhez tartozik még a kovetkezo metódus prototípusa, ezt azért adjuk meg, hogy tudja a program, hogy a későbbiekben lesz majd egy ilyen metódus, csak nem itt írjuk le részletesen.

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

```
    }
    double kovetkezo();
```

A private részben a változóink találhatóak amik megegyeznek a javas kódunknál megadott változókkal, ezek azért privátok mert így más programrészlet nem tudja majd megváltoztatni őket.

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

Ezután megadjuk a kovetkezo metódust, ami megint csak megegyezik a javas kódunkban megadott kovetkezövel. Majd következik a main amiben létrehozunk egy PolarGen objektumot majd egy for ciklus segítségével létrehozunk 10 random számot.

```
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*u1-1;
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;
}
```

Kimenet:



```
Foldesizoltan@Aspire:~/Asztal/prog2/arroway/polar
Fájl Szerkesztés Nézet Keresés Términál Súgó
foldesizoltan@Aspire:~/Asztal/prog2/arroway/polar$ java PolarGenerator
-0.40541083242284065
-0.36224767080949155
0.76484891235971
-0.0873189133532974
0.3392055735183245
-0.3773398846938243
-1.2496289845460096
0.4898969279899916
0.009978142398746
-1.4959480369242457
foldesizoltan@Aspire:~/Asztal/prog2/arroway/polar$
```

12.2. Homokózó

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

Ebben a feladatban a már az első fejezetben megírt C++-os binfát fogjuk javában megírni. A binfa működéséről beszélünk a könyv első fejezetében, most csak a C++ és java különbségeiről fogunk

```
language="java">>

public class LZWBinFa {

    public LZWBinFa() {
        fa = gyoker;
    }

    public void egyBitFeldolg(char b) {
        if (b == '0') {
            if (fa.nullasGyermek() == null)
            {
                Csomopont uj = new Csomopont('0');

                fa.ujNullasGyermek(uj);

                fa = gyoker;
            } else
            {
                fa = fa.nullasGyermek();
            }
        }
    }
}
```

```
        }
    else {
        if (fa.egyesGyermek() == null) {
            Csomopont uj = new Csomopont('1');
            fa.ujEgyesGyermek(uj);
            fa = gyoker;
        } else {
            fa = fa.egyesGyermek();
        }
    }
}

public void kiir() {
    melyseg = 0;
    kiir(gyoker, new java.io.PrintWriter(System.out));
}

public void kiir(java.io.PrintWriter os) {
    melyseg = 0;
    kiir(gyoker, os);
}

class Csomopont {

    public Csomopont(char betu) {
        this.betu = betu;
        balNulla = null;
        jobbEgy = null;
    }

    ;

    public Csomopont nullasGyermek() {
        return balNulla;
    }

    public Csomopont egyesGyermek() {
        return jobbEgy;
    }

    public void ujNullasGyermek(Csomopont gy) {
```

```
    balNulla = gy;
}

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

public char getBetu() {
    return betu;
}

private char betu;

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

};

private Csomopont fa = null;

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

public void kiir(Csomopont elem, java.io.PrintWriter os) {

    if (elem != null) {
        ++melyseg;
        kiir(elem.egyesGyermek(), os);

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

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

public int getMelyseg() {
```

```
melyseg = maxMelyseg = 0;
rmelyseg(gyoker);
return maxMelyseg - 1;
}

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

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

    rszoras(gyoker);

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

    return szoras;
}

public void rmelyseg(Csomopont elem) {
    if (elem != null) {
        ++melyseg;
        if (melyseg > maxMelyseg) {
            maxMelyseg = melyseg;
        }
        rmelyseg(elem.egyesGyermek());
        rmelyseg(elem.nullasGyermek());
        --melyseg;
    }
}

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

```
        }

    }

}

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

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

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

    String inFile = args[0];

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

    try {

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

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

```
byte[] b = new byte[1];

LZWBinFa binFa = new LZWBinFa();

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

boolean kommentben = false;

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

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

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

    if (kommentben) {
        continue;
    }

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

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

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

}
```

```
binFa.kiir(kiFile);

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

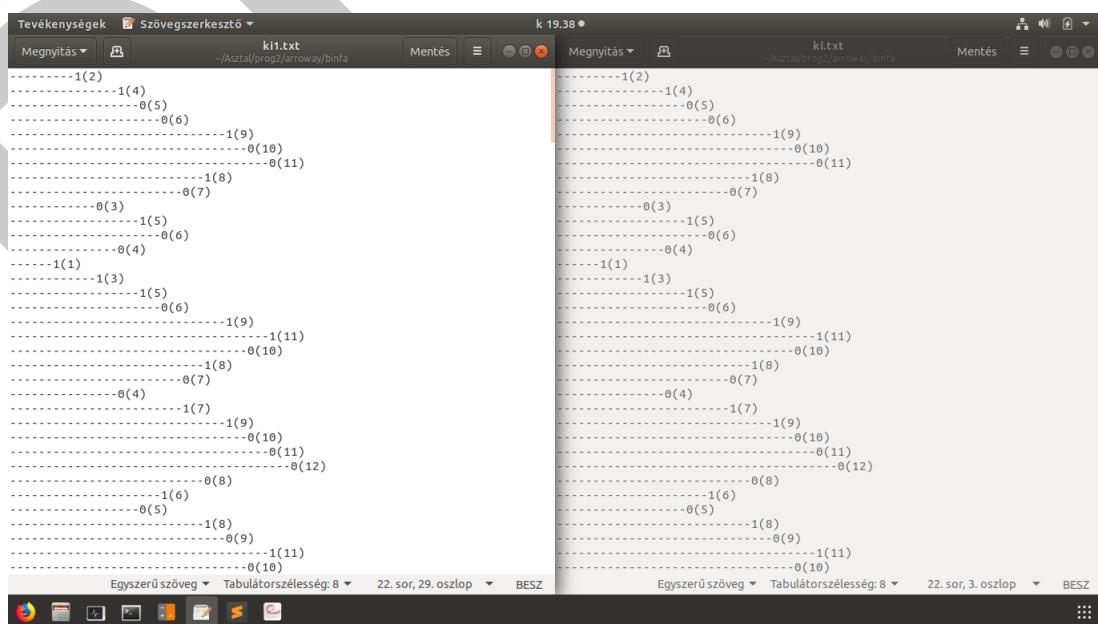
kiFile.close();
beFile.close();

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

}

}
```

Amint látjuk nem sokban különbözik a kódunk a C++-os változathoz képest. Először is a pointerek illetve a referenciajelk tüntek el. Ez azért van mert a java nem támogatja a pointereket. Javaban minden referencia szerint működik. Mikor egy osztályban létrejön egy objektum akkor a referenciáját kapjuk meg. Javaban operátor túlterhelés sincs, ezért ezt egy függvényel helyettesítettük. Ezenkívül ebben a kódban nincs szükségünk destruktorra, ugyanis javaban működik az automatikus szemétfelügyelet, ami automatikusan felszabadítja a nem használt memóriaterületet. Classon belül nincs külön public illetve private rész, ezért minden függvény és eljárás előtt kell írnunk, hogy public vagy private. Ezeken kívül még pár szintaktikai dolgot kellett átírni és minden működik úgy mint C++-ban. A képen látható, hogy ugyanaz a kimenet, tehát az átírás sikeres volt.



A feladat második része,hogy létrehozzunk egy java servletet és a binfa a böngésző címsorából kapja meg a bemeneti adatokat és ezekből készítsen bináris fát.A java servlet egy olyan objektum ami HTTP kéréseket kap és készít.A feladat megoldásához szükség van az apache tomcat telepítésére. Ez a feladat még folyamatban....

Telepítés közben hibába ütköztem, valamiért nem csomagolja ki a fájlokat:

```
földesizoltan@Aspire:/tmp$ ls -l
Összesen 52
-rw-r--r-- 1 földesizoltan földesizoltan 327 szept 25 13:38 apache-tomcat-8.5.5.tar.gz
-rw----- 1 földesizoltan földesizoltan 0 szept 25 12:00 config-err-02Awzz
drwxr-xr-x 2 földesizoltan földesizoltan 4096 szept 25 12:56 hsperrdata_földesizoltan
drwx---- 2 földesizoltan földesizoltan 4096 szept 25 12:31 mozilla_földesizoltan0
drwx---- 2 földesizoltan földesizoltan 4096 szept 25 12:00 ssh-yuR20hb3Qqqd
-rw-r--r-- 1 földesizoltan földesizoltan 675 szept 25 12:37 suite_installer.log
drwx--- 3 root      root      4096 szept 25 11:59 systemd-private-32c2d87b34294db8b28883c9dsaodie3-bolt.service-Wn6CKW
drwx--- 3 root      root      4096 szept 25 11:59 systemd-private-32c2d87b34294db8b28883c9dsaodie3-colord.service-U3JQih
drwx--- 3 root      root      4096 szept 25 12:01 systemd-private-32c2d87b34294db8b28883c9dsaodie3-fwupd.service-yLGrSQ
drwx--- 3 root      root      4096 szept 25 11:59 systemd-private-32c2d87b34294db8b28883c9dsaodie3-rtkit-daemon.service-GFh6V5
drwx--- 3 root      root      4096 szept 25 11:59 systemd-private-32c2d87b34294db8b28883c9dsaodie3-systemd-resolved.service-KNUusb8
drwx--- 3 root      root      4096 szept 25 11:59 systemd-private-32c2d87b34294db8b28883c9dsaodie3-systemd-timesyncd.service-4UWL40
drwx--- 2 földesizoltan földesizoltan 4096 szept 25 12:01 Temp-201d742f-cc4a-4c7a-abef-deb7387c1682
drwx--- 3 földesizoltan földesizoltan 4096 szept 25 12:17 Temp-9723dce2-05b9-430c-a5c6-a1d9a049681
földesizoltan@Aspire:/tmp$ sudo tar xzvf apache-tomcat-8.5.5.tar.gz -C /opt/tomcat --strip-components=1

gzip: stdin: not in gzip format
tar: Child returned status 1
tar: Error is not recoverable: exiting now
földesizoltan@Aspire:/tmp$
```

12.3. Gagyi

Az ismert formális2,,

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

tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciaja) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására3, hogy a 128-nál inkluzív objektum példányokat poolozza!

JDK forrás:

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

A java eltárolja a -128 és 127 intervallumba tartozó integer típusú számokat.Ha egy ebben a tartományban található számot akarunk egy változóhoz rendelni,akkor meg fog hívódni az Integer.valueOf függvény és hozzá fog rendelni egy címet ami már alapból benne volt az úgynevezett poolba,ahol ezek az előre elmentett számok vannak.

Azonban ha egy az intervallumon kívül eső számot rendelünk egy változóhoz akkor a függvény létre fog hozni egy teljesen új objektumot.A java az == és != operátorok esetén a címeket hasonlítja össze. Tehát ha két változóhoz ugyanazt az értéket rendeljük és ezek az értékek benne vannak a -128 és 127 intervallumba,akkor a két változónak meg fog egyezni a címük,tehát egy összehasonlításkor megegyeznek majd. Azonban ha ezen az intervallumon kívül választunk számot akkor két új objektum fog létrejönni és egy összehasonlítás során nem fog megegyezni a címük.

Most pedig nézzünk két programot,ahol a feladatban megadott feltételt megadva egyik esetben végtelen ciklust,másik esetben pedig hamis feltételt fogunk kapni.

```
        class vegtelen{
    public static void main(String args[])
    {
        Integer t = 130;
        Integer x = 130;

        while(x <= t && x>=t && t != x)
            System.out.println("Vegtelen lesz");
    }

}
```

Látható,hogy ebben a kódban a t ls x értékéül egy olyan számot adtunk meg ami kívül esik az eltárolt intervallumon,ezáltal új objektuok jönnek létre,melyeknek nem ugyanaz lesz a címük.Így a while-ban lévő kifejezés mindenkor ugyanaz lesz,így egy végtelen ciklust kapunk.

```
        class igaz{
    public static void main(String args[])
    {
        Integer t = 100;
        Integer x = 100;

        while(x <= t && x>=t && t != x)
            System.out.println("hamis");
    }

}
```

Ebben a kódban például láthatjuk,hogy t és x értékeként egy olyan számot adtunk amely benne van a poolba,így a java,ugyanazt a címet fogja mindenkor változónak adni. Így a while ciklus feltétele hamis lesz,nem lép végtelen ciklusba.

12.4. Yoda

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

A programozásban a yoda conditions egy programozási stílus.Ezt összehasonlításkor használjuk.A megszokott feltétel úgy néz ki,hogy először a változót írjuk amit hasonlítani akarunk valamelyen konstanshoz azonban a yoda conditions esetében a konstansot fogjuk a kifejezés bal oldalára írni.A nevét a Star Warsban szereplő Yoda karakterről kapta,ugyanis ő sem a megszokott szintaxis szerint beszél a filmben. Ez a stílus az elírásokból következő hibákat próbálja orvosolni,ugyanis gyakran előfordul,hogy valamit összeakarunk hasonlítani azonban csak egy egyenlőségjelet írunk kettő helyett ami által felülírjuk a változó értékét.Hátránya azonban,hogy a kód olvasása nehezebb lesz.

Egy nem yoda szerint írt kifejezés:

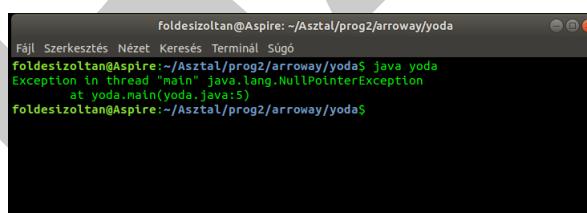
```
if (number == 14) {do something}  
//Itt látszik, hogy a változó van a kifejezés bal oldalára írva a ←  
konstans pedig a jobb oldalon
```

Ugyanez yoda szerint:

```
if (14 == number) {do something}  
//Látszik, hogy a konstans a bal oldalra került míg a változó a jobb ←  
oldalra.
```

A fenti példákból jól látható ha csak egy egyenlőségjelet írnánk a kifejezésbe, akkor yoda condition nélkül a number változóhoz 14 et rendelne a program, azonban ha yoda condition szerint írnánk egy egyenlőségjelet akkor hibát dobna a fordítás során. Tehát a yoda stíllussal kevesebb a hiba lehetőségeink. A yoda conditionnel elkerülhetőek a nem kívánatos null viselkedések is. Például ha egy string változónak null értéket adunk és ezt a stringet hasonlítjuk egy másik stringhez akkor nullpointerexceptiont okoz. Azonban ha yoda condition szerint először a stringet írjuk le és csak utána a változót akkor a kifejezés megfelelően fog működni. Erre írtunk egy programot ami egy nullpointerexceptiont dob.:

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



```
Foldesizoltan@Aspire:~/Asztal/prog2/arroway/yoda  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
Foldesizoltan@Aspire:~/Asztal/prog2/arroway/yoda$ java yoda  
Exception in thread "main" java.lang.NullPointerException  
at yoda.main(yoda.java:5)  
Foldesizoltan@Aspire:~/Asztal/prog2/arroway/yoda$
```

12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadysz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Ebben a feladatban a BBP algoritmust fogjuk megírni. A Bailey-Borwein-Plouffe vagy röviden BBP féle algoritmus 1995-ben lett felfedezve. Ezzel az algoritmussal a Pi egy általunk megadott számjegyétől tudunk további jegyeit számolni hexadecimális formában. Magát az algoritmust is egy gép fedezte fel.

A kód elején létrehozzuk a "d16PiHexaJegyek" változót amiben a hexadecimális jegyeket fogjuk tárolni. A kódban négy darab függvény fogja a számításokat végezni. Nézzük őket egyenként.

```

public PiBBP(int d) {

    double d16Pi = 0.0d;

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

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

    d16Pi = d16Pi - StrictMath.floor(d16Pi);

    StringBuffer sb = new StringBuffer();

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

    while(d16Pi != 0.0d) {

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

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

        d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
    }

    d16PiHexaJegyek = sb.toString();
}
}

```

Ez a függvény paraméterül kap egy integer-t, ettől az integer+1-től fogjuk számolni a jegyeket. Ezután négy double változót hozunk létre, ezek szükségesek a képlethez, ezeket majd a d16Sj függvény fogja kiszámolni. Majd következik maga a képlet. Ennek az eredménynek az egész részét fogjuk kapni a StrictMath.floor-nak kösönhetően, ugyanis ez egy nem egész számnak a nála kisebb egész részét fogja visszaadni. Ezekután megadjuk a hexajegyeket majd a kapott értékeket átváltjuk, és az sb.append függvény segítségével összefűzzük őket.

```

public 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 += StrictMath.pow(16.0d, d-k) / (double)(8*k + j);
}

```

```
* /  
  
    return d16Sj - StrictMath.floor(d16Sj);  
}
```

Ez a második függvény a programban, ez hívódik meg az előző függvényben és ez fogja számolni az S1 S4 S5 és S6 értékeket.A függvény két értéket kap paraméterül.A for ciklus után itt is történik egy másik függvényhívás, az n16modk függvény fog meghívódni.A függvény ennek az értékeknek is az egész részét fogja visszaadni.

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

Ez a függvény bináris hatványozást fog végezni.Ez $16^n \text{ mod } k$ -t fogja kiszámolni.Ezután a toString függvény fogja visszaadni a kiszámolt hexajegyeket.

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

Itt látható a main függvény ahol példányosítjuk a PiBBP osztályt.A 1 milliomodik számjegytől fogjuk kiíratni a Pi-t, vagyis az 1 millió+1-től fogja kiírni a hexadecimális számokat.

Eredmény:



```
foldesizoltan@Aspire:~/Asztal/prog2/arroway/kodfromscratch$ javac PtBBP.java
foldesizoltan@Aspire:~/Asztal/prog2/arroway/kodfromscratch$ java PtBBP
6C6SE530@foldesizoltan@Aspire:~/Asztal/prog2/arroway/kodfromscratch$
```

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

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

Ebben a feladatban egy java illetve egy C++ kódot fogunk írni,melyek megsértik a Liskov elvet. Először nézzük meg,hogy mi is az a Liskov elv. A Liskov elv egyike annak az öt objektumorientált alapelvnek. A SOLID mozaik szó jelzi az öt alapelv kezdőbetűit, ebből az L betű a Liskov elv. Liskov fogalma: Ha S altípusa T-nek, akkor egy T típusú objektum helyettesíthető S típussal anélkül, hogy megváltozna a program helyessége. Tehát, a Liskov elv kimondja,hogyha van egy classunk és annak van egy gyermeké,akkor az eredeti szülő osztályt tudnunk kell helyettesíteni a gyermek osztályával, anélkül,hogy ez bármi nem kívánatos viselkedést okozna a programunk működésében. Gyakori példának a téglalap és négyzet területének kiszámítását mutatják meg. Ugyebár a matematikából tudjuk,hogy a négyzet az egy speciális téglalap,melynek oldalai egyenlőek. Szóval két osztályunk lesz, az első a téglalap,melynek lesz 'a' és 'b' oldala,ezek különbözőek lesznek. És mivel a négyzet egy speciális téglalap,ezért következik belőle, hogy a másik négyzet osztály a téglalap osztály gyermeké lesz. Itt azonban átállítjuk az oldalakat,ugyanis egyenlőeknek kell lennie. Így már láthatjuk,hogy a Liskov elv nem fog működni,ugyanis ha megpróbáljuk a téglalap osztályt a négyzet osztállyal helyettesíteni akkor nem megfelelő eredményt fogunk kapni. A mi példánk a madarakra épül. Nézzük a kódot Java illetve C++ nyelven:

```
class Madar {  
public void repul() {};  
};  
  
class Prog {  
public void fgv ( Madar madar) {  
    madar.repul();  
}  
};  
  
class Sas extends Madar {};  
class Pingvin extends Madar {};
```

```
class Liskov{
    public static void main(String[] args)
    {
        Prog kod = new Prog();
        Madar mad = new Madar ();
        kod.fgv (mad);

        Sas sas = new Sas();
        kod.fgv (sas);

        Pingvin pingvin = new Pingvin();
        kod.fgv(pingvin);
    }
}
```

```
#include <iostream>

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 );
}
```

```
}
```

Amint látjuk itt is két osztályunk lesz. Az első a madár osztály,ez lesz a szülő. Ennek az osztálynak van egy repül függvénye, ebből következik, hogy minden madár tud majd repülni. Ezután létrehozunk egy gyermekosztályt, a sast. A sas megörökli a szülőtől a repül függvényt, de itt ugyebár nem lesz probléma mivel a sas tud repülni. Azonban ezután létrehozunk egy pingvin osztályt amely ugyanúgy megörökli a repül függvényt, azonban itt már problémába ütközünk, ugyanis így a program szerint a pingvin tud repülni, de ezt tudjuk, hogy nem igaz. Megoldás a jobb tervezés lenne, például lenne egy repülni képes és egy repülni nem képes osztályunk.

13.2. Szülő-gyerek

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

Ez a feladat azt mutatja meg, hogy egy származtatott osztály eléri az ősei függvényeit, de a szülő nem éri el a leszármazottjai függvényeit. Erre írtunk egy kódot javaban illetve C++-ban.

```
class Szulo
{
public void szul()
{
    System.out.println("Szulo vagyok");
}
};

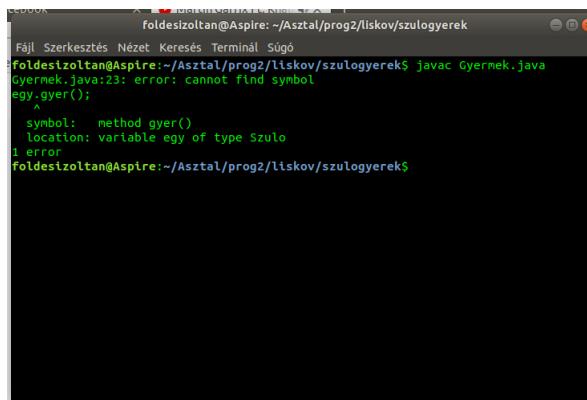
class Gyermek extends Szulo
{
public void gyer()
{
    System.out.println("Gyermek vagyok");
}

public static void main (String[] args)
{
Szulo egy = new Szulo();
Gyermek ketto = new Gyermek();
egy.szul();
ketto.gyer();
ketto.szul();
egy.gyer();

}
```

```
};
```

Két darab osztályunk van, a főosztály a Szulo, ennek lesz egy leszármazott osztálya a gyermek. Az extends utasítással jelezzük, hogy a Gyermek osztály a Szulo osztály leszármazottja. Mindkét osztálynak lesz egy-egy függvénye. Ugyebár az öröklődés szerint a leszármazott osztály örökli a szülő függvényeit, tehát majd megtudja hívni. Azonban a szülőosztály nem tudja használni a leszármazottjai függvényeit. A main függvényben minden két osztályt példányosítjuk, majd a gyermekkel meghívuk a szülő függvényét majd a szülővel hívjuk meg a gyermek függvényét. Ez a kód már a fordításnál hibát fog dobni.



```
Foldesizoltan@Aspire:~/Asztal/prog2/liskov/szulogyerek$ javac Gyermek.java
Gyermek.java:23: error: cannot find symbol
    egy.gyer();
           ^
symbol:   method gyer()
location: variable egy of type Szulo
1 error
Foldesizoltan@Aspire:~/Asztal/prog2/liskov/szulogyerek$
```

Kód C++-ban

```
#include <iostream>

using namespace std;

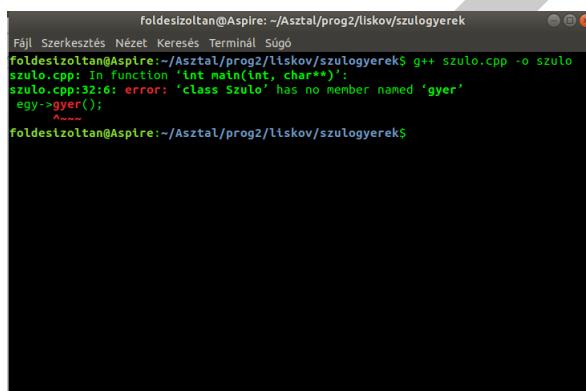
class Szulo
{
public:
    void szul()
    {
        cout<<"Szulo vagyok"=<endl;
    }
};

class gyermek: public Szulo
{
public:
    void gyer()
    {
        cout<<"gyerek vagyok"=<endl;
    }
};

int main(int argc, char **argv)
{
    Szulo* egy = new Szulo();
    gyermek* ketto = new gyermek();
```

```
egy->szul();  
kettő->gyer();  
kettő->szul();  
egy->gyer();  
}
```

Ez a kód is ugyanazt tudja mint a java, csak szintaktikai különbségek vannak. Itt `:`-al jelöljük, hogy a gyermek a szulo leszármazottja. Megint két függvényünk lesz, és a mainben megint példányosítjuk őket. Ez a kód ugyanúgy nem fog lefordulni.



A screenshot of a terminal window titled "földesizoltan@Aspire: ~/Asztal/prog2/liskov/szulogyerek". The terminal shows the command "g++ szulo.cpp -o szulo" being run. The output indicates an error: "szulo.cpp: In function 'int main(int, char**)': szulo.cpp:32:6: error: 'class Szulo' has no member named 'gyer'" followed by a cursor arrow pointing to the opening brace of the class definition. The terminal prompt "földesizoltan@Aspire: ~/Asztal/prog2/liskov/szulogyerek\$" is visible at the bottom.

13.3. Ciklomatikus komplexitás

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

Ebben a feladatban az első védési programnak, azaz a binfának fogjuk kiszámítani a ciklomatikus komplexitását. A ciklomatikus komplexitás egy szoftvermetrika. Egy konkrét számmal kifejezve fogja megadni a programunk komplexitását a forráskód alapján. Ez a számítás a gráfelméleten alapszik. A gráfot az adott függvényben lévő utasítások alkotják és akkor lesz köztük él ha egyből át tudunk lépni egyikről a másikra.

A képlet:

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

Ahol E a gráf éleinek száma, P a csúcsok száma a P pedig az összefüggő komponensek száma.

Ebben a feladatban nem kézzel fogjuk a komplexitást kiszámolni, hanem egy program segítségével. Én a lizard programot használtam, ezzel a programmal csak egy parancs kiadására volt szükség és már adta is az eredményt.

Eredmény:

```

fordesizoltan@Aspire-:~/Asztal/prog2/arroway/binfa\$ lizard binfa.cpp
=====
NLOC CCN token PARAM length location
=====
3 1 11 0 3 LZWBInfa::LZWBInfa@16-18gbInfa.cpp
5 1 23 0 5 LZWBInfa::LZWBInfa@19-23gbInfa.cpp
29 4 108 1 36 LZWBInfa::operator <@26-61@bInfa.cpp
5 1 20 1 7 LZWBInfa::kill@63-69gbInfa.cpp
5 1 26 2 5 LZWBInfa::operator <@78-82@bInfa.cpp
5 1 22 1 5 LZWBInfa::kill@83-87@bInfa.cpp
3 1 24 1 3 LZWBInfa:::Csomopont@94-96gbInfa.cpp
3 1 5 0 3 LZWBInfa:::Csomopont@-Csomopont@97-99gbInfa.cpp
4 1 9 0 4 LZWBInfa:::Csomopont@nullasgyernek@101-104@bInfa.cpp
4 1 9 0 4 LZWBInfa:::Csomopont@egyesGyernek@106-109gbInfa.cpp
4 1 12 1 4 LZWBInfa:::Csomopont@Jnullasgyernek@111-114@bInfa.cpp
4 1 12 1 4 LZWBInfa:::Csomopont@UegyesGyernek@116-119gbInfa.cpp
4 1 9 0 4 LZWBInfa:::Csomopont@getBetu@121-124@bInfa.cpp
13 3 94 2 15 LZWBInfa::kill@149-163@bInfa.cpp
9 2 37 1 11 LZWBInfa:::szabadit@164-174@bInfa.cpp
6 1 25 1 6 LZWBInfa:::getMelyseg@192-197@bInfa.cpp
7 1 36 1 7 LZWBInfa:::getAtlag@206-206@bInfa.cpp
12 2 68 1 15 LZWBInfa:::getSzoras@209-223@bInfa.cpp
12 3 52 1 13 LZWBInfa:::rmelyseg@226-238@bInfa.cpp
15 4 69 1 15 LZWBInfa:::ratlag@241-255@bInfa.cpp
15 4 81 1 15 LZWBInfa:::szoras@258-272@bInfa.cpp
4 1 18 1 4 usage@276-279@bInfa.cpp
59 13 334 2 95 main@282-376@bInfa.cpp
1 file analyzed.
=====
NLOC Avg.NLOC AvgCCN Avg.token function_cnt file
=====
271 10.0 2.2 48.0 23 binfa.cpp
=====
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or parameter_count > 100)
=====
```

Látható, hogy minden függvényt végig vizsgált a program és minden függvénynél a CCN oszlopban látható a komplexitás száma.

13.4. Anti OO

A BBP algoritmussal 4a Pi hexadecimális kifejtésének a 0. pozíciótól számított 106, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/hu/tartanitok-javat/apas03.html#id561066>

Ebben a feladatban az előző csokorban megírt BBP algoritmus futási idejét fogjuk összehasonlítani C, C++, C# és java nyelveken. A kód annyiban változott, hogy magát a képlet kiszámítását most a main függvényben végezzük. Hárrom eset lesz, amikor 10^6 , 10^7 és 10^8 -on jegyét fogjuk meghatározni a Pi-nek. Ehhez csak a for ciklusban lévő változót kell növelnünk., majd újrafordítani és futtatni. A források megtalálhatóak a mapámban, ide nem lesznek beillesztve

A teszteket egy acer laptopon végeztem, ami egy Intel Core i3-5005u processzorral illetve 4gb DDR3 rammal van felszerelve.

Táblázat:

	JAVA	C	C++	C#
10^6	2,885	3,238331	3,7902	2,919461
10^7	34,152	39,175441	44,7182	34,263677
10^8	386,423	449,211466	487,906	398,638318

A leggyorsabban nekem a Javas verzió futott, a leglassabban pedig a C++.

```

3.7992
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ gedit pi.cpp
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ ./pi
4.
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ g++ pi.cpp -o pi
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ ./pi
7
44.7182
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ gedit pi.cpp
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ g++ pi.cpp -o pi
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ ./pi
12
407.906
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ 
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ java PiBBP Bench
0
2.885
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ java PiBBP Bench
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ java PiBBP Bench
7
14.152
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ java PiBBP Bench
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ java PiBBP Bench
12
386.423
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ 

foldestizoltan@Aspire:~/Asztal/prog2/liskov$ mcs pi.cs
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ mono pi.exe
6
2.919461
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ mcs pi.cs
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ mono pi.exe
7
34.263677
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ mcs pi.cs
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ mono pi.exe
12
398.638318
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ 

foldestizoltan@Aspire:~/Asztal/prog2/liskov$ java PiBBP Bench
0
3.238331
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ gedit pi_bb_bench.c
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ gcc pi_bb_bench.c -o pi_bb_bench
pi_bb_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit]
main ()
^~~~~
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ ./pi_bb
7
39.175441
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ gedit pi_bb_bench.c
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ gcc pi_bb_bench.c -o pi_bb
pi_bb_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit]
main ()
^~~~~
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ ./pi_bb
12
449.211466
foldestizoltan@Aspire:~/Asztal/prog2/liskov$ 

```

13.5. Hello, Android!

Élesszük fel az SMNIST forHumans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNIST> módosításokat eszközölj benne, pl. színvilág.

Ebben a feladatban az SMNIST for Humans androidos projektet élesztettük fel illetve végeztünk rajta egy kis változtatást. Racs Tamás UDPORG-ba kirakott forrásai alapján tudtam a programot beimportálni az android studióba. A hibakereséshez e telefonomat használtam, ehhez USB kábellel hozzá kellett kötni a géphez a telefont, és annak fejlesztői beállításaiban engedélyezni kellett az USB-s hibakeresést, majd a hivatalos weboldaláról kellett letölteni egy USB drivert. Ezután már a telefonon jelent meg az alkalmazás. Négy darab forrásfájl található a forrásokban. Nekünk a program színvilágát kellett megváltoztatni, ezt az SMNISTSurfaceView.java fájlból tehetjük meg.

Először a bgColor függvényben írtuk át az rgb értékeit. Ez a függvény állítja a háttér színeit. Két fajta szín villok, én az egyiket zöldre, a másikat pirosra állítottam.

```

int [] bgColor =
{
    android.graphics.Color.rgb(0, 100, 0),
    android.graphics.Color.rgb(100, 0, 0)
};
```

Ezután a dotPaint.setColor függvényénél írtam át a színt, jelen esetben zöldre, ez a kis pöttyök színéét változtatja meg.

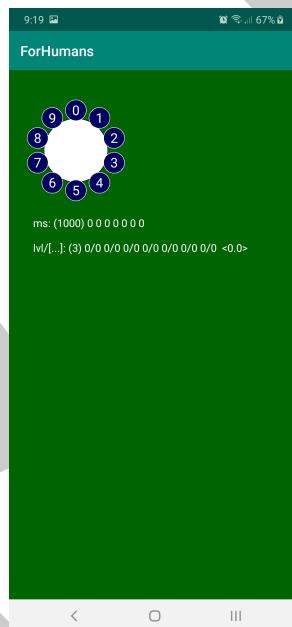
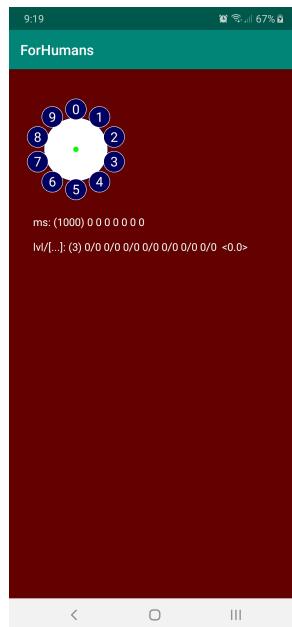
```
dotPaint.setColor(Color.GREEN);
```

Majd a borderPaint.setColor-nál állítottam át a színt fehérre, amely a kis kör színét változtatja meg.

```
borderPaint.setColor(Color.WHITE);
```

Majd a végén a fillPaint.setColor függvényénél a színt kékre állítottam, ez a számok mögötti színt állítja.

```
fillPaint.setColor(android.graphics.Color.rgb(0, 0, 100));
```



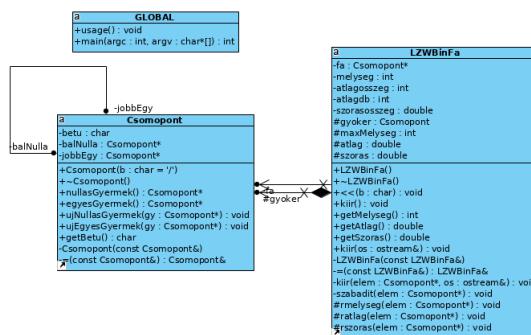
14. fejezet

Helló, Mandelbrot!

14.1. Reverse engineering UML osztálydiagram

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

Ebben a feladatban az első védési programhoz,azaz a binfához készítettünk egy osztálydiagramot. Azt osztálydiagram készítéséhez a visual paradigm programot használtam. A program a forráskód importálása után generálta le az osztálydiagramot.Az osztálydiagram az objektumok típusait és a köztük lévő statikus kapcsolatokat írja le. Ezen kívül ezek az osztályok mutatják az osztályok tulajdonságait, illetve műveleteiket.



Az osztálydiagrammal osztályokat tudunk ábrázolni.Az osztályokat téglalapok fognak jelezni,melynek fejlécében az osztály neve szerepel,alatta az osztály attribútumai,az alatt pedig az osztály műveletei. A + - # ~ szimbólumok jelölik a láthatóságot.A + a nyilvános a - a privát a # a védett a ~ a csomagszintű látathatóságot jelöli.Ezen kívül látjuk még a műveletek visszaadási típusát.Asszociáció alatt az osztályok közötti kétirányú összelöttetést értjük.Ez magyarul azt jelenti h az egyik osztály használja a másik osztály valamely részét(egy metódust pl.).Aggregáció alatt azt értjük,hogy az egyik osztály tartalmazza vagy birtokolja a másikat,ezt egy üres rombuszal jelöljük.A kompozíció az egy erős aggregáció azt jelenti a tartalmazó a tartalmazott nélkül nem létezhet.Ennek a jele a kitöltött rombusz, a kódunkban jelenleg ez a gyoker elem.

14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Ebben a feladatban az első feladat ellentétjét fogjuk megtenni. Egy UML diagrammból fogunk forráskódot generálni. Erre is alkalmas a visual paradigm program. Én az előzőleg létrehozott UML diagramból generáltattam kódot. Az eredmény a következő lett:

```
class LZWBInFa {  
  
private:  
    Csomopont* fa;  
    int melyseg;  
    int atlagosszeg;  
    int atlagdb;  
    double szorasosszeg;  
protected:  
    Csomopont gyoker;  
    int maxMelyseg;  
    double atlag;  
    double szoras;  
  
public:  
    LZWBInFa();  
  
    void ~LZWBInFa();  
  
    void kiir();  
  
    int getMelyseg();  
  
    double getAtlag();  
  
    double getSzoras();  
  
    void kiir(std::ostream& os);  
  
private:  
    LZWBInFa(const LZWBInFa& unnamed_1);  
  
    void kiir(Csomopont* elem, std::ostream& os);  
  
    void szabadit(Csomopont* elem);  
  
protected:  
    void rmelyseg(Csomopont* elem);  
  
    void ratlag(Csomopont* elem);  
  
    void rszoras(Csomopont* elem);  
};
```

```
class Csomopont {  
  
private:  
    char betu;  
    Csomopont* balNulla;  
    Csomopont* jobbEgy;  
  
public:  
    Csomopont(char b = '/');  
  
    void ~Csomopont();  
  
    Csomopont* nullasGyermek();  
  
    Csomopont* egyesGyermek();  
  
    void ujNullasGyermek(Csomopont* gy);  
  
    void ujEgyesGyermek(Csomopont* gy);  
  
    char getBetu();  
  
private:  
    Csomopont(const Csomopont& unnamed_1);  
};
```

Látható,hogy a program vázlatát szépen visszaadja.Azonban ezek csak a header file-ok.Csodákra nem képes a generálás, a különböző függvényeket nekünk kell deklarálnunk.A generáló program nem tudhatja,hogy mi mit szeretnénk az adott függvényben végezni.Azonban sok gépelést meglehet egy ilyen generálással spárni.Ha van egy jó ötletünk és szeretnénk gyorsan egy program vázlatot kapni,ez egy nagyon jó megoldás erre.

14.3. Egy esettan

A BME-s C++ tankönyv 14. fejezetét (427-444 elmélet, 445-469 az esettan) dolgozzuk fel!

Ebben a feladatban az előző félévben olvasott BME-s C++ könyv 14.fejezetében található feladatot fogjuk átnézni.A feladat az,hogy egy kereskedésnek(ami számítógépes alkatrészek és konfigok eladásával foglalkozik) kell egy olyan alkalmazást elkészíteni,amely ezen alkatrészek nyilvántartására megfelelő lesz.Támogatnia kell az alkalmazásnak a termékek fjlból való beolvasását illetve képenyőre és fájlba való kiíratást, illetve az árképzés rugalmas kialakítását.Mindezt,úgy kell megtennünk,hogy könnyen bővíthető legyen a nyilvántartás. Szükség lesz egy keretrendszerre amely a termékcsaládokat fogja kezelni,ennek a követelményei a következők:a forráskód kiadása nélkül működjön,fájlból való beolvasást illetve kiíratást tudja, illetve a

kijelzőre írasson ki. A termékek attribútumai a következők:Bezserzési ár,bezserzés dátuma,név illetve típus. Ezenfelül az aktuális ár mely a többi attribútumból számolható.És végül egy termék lehet elemi vagy összetett, az összetett azt jelenti,hogy több termékből áll,ilyen pl egy számítógép konfiguráció.

```
#ifndef PRODUCT_H
#define PRODUCT_H

#include <iostream>
#include <ctime>

class Product
{
protected:
    int initialPrice;
    time_t dateOfAcquisition;
    std::string name;
    virtual void printParams(std::ostream& os) const;
    virtual void loadParamsFromStream(std::istream& is);
    virtual void writeParamsToStream(std::ostream& os) const;

public:
    Product();
    Product(std::string name, int initialPrice, time_t dateOfAcquisition);
    virtual ~Product() {};
    int getInitialPrice() const;
    std::string getName() const;
    time_t getDateOfAcquisition() const;
    int getAge() const;
    virtual int getCurrentPrice() const;
    void print(std::ostream& os) const;
    virtual std::string getType() const = 0;
    virtual char getCharCode() const = 0;
    friend std::istream& operator>>(std::istream& is, Product& product);
    friend std::ostream& operator<<(std::ostream& os, Product& product);
};

#endif // PRODUCT_H
```

A termékek reprezentálására szolgál a Product osztály.Ebből az osztályból fogjuk származtatni a specifikus termékeket reprezentáló osztályokat(pl. HardDisk vagy Display.).Ennek az osztálynak a tagváltozói védettek, azaz a külvilágnak csak olvashatóak.Ezeket csak Getterekkel érhetjük majd el.A GetCurrentPrice fogja kiszámolni az aktuális árat, ami ugyebár termékfüggő,ezért ezt virtuálisan definiáljuk,ugyanis majd a leszármazott osztályok felül fogják írni.Ebben az osztályban a kiindulási árat fogja visszaadni,tehát a további osztályok ezt fogják megörökölni.Található még két virtuális függvény az osztályban,ezek a GetType illetve a GetCharCode, azaz típus,illetve típuskód.Ezek is termékfüggők,tehát majd feliül lesznek definiálva.A Print függvény fogja a termékeket kiíratni.De viszont a paraméterek kiírásához szükségünk van még egy virtuális függvényre,ez lesz a printParams.

Minden termék kapott egy termékkódt,pl a h a HardDisk terméket jelöli.Ez a beolvásásnál segítség, beol-

vasás során egy sor jelöl egy terméket, és ez a típuskód lesz az első karakter, ezután jön a név, beszerzési ár, beszerzés dátuma illetve a paraméterek. A fájlbairásról a writeParamsToStream függvény felelős. Ez is virtuális függvény lesz, tehát felül lesz definiálva. A beolvásás már összetettebb feladat, soronként kell beolvasni az első karaktert, ezt megtehetjük a Product osztályban, azonban a többi paraméter termékfüggő lesz, szóval ezt minden alosztályban felül kell definiálni.

```
#ifndef COMPOSITEPRODUCT_H
#define COMPOSITEPRODUCT_H

#include <vector>
#include <iostream>

#include "product.h"

class CompositeProduct: public Product
{
    std::vector<Product*> parts;
protected:
    void printParams(std::ostream& os) const;
    void loadParamsFromStream(std::istream& is);
    void writeParamsToStream(std::ostream& os) const;
public:
    CompositeProduct();
    ~CompositeProduct();
    void addPart(Product* product);
};

#endif // COMPOSITEPRODUCT_H
```

Az összetett termékekhez vezetjük a a CompositeProduct osztályt, ami a Product gyermeke lesz. Egy tagváltózóban fogjuk tárolni a tartalmazott termékekre mutató mutatókat. A kiírás során a c termékkód fogja jelölni, hogy összetett termékről van szó, és legvégén lesz egy szám ami a benne lévő termékek számát jelöli, innen tudjuk, hogy az alatta lévő x darab termék tartozik bele.

```
#ifndef PRODUCTINVENTORY_H
#define PRODUCTINVENTORY_H

#include <vector>
#include <iostream>
#include "product.h"

class ProductInventory
{
    void emptyProducts();
protected:
    std::vector<Product*> products;
public:
    virtual ~ProductInventory();
```

```

void readInventory(std::istream& is);
void writeInventory(std::ostream& os) const;
void printProducts(std::ostream& os) const;
void addProduct(Product* product);

};

#endif // PRODUCTINVENTORY_H

```

A ProductInventory osztály fogja betölteni a fájlból a termékeket, majd a memóriában eltárolni azokat, valamint fájlba írni illetve kiíratni őket. Ezt az osztályt a keretrendszerben definiáltuk és minden termékre Product mutatóként fog hivatkozni. Az AddProduct fogja felvenni a paraméterként kapott terméket a terméklistára. A termékeket létrehozni csakis virtuális függvényvel tudjuk, ugyanis mindegyik terméket a saját osztálynál belül tudunk létrehozni.

Kiíratás:

```

Fájl Szerkesztés Nézet Keresés Terminal Súgó
Földesizoltan@Aspire:~/Asztal/prog2/mandel$ ./esettan
Test1: create inventory and printing it to the screen.
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20191008, Age: 0, Current price: 30000, InchWidth: 13, InchHeight: 12
1.: Type: HardDisk, Name: WD, Initial price: 25000, Date of acquisition: 20191008, Age: 0, Current price: 25000, SpeedRPM: 7500
Press any key to continue...

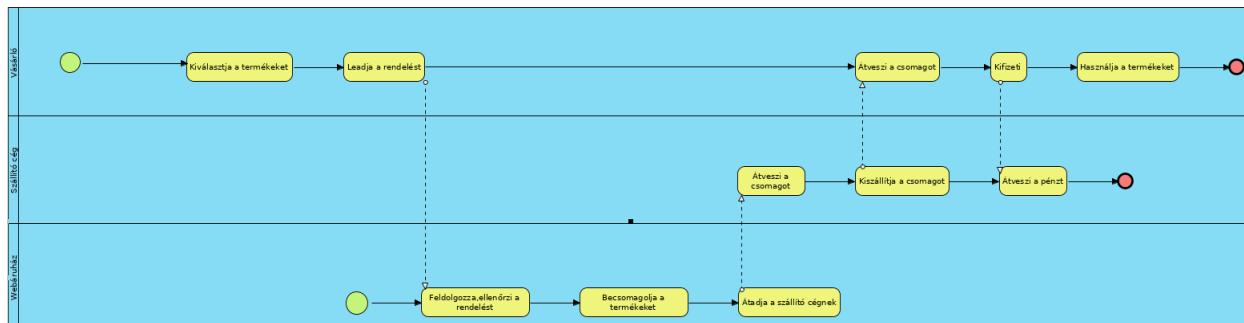
Test2: loading inventory from a file (computerproducts.txt), printing it, and then writing it to a file (computerproducts_out.txt).
End of reading product items.The content of the file is:
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20011001, Age: 6580, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 4755, Current price: 28000, InchWidth: 10, InchHeight: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age: 4755, Current price: 7000
Items:
0. Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20011001, Age: 6580, Current price: 24000, InchWidth: 12, InchHeight: 13
1. Type: HardDisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 4755, Current price: 28000, SpeedRPM: 7000
2.: Type: HardDisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20050228, Age: 5334, Current price: 20000, SpeedRPM: 7000
The content of the inventory has been written to computerproducts_out.txt
Done.

```

14.4. BPMN

Rajzolunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog47.folia>)

Ebben a feladatban egy BPMN diagramot készítettünk el. A Business Process Model and Notation vagy röviden BPMN diagrammal üzleti folyamatokat tudunk ábrázolni, mint folyamatábra. A feladat elvégzésáhez most is a visual paradigm programot használtam.



Én egy webáruházi rendelés menetét ábrázoltam. Először is a zöld pontok jelzik a folyamat kezdetét. A vásárló kiválasztja majd leadja a rendelést. Ezt követően kezdődnek meg az áruház tevékenységei, feldolgozza, ellenőrzi, a rendelést, majd átadja a szállítócégnek. A szállítócég átveszi a csomagot, majd kiszállítja. Ez után a vevő átveszi a csomagot majd kifizeti azt. A futár átveszi a pénzt, majd a vásárló használja a termékeit. A piros pontok jelzik a tevékenységek végeit.

14.5. TeX UML

Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat).

Ebben a feladatban latexben kellett UML diagrammot rajzolatnunk. A latex legfőképp elektronikus dokumentumok, szakdolgozatok és tudományos cikkek írására.

Én egy online szerkesztőt használtam az overleaf-t. Az UML diagrammok rajzolásához a tikz-uml csomagot használtam. A megszokott módon fogjuk jelölni a függvények láthatóságát. Az OOCWC projektről kellett csinálni a diagramokat. Nézzük a szintaxist.

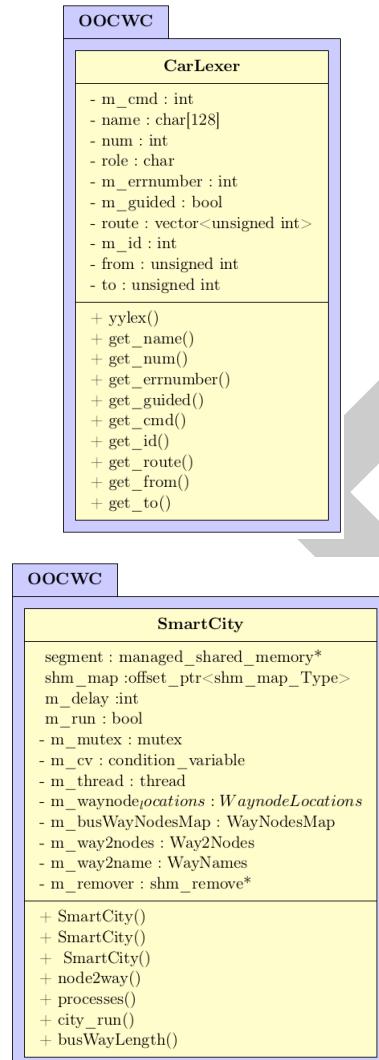
```
\begin{umlpackage} [x=0, y=0] {OOCWC}
```

Ezzel a parancssal kezdjük el az UML diagramunkat. Az x és y koordinátákkal megadjuk a kezdeti pozíciót. Azután pedig a címét a diagramnak.

```
\umlclass [x=0, y=0] {CarLexer} {
    - m\_\cmd: int \\
    - name: char[128] \\
    - num: int \\
    - role: char \\
    - m\_\errnumber: int \\
    - m\_\guided: bool \\
    - route: vector<unsigned int> \\
    - m\_\id: int \\
    - from: unsigned int \\
    - to: unsigned int
}
+ yylex() \\
+ get\_\name() \\
+ get\_\num() \\
+ get\_\errnumber() \\
+ get\_\guided() \\
+ get\_\cmd() \\
+ get\_\id() \\
+ get\_\route() \\
+ get\_\from() \\
+ get\_\to()
}
```

Ezzel kezdjük a tényleges osztálydiagramot, jelen esetben a CarLexer osztályét. Ezután kapcsos zárójelek közé adhatjuk meg az attribútumokat, majd még egy páros zárójel közé a függvényeket. minden sor

végére \\\ kell tenni, ezzel jelezzük, hogy ott van vége. Ezután az end szócskával be is kell zárnunk az uml-packagest.



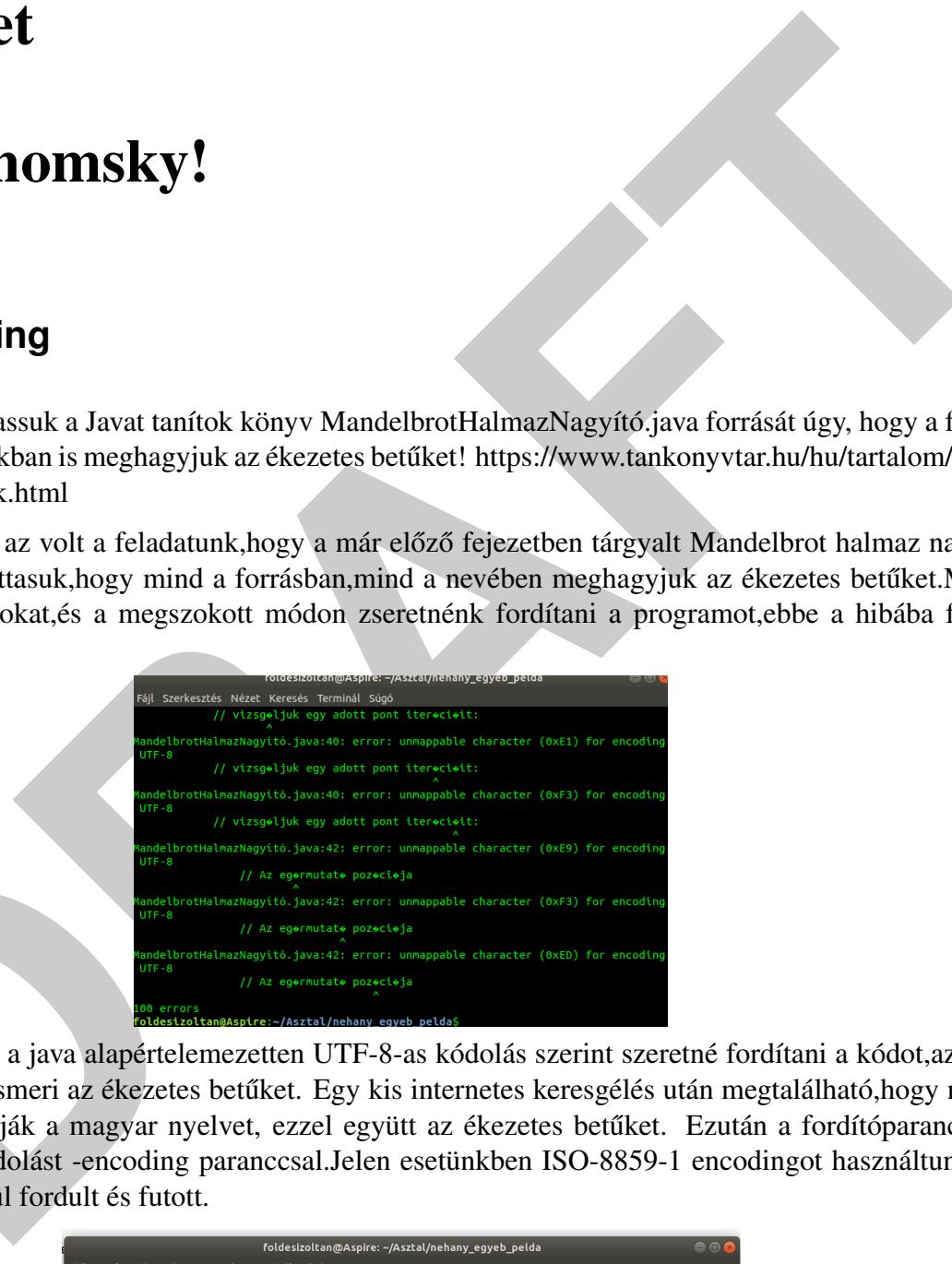
15. fejezet

Helló, Chomsky!

15.1. Encoding

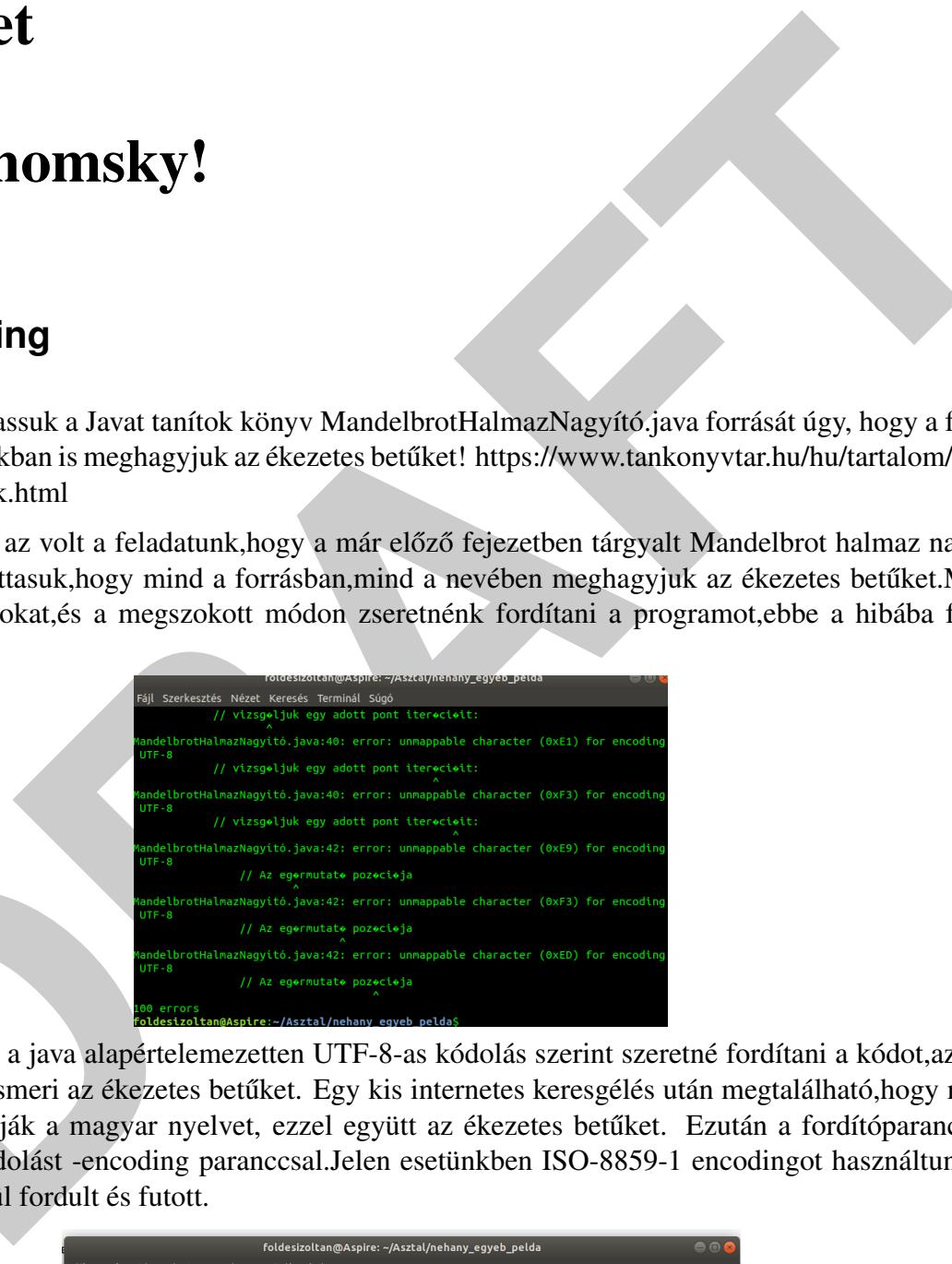
Fordítsuk le és futassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Ebben a feladtból, az volt a feladatunk, hogy a már előző fejezetben tárgyalt Mandelbrot halmaz nagyítós alkalmazást, úgy futtasuk, hogy mind a forrásban, mind a nevében meghagyjuk az ékezes betűket. Miután letöltöttük a forrásokat, és a megszokott módon szeretnénk fordítani a programot, ebbé a hibába fogunk ütközni:

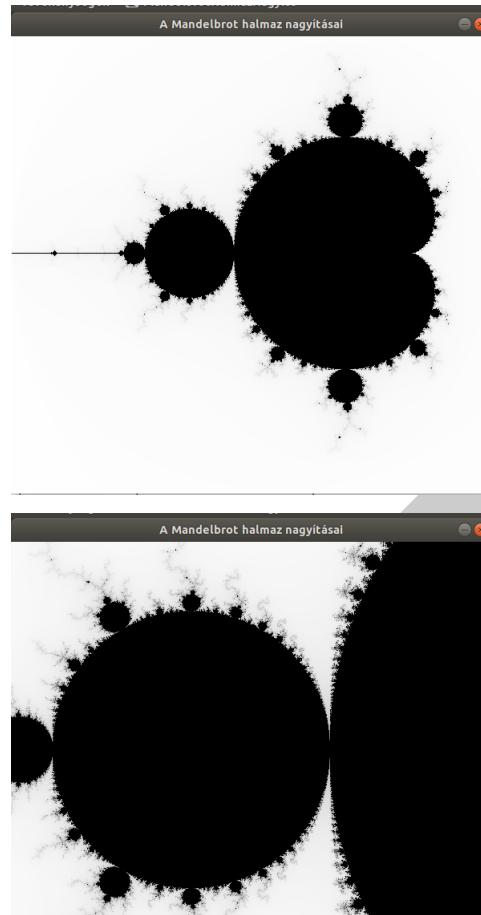


```
foldesizoltan@Aspire: ~/Asztal/nehany_egyeb_pelda
Fájl Szerkesztés Nézet Keresés Terminál Súgó
        // vizsgáljuk egy adott pont iterációt:
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
UTF-8
        // vizsgáljuk egy adott pont iterációt:
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
UTF-8
        // vizsgáljuk egy adott pont iterációt:
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
UTF-8
        // Az egérmutató pozíciója
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
UTF-8
        // Az egérmutató pozíciója
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
UTF-8
        // Az egérmutató pozíciója
100 errors
foldesizoltan@Aspire:~/Asztal/nehany_egyeb_pelda
```

Ennek oka az, hogy a java alapértelmezetten UTF-8-as kódolás szerint szeretné fordítani a kódot, azonban ez a kódolás nem ismeri az ékezes betűket. Egy kis internetes keresgélés után megtalálható, hogy melyik kódolások támogatják a magyar nyelvet, ezzel együtt az ékezes betűket. Ezután a fordítóparancsukba kell beírnunk a kódolást -encoding parancssal. Jelen esetünkben ISO-8859-1 encodingot használtunk és a program hiba nélkül fordult és futott.



```
foldesizoltan@Aspire: ~/Asztal/nehany_egyeb_pelda
Fájl Szerkesztés Nézet Keresés Terminál Súgó
foldesizoltan@Aspire:~/Asztal/nehany_egyeb_pelda$ javac -encoding "ISO-8859-1" MandelbrotHalmazNagyító.java
foldesizoltan@Aspire:~/Asztal/nehany_egyeb_pelda$
```



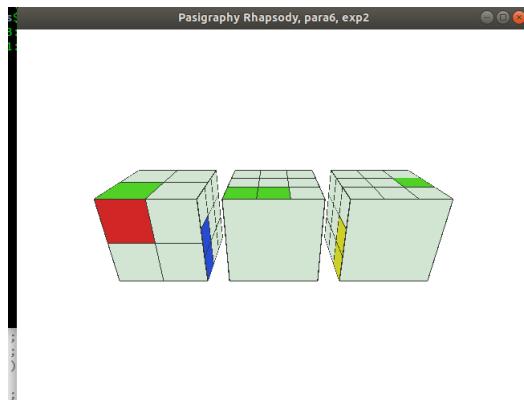
15.2. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Ebben a feladatban a Paszigráfia Rapszódia OpenGL-es megvalósításával foglalkozunk. Ez a Paszigráfia, vagy röviden PaRa egy mesterséges nyelv amely az esport kultúra nyelve. A mi feladatunk, hogy a színvilágot vagy az irányítást megváltoztassuk. Ahhoz, hogy a programunk lefusson, szükség lesz a libboost illetve a freeglut3 telepítése. Ezután ha fordítjuk és futtatjuk a programot ez az ablak nyílik meg:

fordítás, futtatás:

```
Foldesizoltan@Aspire:~/Asztal/docs$ g++ para6.cpp -lboost_system -lGL -lGLU -lglut
Foldesizoltan@Aspire:~/Asztal/docs$ ./para
Foldesizoltan@Aspire:~/Asztal/docs$
```



Három darab kockát látunk, számokkal tudunk köztük váltani, illetve a nyilakkal tudjuk forgatni őket, valamint a + és - jelekkel nagyítani, illetve kicsinyíteni tudunk. Lássuk a változtatásokat. A színeket a

```
glColor3f ( 0.0f, 0.44f, 0.21f );
```

sorok fogják változtatni, az első ilyen sor a 3 kocka alapszínét fogja változtatni. A

```
glBegin ( GL_LINES );
```

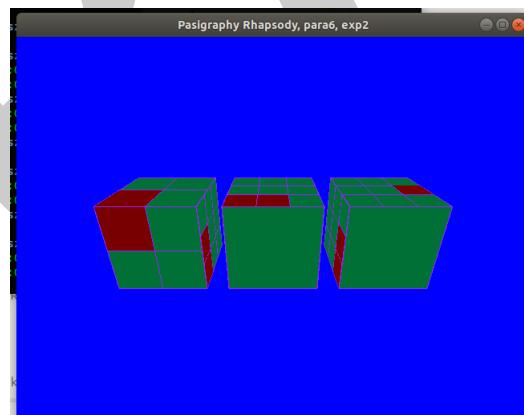
sort követő glColor3f -ek fogják állítani a kockák vonalait. A

```
glBegin ( GL_QUADS );
```

sort követők pedig a kis négyzetek színeit fogják állítani. A

```
glClearColor ( 0.0f, 0.0f, 1.0f, 1.0f );
```

fogja állítani a háttérszínt. Ezek után ilyen kockákat kaptam:



15.3. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás.

Ebben a feladatban az előző feladathoz hsonaló kockákat fogunk átszínezgetni. Csak ezt most Luatex-ben fogjuk megtenni. Lesz egy prelpara.lua fájlunk, amiben a színeket fogjuk megváltoztatni, és lesz egy tex fájlunk amiből generálni fogjuk a képet. Én megint csak az online overleaf oldalt használtam, ahova feltöltöttem a megadott mappa tartalmát, és a fordítót átállítottam luateX-re. Ezután már lehetett a színekkel szórakozni.

Mint már írtam, a színeket a prelpara.lua fájlban tudjuk állítani.

A

```
\\" shade[shifting, left color=black!100, right color=black!100
```

soroknál a color-nál kell megadni a színeket,én a fekete és zöld színekkel variáltam.A mögötte lévő számmal a szín intenzitását adhatjuk meg. Így a kockák eleje illetve oldala zöld lett,a teteje pedig fekete.

A

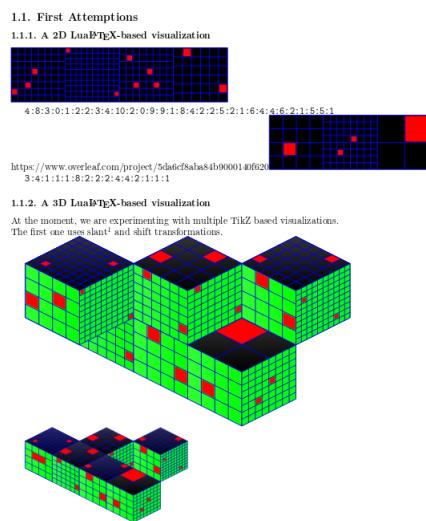
```
"\\node[rightslant, fill=red, anchor=south west, inner sep=0pt,minimum ←
size=\\N{"n,"}cm-\\pgflinewidth] at \\D{"n,"}{",x,"}{",y,"+.5*x,x,"} ←
{};"
```

sorok fogják létrehozni a kis négyzeteket,itt a fill parancs után adhatjuk meg,hogy milyen színnel szeretnénk kitölteni őket,jelen esetben én pirosra színeztem őket.

A

```
\\draw[topslant, step=\\N{"n,"}cm, blue] (0,0) grid (1,1);"
```

sorok fogják rajzolni a kis négyzetek közötti vonalakat,ezeket a cm után adhatjuk meg jelen esetben kékre lettek állítva.Így a következő kockákat kaptuk:



15.4. Full screen

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

Ebben a feladatban egy teljesképernyős java programot kellett írni.Én Bátfai Norbert tanár úr által belinkel labirintusos játékot élesztettem fel.A források letöltése és fordítás után probléma nélkül futott a játék.A teljes képernyős futás a LabirintusJáték.java fájlban található,nézzük meg ezt a kód részletet.

```
public void teljesKépernyősMód(java.awt.GraphicsDevice graphicsDevice ←
) {
```

```
int szélesség = 0;
int magasság = 0;

setUndecorated(true);

setIgnoreRepaint(true);

setResizable(false);

boolean fullScreenTamogatott = graphicsDevice.isFullScreenSupported ↵
();

if(fullScreenTamogatott) {
    graphicsDevice.setFullScreenWindow(this);

    java.awt.DisplayMode displayMode
        = graphicsDevice.getDisplayMode();

    szélesség = displayMode.getWidth();
    magasság = displayMode.getHeight();
    int színMélység = displayMode.getBitDepth();
    int frissítésiFrekvencia = displayMode.getRefreshRate();
    System.out.println(szélesség
        + "x" + magasság
        + ", " + színMélység
        + ", " + frissítésiFrekvencia);

    java.awt.DisplayMode[] displayModes
        = graphicsDevice.getDisplayModes();

    boolean dm1024x768 = false;
    for(int i=0; i<displayModes.length; ++i) {
        if(displayModes[i].getWidth() == 1366
            && displayModes[i].getHeight() == 768
            && displayModes[i].getBitDepth() == színMélység
            && displayModes[i].getRefreshRate()
            == frissítésiFrekvencia) {
            graphicsDevice.setDisplayMode(displayModes[i]);
            dm1024x768 = true;
            break;
    }
}

if(!dm1024x768)
    System.out.println("Nem megy az 1024x768, de a példa ↵
        képméretei ehhez a felbontáshoz vannak állítva.");

} else {
    setSize(szélesség, magasság);
```

```
    validate();
    setVisible(true);
}

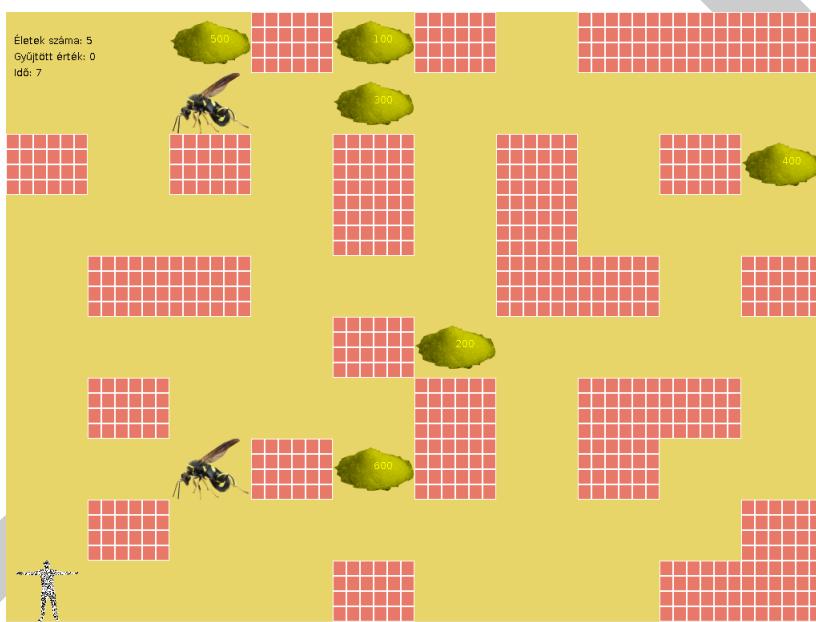
createBufferStrategy(2);

bufferStrategy = getBufferStrategy();

}
```

Ez a függvény a felelős a teljesképernyős módért. Először létrehozunk két változót a kijelző szélességének illetve magasságának. Majd az isFullScreenSupported függvénnyel megnézzük, hogy a kijelző támogatja-e a teljesképernyősmódot. Majd ezekután lekérjük a kijelzőnk szélességét, magasságát, és a frissítési frekvenciáját. Ezeket az adatokat ki is íratjuk. Ezután megnézzük, hogy a kijelző támogatja-e az 1024*768-as felbontást, mivel ilyen felbontásúak a képeink. Ezt azonban átírtam a laptopom kijelzőjének 1366x768-as felbontására, így elérve, hogy teljesképernyőben fussen.

futás közben:



15.5. Perceptron osztály

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

Ebben a feladatban a perceptron osztályt fogjuk használni. A feladatban az előző félévben elmagyarázott mandelbrot program által generált mandel képet fogjuk megadni bemenetnek a perceptronnak. A perceptron egy gráf alapú modell, melyben mesterséges neuronok kommunikálnak rétegekbe rendezve. A neurális hálók általában háromfajta réteggel rendelkeznek. A bemeneti réteg módosítatlanul továbbítja a bemenetet a háló többi részének. Ezután a rejtett rétegek kódolják a az információt. Majd a kimeneti réteg adja az eredményt.

```
#include <iostream>
#include "mlp.hpp"
```

```
#include "png++/png.hpp"
//g++ mlp.hpp main.cpp -o perc -lpng -std=c++11

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

    Perceptron* p = new Perceptron(3, size, 256, 1);

    double* image = new double[size];

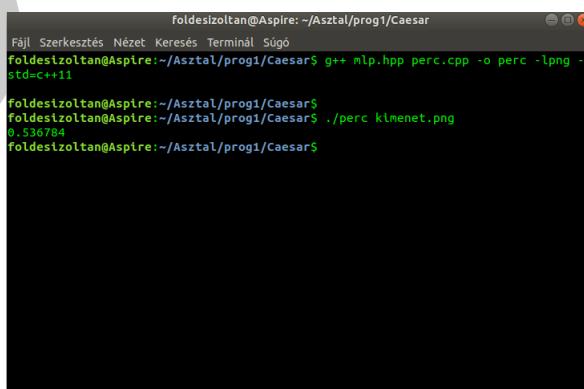
    for(int i {0}; i<png_image.get_width(); ++i)
        for(int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width() + j] = png_image[i][j].red;

    double value = (*p)(image);

    std::cout << value << std::endl;

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

Először is szükségünk van az mlp.hpp fájlra, ez fogja tartalmazni magát a Perceptron osztályt. Legelőször is beolvassuk a bemeneti képet, majd kiszámoljuk a méretét, úgy, hogy összeszorozzuk a magasságát a szélességgel, ezt a size változóban fogjuk elmenteni. Majd ezután példánypsítjuk a Perceptron osztályt. Egy háromrétegű neurális hálót hozunk létre, melynek első rétegének size db neuronja lesz, a másodiknak 256 és egy szál lesz az eredmény. Majd két for ciklussal betöljük a képet az image változóba, vagyis a képnek is a red komponensét. Majd a p objektumot függvényként használjuk és meghívjuk az image-re. Ezután kapunk egy számot eredményként. Majd a legvégén felszabadítjuk a memóriát.



16. fejezet

Helló, Stroustrup!

16.1. JDK osztályok

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

Ebben a feladtból, a jdk forrásból fogjuk kiíratni az osztályokat, illetve összefogjuk számolni, hogy összesen hány osztály van. Ehhez a Bátfai Norbert tanár úr fenykard.cpp kódjából a read_acts függvényt fogjuk használni, egy kis módosítással. Nézzük is a kódot:

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

#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>
#include <boost/program_options.hpp>
#include <boost/tokenizer.hpp>
#include <boost/date_time posix_time posix_time.hpp>

using namespace std;
using namespace boost::filesystem;

int osztalyok = 0;

void read_acts ( boost::filesystem::path path, std::vector<std::string> ←
acts )

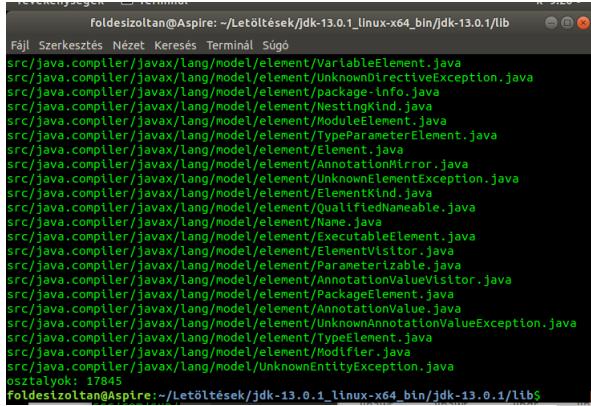
{
    if ( is_regular_file ( path ) ) {

        std::string ext ( ".java" );
    }
}
```

```
if ( !ext.compare ( boost::filesystem::extension ( path ) ) ) {  
  
    cout<<path.string()<<endl;  
  
    std::string actpropspath = path.string();  
  
    std::size_t end = actpropspath.find_last_of ( "/" ) ;  
  
    std::string act = actpropspath.substr ( 0, end );  
  
    acts.push_back(act);  
  
    osztalyok++;  
}  
  
} else if ( is_directory ( path ) )  
  
for ( boost::filesystem::directory_entry & entry :  
  
boost::filesystem::directory_iterator ( path ) )  
  
read_acts ( entry.path(), acts );  
  
}  
  
int main ( int argc, char *argv[] )  
  
{  
  
    string path="src";  
  
    vector<string> acts;  
  
    read_acts(path,acts);  
  
    cout<<"osztalyok: "<<osztalyok<< std::endl;  
}
```

Először is létrehoztunk egy globális változót,ezt fogjuk majd növelni a ha találunk egy osztályt. A read_acts függvény kapni fog egy path stringet,ami a kiinduló könyvtár lesz,illetve kap majd egy vektort amibe a fájlokat fogjuk eltárolni. A függvény először megnézni, hogy az adott vizsgált elem fájl-e vagy sem. Ha fájl akkor megnézi,hogy a kiterjesztése .java vagy sem,ugyanis akkor lesz osztály ha java a kiterjesztése.Ezután ha ezek a feltételek teljesülnek akkor ennek az elérési útját ki is íratjuk.Majd a számlálónkat növeljük.Azonban ha az adott vizsgált elem nem fájl,akkor megnízzük hogy mappa-e, ha igen akkor rá is meghívódik a read_acts függvény.A végén a main függvényben pedig létrehozzuk a változókat,majd kiírjuk a számlálónkat.

futtatás:



A screenshot of a terminal window titled "Fájl Szerkesztés Nézet Keresés Terminál Súgó". The window displays a list of Java source files from the "javac.lang.model.element" package. The files listed include VariableElement.java, UnknownDirectiveException.java, PackageInfo.java, NestingKind.java, ModuleElement.java, TypeParameterElement.java, Element.java, AnnotationMirror.java, UnknownElementException.java, ElementKind.java, QualifiedNameable.java, Name.java, ExecutableElement.java, ElementVisitor.java, Parameterizable.java, AnnotationValueVisitor.java, PackageElement.java, AnnotationValue.java, UnknownAnnotationValueException.java, typeElement.java, Modifier.java, and UnknownEntityException.java. The command "osztalyok: 17845" is shown at the bottom, followed by the prompt "Földesizoltan@Aspire:~/Letöltések/jdk-13.0.1_linux-x64_bin/jdk-13.0.1/lib\$".

16.2. RSA és összefoglalás

Ha valamilyen bizalmas információt egy nem biztonságos környezetben kell tárolnunk, akkor biztosítanunk kell annak védelmét. Erre a védelemre szolgálnak a különböző kriptográfiai, azaz kódolási módszerek. Ezekkel a módszerekkel titkosíthatjuk, vagy hitelesíthetjük az adatokat. Mindkét esetben az adatokat először kódolnunk kell, ha pedig ismét hozzá szeretnénk férfi akkor dekódolni kell őket. Ezek a folyamatok általában valamilyen úgynevezett kriptográfiai kulcsokkal zajlanak. A kulcs lehet egy szám vagy bitsorozat, amelyeknek a hossza meg van adva. Ez a hossz a kulcsméret, ez meghatározza a kódolás biztonságát. Kulcs generálás soránaz adott hosszságú bitsorozatok közül véletlen módszerrel választunk egyet. A kódolást feltörni úgy lehetséges ha egy program segítségével végigpróbáljuk az összes lehetséges esetet, azonban ez egy elég nagy hosszságú kulcs esetén szinte lehetetlen.

Ezeket a módszereket két fő csoportra lehet osztani: Szimmetrikus illetve asszimetrikus kulcsú kriptográfiára. Szimmetrikus kódolás esetén, ugyanazt a kódot használjuk kódolásra illetve dekódolásra is. Ebben az esetben ezt a kulcsot nem adhatjuk ki senkinek. Ennek a hátránya, hogy a kulcsot nehéz biztonságosan eljuttatni másik félnek. Az ilyen titkosítások gyorsak és kisméretű kulcsokat használnak. Asszimetrikus kódolás esetén van egy publikus illetve egy magánkulcsunk. A publikus kulcsunkkal tudnak nekünk üzenetet küldeni, azonban dekódolni csak a magánkulccsal lehet. Hátrányuk ezeknek a módszereknek, hogy jelentősen lassabbak, illetve kevés ilyen módszer nyilvános. A nyilvános illetve magánkulcsok között általában matematikai összefüggés van.

Az RSA egy asszimetrikus algoritmus. Az alapötlet az, hogy a titkosítást elválasztja a dekódolástól, ezt nyilvános kulcsú rejtjelezésnek nevezzük. Tehát lesznek nyilvános kulcsok, amik segítségével nekünk tudnak küldeni ezzel a kóddal kódolt üzeneteket. Azonban, ezt az üzenetet ugyanazzal a kóddal nem tudjuk dekódolni, csak a titkos kulccsal. Ezt az algoritmust 1976-ban fejlesztettek ki, és a mai napig ez a leggyakrabban használt asszimetrikus titkosítási eljárás. Ez a fajta titkosítás moduláris hatványozással történik. Ez azt jelenti, hogy amikor kódolunk, akkor az adott üzenetet felemeljük a kulcs által megadott hatványra, majd az így kapott eredményt elosztjuk egy moduloval és az így kapott eredménnyel számolunk tovább. Nézzük lépésenként, hogy is történik egy ilyen titkosítás: Először is kiválasztunk ket nagy prímszámot véletlenszerűen, ezeket p illetve q betűvel jelöljük. Ezután ezt a két számot összeszorozva kapjuk a modulot, azaz m-et. Ezután az Euler féle fű függvény értékét számoljuk ki m-re. Majd kiválasztunk egy olyan e számot amely 1-nél nagyobb de az előbb kapott érténél kisebb és legnagyobb közös osztójuk 1 lesz. Ez az e értéke lesz a nyilvános kulcsunk kifejezése. A d érték lesz a titkos kulcsunk kifejezése amit ugy kapunk meg d^e kongruencia(mod fí(m)) -re teljesüljön. A nyilvános kulcsunk az m-ből illetve az e kifejezéstől fog állni. A titkos kulcs pedig az m-ből illetve a d kifejezéstől.

Bátfai Norbert tanár úr által belinkelt diáról kiindulva, nézzünk meg egy RSA kulcs generátort javabán.

```
public class RSAA {  
  
    public static void main(String[] args) {  
  
        int meretBitekben = 700* (int) (java.lang.Math.log((double) 10)/ java. lang.Math.log((double) 2 ));  
        System.out.println("Méret bitekben:");  
        System.out.println(meretBitekben);  
        java.math.BigInteger p_i =  
            new java.math.BigInteger(meretBitekben, 100, new java.util.Random());  
        System.out.println("p_i");  
        System.out.println(p_i);  
        System.out.println("p_i hexa");  
        System.out.println(p_i.toString(16));  
        java.math.BigInteger q_i =  
            new java.math.BigInteger(meretBitekben, 100, new java.util.Random());  
        System.out.println("q_i");  
        System.out.println(q_i);  
        java.math.BigInteger m_i = p_i.multiply(q_i);  
        System.out.println("m_i");  
        System.out.println(m_i);  
        java.math.BigInteger z_i = p_i.subtract (java.math.BigInteger.ONE).multiply (q_i.subtract(java.math.BigInteger.ONE));  
        System.out.println("z_i");  
        System.out.println(z_i);  
        java.math.BigInteger d_i;  
        do{  
            do{  
                d_i = new java.math.BigInteger(meretBitekben,new java.util.Random());  
            } while (d_i.equals(java.math.BigInteger.ONE));  
        }while (!z_i.gcd(d_i).equals(java.math.BigInteger.ONE));  
        System.out.println("d_i");  
        System.out.println(d_i);  
  
        java.math.BigInteger e_i = d_i.modInverse(z_i);  
        System.out.println("e_i");  
        System.out.println(e_i);  
    }  
}
```

Ez a kód nyilvános és magánkulcsokat fog generálni. Hazsnálni fogjuk a BigInteger javas osztályt, ami egy kibővített integer osztály, mivel nagy hosszúságú számokkal dolgozunk, és a sima integer nem tudná eltárolni őket. Ezután megadjuk a kulcsméretét bitekben. Majd a Random függvény segítsgével ezen intervallumon belül véletlenszerűen választunk egy p illetve q prímszámot. Majd a többi változót is létrehozzuk, és a már fent említett kulcsok jönnek létre.

a kapott kulcsok:


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

Az RSA osztály fogja a szöveget kódolni, illetve deódolni. Először példányosítjuk a KulcsPar osztályt majd beolvassuk a szöveget. Majd a szöveget karakterenként fogja kódolni a kód, ami egy rossz implementálás, mert így könnyebben visszatörhető, általában az egész szöveget egyszerre szokták kódolni. Ebben a kódban is használjuk a BigInteger osztályt. A modPow függvény fogja végezni a hatványozást. Majd az osztály vissza is kódolja nekünk a szöveget. A kód másik rész a KulcsPar osztály ami a kulcsokat fogja nekünk megadni. Itt is megadjuk a kulcsméretet, majd véletlenszerűen létrehozzuk p-t illetve q-t.

Futtatás:


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

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

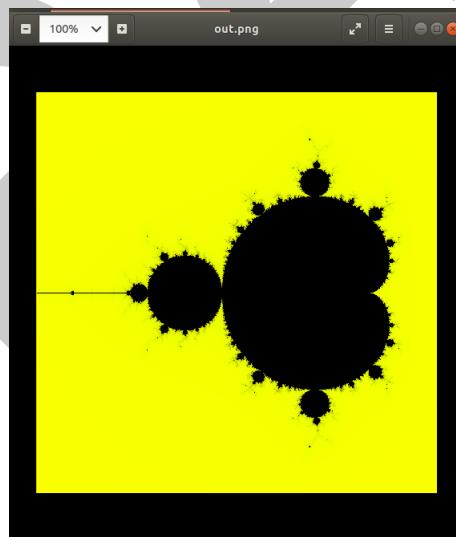
for (int i = 0; i < png_image.get_width(); i++)
    for (int j = 0; j < png_image.get_height(); ++j)
        png_image[i][j].blue = newimage[i*png_image.get_width() + j];

png_image.write("out.png");

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

ELÁTHATÓ,hogy kibővült egy kicsit a kódunk.A Perceptron osztály konstruktora egy változó argumentumszámú konstruktor,azaz először megnézi a kapott argumentumokat,és az alapján fogja létrehozni az objektumokat.A kódot már addig elmagyaráztam az előző csokorban,hogy eltároljuk a red komponensét.Azután következnek az új dolgok. Először is létrehozunk egy double pointert, ami megkapja a p objektumot illetve a beolvassott képet. Majd ebbe az új képhez beolvassuk az eredeti kép kék komponensét.Majd ezekután a png_image.write-al kimentjük a képet out néven.Azonban,hogy ez a kód működjön az mlp.hpp header fájlunkat is át kell alakítanunk, az operator() visszatérési értékét pointerre kell tennünk,hogy módosítani tudjuk a képünket.

a kapott kép:



17. fejezet

Helló, Gödel!

17.1. STL map érték szerinti rendezése

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

Ebben a feladatban egy STL mapot kell érték szerint rendeznünk. Az STL map egy alapjában növekvően rendezett asszociatív tömb. Ebben kulcs érték párokat tudunk tárolni. Deklarálás során meg kell adnunk a kulcs illetve az érték típusát is. A kódöt a Bátfai Norbert tanár úúr által belinkelt oldalról szedtem le.

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

#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>
#include <boost/program_options.hpp>
#include <boost/tokenizer.hpp>
#include <boost/date_time posix_time posix_time.hpp>

using namespace std;

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

    for ( auto & i : rank ) {
        if ( i.second ) {
            pair<string, int> p {i.first, i.second};
            ordered.push_back ( p );
        }
    }

    sort (
        begin ( ordered ), end ( ordered ),
        less<pair<string, int>>()
    );
}
```

```
[ = ] ( auto && p1, auto && p2 ) {
    return p1.second > p2.second;
}

return ordered;
}

int main(){

map<string, int> map;

map["alma"] = 1;
map["barack"] = 2;
map["citrom"] = 3;
map["dinnye"] = 4;
map["eper"] = 5;

vector<pair<string, int>> done = sort_map(map);

for(auto& i : done)
    cout<<i.first << " " <<i.second<<endl;

}
```

A sort_map függvény fogja végezni a rendezést.Egy STL map alapból kulcsszerint van rendezve, nekünk pedig érték szerint kell.Ahhoz hogy ezt megtudjuk tenni létre kell hoznunk egy vektort, ami párokat fog tárolni, és ezt a vektort fogjuk rendezni.Ehhez a sort függvényt fogjuk használni.Ebben egy lambda kifejezést használunk, ami igaz lesz ha az első pair értéke nagyobb lesz mint a második.Ezután ezt a rendezett vektort fogja visszaadni a függvény.A main függvényben létrehozunk egy map-et, majd feltöljük azt és meghívjuk rá a sort_map függvényt.A végén pedig a már visszakapott vektort fogjuk kiíratni egy for ciklus segítségével.

futtatás:

```
foldesizoltan@Aspire: ~/Asztal/prog2/godel
Fájl Szerkesztés Nézet Keresés Terminál Súgó
foldesizoltan@Aspire:~/Asztal/prog2/godel$ ./stl
eper 5
dinnye 4
citrom 3
barack 2
alma 1
foldesizoltan@Aspire:~/Asztal/prog2/godel$
```

17.2. Gengszterek

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

Ebben a feladatban a lambda függvényekről fogunk beszélni. A lambda függvények vagy kifejezések nvtelenek. Ott definiálhatjuk őket a kódban ahol éppen szükségünk van rá. Nézzünk egy példát lambda függvényre.

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

A lambda függvényeket egy üres szöglletes zárójellel tudjuk bevezetni. Ezután kerek zárójelek közé megadhatjuk a várt paramétereket, de ha nem tesszük meg akkor a fordító kitalálja azokat. Ezután kapcsos zárójelek közt a return szó után adjuk meg a visszatérítéi értéket, de itt nem kell megadnunk típust, ezt a fordító fogja megadni. Ezek után a sima zárójelpár a függvényhívást jelöli.

Nézzük a feladatunkat, a Robotautó Világ bajnokságos kódban kell értelmezniünk a rendezéshez használt lambda kifejezést. Ezt a kifejezést a myshmclient.cpp fájlban találjuk.

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

Amint látjuk, itt a gangster vektor rendezéséhez a sort függvényt használjuk, és ennek a függvénynek a paraméterei között van a lambda kifejezés. Ez a lambda kifejezés igaz vagy hamis értéket fog visszatéríteni. A paraméter listájában láthatjuk, hogy két Gangster objektumot vár paraméterként. Akkor fog igaz értéket visszaadni ha az x gangster közelebb van a cophoz mint y. Így az egész rendezés végén az a gangster lesz a vektorunk elején aki közelebb van a cophoz.

17.3. GIMP Schme hack

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

Ebben a feladatban egy scheme programot fogunk megírni, ami majd a gimp programban fog egy mandalát rajzolni, az általunk megadott szövegből illetve az általunk megadott színekkel. A scheme programozási nyelv a Lisp programozási nyelvnek az egyik nyelvjárása. Scheme-ben, csakúgy mint a lispben is teljesen zárójelezett prefix kifejezések vannak. Tehát a műveleti jelek előre fognak kerülni és csakis teljesen zárójelezve fognak működni a kifejezések. Ebben a nyelvben is működik az automatikus szemétgyűjtés, csakúgy mint java-ban. Az eljárások hívásánál a paramétereket minden érték szerint adjuk át. A kódot Bátfai Norbert tanáár úr honlapjáról szedtem le. A kódot a Gimp mappáján belül a scripts mappába kell bemásolni, és már látrehozható lesz a mandala.

```
; bhax_mandala9.scm
;
; BHAX-Mandala creates a mandala from a text box.
; Copyright (C) 2019 Norbert Bátfai, batfai.norbert@inf.unideb.hu
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with this program. If not, see <https://www.gnu.org/licenses/>.
;
; Version history
;
; This Scheme code is partially based on the Python code
; Pat625_Mandala_With_Your_Name.py by Tin Tran, which is released under ↵
; the GNU GPL v3, see
; https://gimplearn.net/viewtopic.php/Pat625-Mandala-With-Your-Name-Script-for-GIMP?t=269&p=976
;
; https://bhaxor.blog.hu/2019/01/10/a\_gimp\_lisp\_hackelese\_a\_scheme\_programozasi\_nyelv
;

(define (elem x lista)

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

)
```

```
(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font))))
  text-width
)
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  ;;;
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  ;;; ved ki a lista 2. elemét
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
  ;;;
  (list text-width text-height)
)
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
  gradient)
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
      LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-layer)
    (text-width (text-width text font fontsize))
    ;;;
    (text2-width (car (text-wh text2 font fontsize)))
    (text2-height (elem 2 (text-wh text2 font fontsize)))
    ;;;
    (textfs-width)
    (textfs-height)
    (gradient-layer)
```

```
)  
  
(gimp-image-insert-layer image layer 0 0)  
  
(gimp-context-set-foreground '(0 255 0))  
(gimp-drawable-fill layer FILL-FOREGROUND)  
(gimp-image-undo-disable image)  
  
(gimp-context-set-foreground color)  
  
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←  
    ))  
(gimp-image-insert-layer image textfs 0 -1)  
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←  
    height 2))  
(gimp-layer-resize-to-image-size textfs)  
  
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))  
(gimp-image-insert-layer image text-layer 0 -1)  
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)  
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←  
    -LAYER)))  
  
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))  
(gimp-image-insert-layer image text-layer 0 -1)  
(gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)  
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←  
    -LAYER)))  
  
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))  
(gimp-image-insert-layer image text-layer 0 -1)  
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)  
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←  
    -LAYER)))  
  
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))  
(gimp-image-insert-layer image text-layer 0 -1)  
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)  
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←  
    -LAYER)))  
  
(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)  
(set! textfs-width (+ (car (gimp-drawable-width textfs)) 100))  
(set! textfs-height (+ (car (gimp-drawable-height textfs)) 100))  
  
(gimp-layer-resize-to-image-size textfs)  
  
(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←  
    (/ textfs-width 2)) 18)  
    (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
```

```
    textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
    (/ textfs-width 2)) 18)
    (- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
        textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ←
    "gradient" 100 LAYER-MODE-NORMAL-LEGACY)))

(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
    GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ ←
    (/ width 2) (/ textfs-width 2)) 8) (/ height 2))

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
    )))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
    height 2) (/ text2-height 2)))

;(gimp-selection-none image)
;(gimp-image-flatten image)

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-mandala "Bátfa Norbert" "BHAX" "Ruge Boogie" 120 1920 ←
 1080 '(255 0 0) "Shadows 3")

(script-fu-register "script-fu-bhax-mandala"
```

```
"Mandala9"
"Creates a mandala from a text box."
"Norbert Bátfai"
"Copyright 2019, Norbert Bátfai"
"January 9, 2019"
""

SF-STRING      "Text"        "Bátf41 Haxor"
SF-STRING      "Text2"       "BHAX"
SF-FONT        "Font"        "Sans"
SF-ADJUSTMENT  "Font size"   '(100 1 1000 1 10 0 1)
SF-VALUE        "Width"       "1000"
SF-VALUE        "Height"     "1000"
SF-COLOR        "Color"       '(255 0 0)
SF-GRADIENT    "Gradient"   "Deep Sea"
)

(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

A script-fu-bhax-mandala függvény fogja a mandalat rajzolni. A define szóval deklarálhatjuk a függvényeket. Ez a függvény 8 paramétert fog kérni. Kettő szöveget, betűtípusot és betűméretet, a kép magasságát és szélességét, valamint a betűk színét, színátmenetét. A let szócskával azt adjuk meg, hogy az ebben a függvényben létrejövő lokális változókat fogjuk a későbbiekben is használni. Scheme-ben minden függvény egy listát fog visszaadni. minden beépített függvényt megtudunk nézni a súgó menüpont eljárásbongésző menüpont alatt.

Például az

```
(image (car (gimp-image-new width height 0)))
```

sor image néven létre fog hozni egy új képet, a megadott magasság illetve szélesség paraméterrel, a 0 pedig a végén azt jelenti, hogy RGB képet fogunk kapni, a 0 helyett írhatnánk RGB-t is. Ez a függvény egy darab képet fog visszaadni, vagyis egy egyelemű listát, amiben csak a kép lesz. A car szócskával minden listából az első elemet vesszük ki.

Ezután létrehozzuk a többi változókat. Scheme-ben a színeket a következőképpen adhatjuk meg

```
' (0 255 0)
```

ez rgb szerinti megadást jelent. A következőkben beállítjuk a háttérszínt, illetve a betűk színeit, és létrehozzuk a szükséges rétegeket. A gimo-item-transform-rotate függvénytel fogjuk a szöveget forgatgatni. A kód végén található script-fu-register résszel regisztráljuk a szkriptet, ezáltal válik elérhetővé a létrehozás menüpontban. Itt megadjuk a paraméterek típusát, és egy alapértéket is.

a kapott kép:



17.4. Alternatív Tabella rendezése

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

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

Ebben a feladatban a labdarugó bajnoksághoz fogunk alternatív tabellát készíteni. Az alternatív tabella nem a hagyományos módon határozza meg a csapatok sorrendjét, hanem figyelembe veszi, hogy milyen csapat játszott milyennel szemben. Tehát figyeli a csapat erősséget is, és ha egy gyengébb csapat egy erősebb csapatot ver meg akkor az többet ér. Ehhez a már az előző félévben tanult PageRank algoritmusát használja fel a kód.

Először is két programunk lesz, a Wiki2Matrix fogja a kereszttáblázatot mátrixká alakítani, és az ezáltal a program által visszakapott értékeket kell majd betennünk az AlternativTabella programba, ami a rangsorlást fogja végezni. A Wiki2Matrix programban a kereszt tömbjébe a wikipédián található kereszttáblázatot kell reprezentálnunk, még hozzá úgy, hogy a zöld szín az 1-es lesz, a sárga 2-es a piros 3-as az üres pedig 0-ás. Ezután a kapott link mátrixot bemásolva a másik programunkba kapjuk az eredményt.

```
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.erkek < csapat.erkek) {  
    return -1;  
} else if (this.erkek > csapat.erkek) {  
    return 1;  
} else {  
    return 0;  
}
```

A Csapat osztályunk implementálja a Comparable interfést. Az interfészben lehetnek konstansok illetve függvények. A Comparable interfészben csak egyetlen metódus található. Ez a metódus a Csapat objektumokra fog működni és egy negatív, vagy egy pozitív számot, vagy nullát fog visszaadni, attól függően, hogy kisebbek, nagyobbak, vagy egyenlők az összehasonlított objektumok.

Ez az implementálás a következő sor nál lesz lényeges.

```
java.util.Collections.sort(rendezettCspatok);
```

Ugyanis a sort függvény a compareTo metódust fogja használni a rendezéshez, és mivel ez a Comparable interfészben található, így annak az osztálynak implementálnia kell azt, ahhoz hogy tagjait rendezni tudjuk ezzel a függvénnel.

DRAFT

foldesiz

Fájl Szerkesztés Nézet Keresés Terminál Súgó

Csapatok rendezve:

- - | Ferencváros
 - | 74
 - | Ferencváros
 - | 0.1035
- - | Vidi
 - | 61
 - | Paks
 - | 0.1013
- - | Debreceni
 - | 51
 - | Debrecen
 - | 0.1012
- - | Honvéd
 - | 49
 - | Vidi
 - | 0.1008
- - | Ujpest
 - | 48
 - | Ujpest
 - | 0.0975
- - | Mezőkövesd
 - | 44
 - | Mezőkövesd
 - | 0.0850
- - | Puskás Akadémia
 - | 40
 - | Honvéd
 - | 0.0814
- - | Paks

JDK forrás:

```
2:  */
3:  public static <T extends Comparable<? super T>> void sort(List<T> l)
4:  {
5:      sort(l, null);
6:  }
7:
8: /**
9:  * Sort a list according to a specified Comparator. The list must be
10: * modifiable, but can be of fixed size. The sort algorithm is precisely that
11: * used by Arrays.sort(Object[], Comparator), which offers guaranteed
12: * nlog(n) performance. This implementation dumps the list into an array,
13: * sorts the array, and then iterates over the list setting each element from
14: * the array.
15: *
16: * @param l the List to sort (<code>null</code> not permitted)
17: * @param c the Comparator specifying the ordering for the elements, or
18: *          <code>null</code> for natural ordering
19: * @throws ClassCastException if c will not compare some pair of items
20: * @throws UnsupportedOperationException if the List is not modifiable
21: * @throws NullPointerException if the List is <code>null</code> or
22: *          <code>null</code> is compared by natural ordering (only possible
23: *          when c is <code>null</code>)
24: *
25: * @see Arrays#sort(Object[], Comparator)
26: */

```

DRAFT

18. fejezet

Helló,!

18.1. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/Samu> Ebben a feladatban egy webkamera kezelésére kell rámutatnunk a belinkelt feladatban. A SamuCam program emberi arcokat fog tanulni és észlelni. EHhez OpenCV-t fogunk használni. Az OpenCV egy nyíltforrású csomag, amiben a gépi látáshoz illetve gépi tanuláshoz vannak benne algoritmusok.

Ez a program egy QT projekt, azonban nekünk a webkamera kezelését kell bemutatnunk, ezt pedig a SamuCam.cpp állományban tesszük meg. Először is, ahoz, hogy képet kapunk én a laptopom webkameráját használtam, ehhez át kellett egy kicsit írni a kódot. Megnyitva az OpenCV dokumentációját, amiben a VideoCapture-re kerestem, megtalálható, hogy a VideoCapture.open paraméterként ip helyett kaphat indexet is, és ha indexként a 0-át adjuk meg akkor az eszköz alapértelmezett kameráját fogja használni, ami jelen esetben a laptop webkamerája lesz. Itt láthatjuk a kód részletet átírás után.

```
#include "SamuCam.h"

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

SamuCam::~SamuCam ()
{
}

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

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

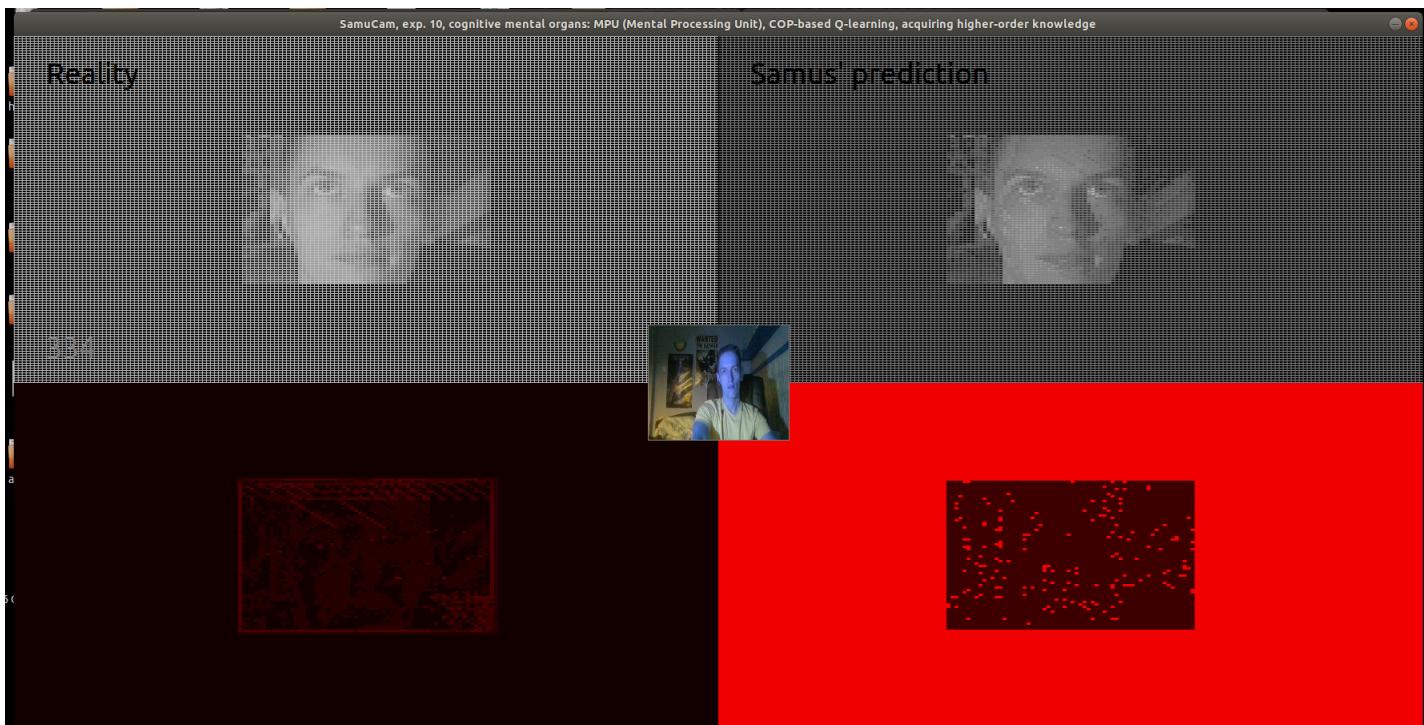
Ebben a kódrészletben láthatjuk még a konstruktort illetve a destruktort. Az openVideoStream függvényben nyitjuk meg a kameránkat. Majd ezután beállítjuk a magasságot, illetve a szélességet. Majd a képfrissítést is, ami jelen esetben 10 lesz. Nézzük tovább a kódunkat.

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

A run függvényben fog lezajlani az arc felismerés. Ezt az úgy nevezett CascadeClassifier -el végezzük el. Ez is az openCV-ben található, és a képek feldolgozásáért lesz felelős. Ehhez szükségünk lesz a lbpcascade_frontalface xml fájlra, enélkül nem fog futni a programunk. Ez fogja úgymond a szabályokat tartalmazni, hogy miszerint keresse az arcokat.

Ezután létrehozunk egy Mat tömböt amiben később a beolvasott képeket fogjuk tárolni. Majd a while ciklusban fog történni maga a futás. 50 milisecundumonként beolvsunk egy képet, az előbb létrehozott frame tömbbe. Majd a resize függvénnyel átméretezzük 176x144 felbontásra. Majd a cvtColor függvénnyel átváltjuk a képet szürkeskálás képpé és elmentjük egy másik tömbbe. Majd elvégzünk egy hisztogram kiegyenlítést, ezáltal a kisebb részletek jobban láthatóak lesznek. Ezek után a detectMultiScale segítségével megkeressük az arcokat a képen, amik egy Rectangle listában lesznek tárolva. Majd ezeknek létrehozunk egy-egy QImage-t. Majd webcam néven létrehozunk egy keretet az arc körül, megint csak a QImage segítségével. A végén pedig ha nincs megnyitva a kameránk akkor visszatér a program a legelejére, és újrapróbalozik.

futtatás:



18.2. BrainB

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

Ebben a feladatban a már az előző fejezetben tárgyalt BrainB programról fogunk beszélni, de most a slot-signal mechanizmust fogjuk részletesebben áttekinteni. A BrainB programmal azt figyeljük, hogy milyen sokáig tudunk követni egy hőst a kijelzőn, miközben az mozog illetve körülötte ugyanolyan hősök jelennek meg. Ez a program is egy Qt-s projekt. Most pedig nézzük, hogy mi is az a slot-signal mechanizmus.

A signalok illetve slotok segítségével tudunk Qt-ben objektumok között kommunikálni. Ez a signal-slot mechanizmus csak a Qt-ben található meg. Az objektumok képesek magukból különböző signalokat kibocsátani (emittálni), ha valami lényeges dolog történik velük. Valamint az objektumoknak lehetnek slotjaik, amelyek ezeket a signalokat fogják figyelni. Ezeket a signalokat és slotokat mi hozzuk létre és kötjük össze őket. Egy signalt és egy slotot a connect parancsval tudunk összekötőni. Két ilyen connect található a BrainBWin.cpp fájlunkban, most ezeket fogjuk megnézni.

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

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

A connectnek négy paramétere lesz. Az első lesz az az objektum ami küldi a signalt. A második lesz a signal, jelen esetben a harmadik a this lesz, azaz az adott osztály. Az utolsó paraméter pedig a slot lesz. Tehát ha a heroesChanged signal kibocsátásra kerül akkor lefut az updateHeroes függvény. Ha pedig

az endAndStats signal kerül kibocsájtásra akkor pedig az endAndStats függvény fog meghívódni ami kiírja a végeredmény.A meghívott függvények :

```
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();
}

void BrainBWin::updateHeroes ( const QImage &image, const int &x, const int &y )
{
    if ( start && !brainBThread->get_paused() ) {

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

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

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

                firstLost = true;

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

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

                if ( state == lost && firstLost ) {
```

```
        lost2found.push_back ( brainBThread ←
                               ->get_bps () ) ;
    }

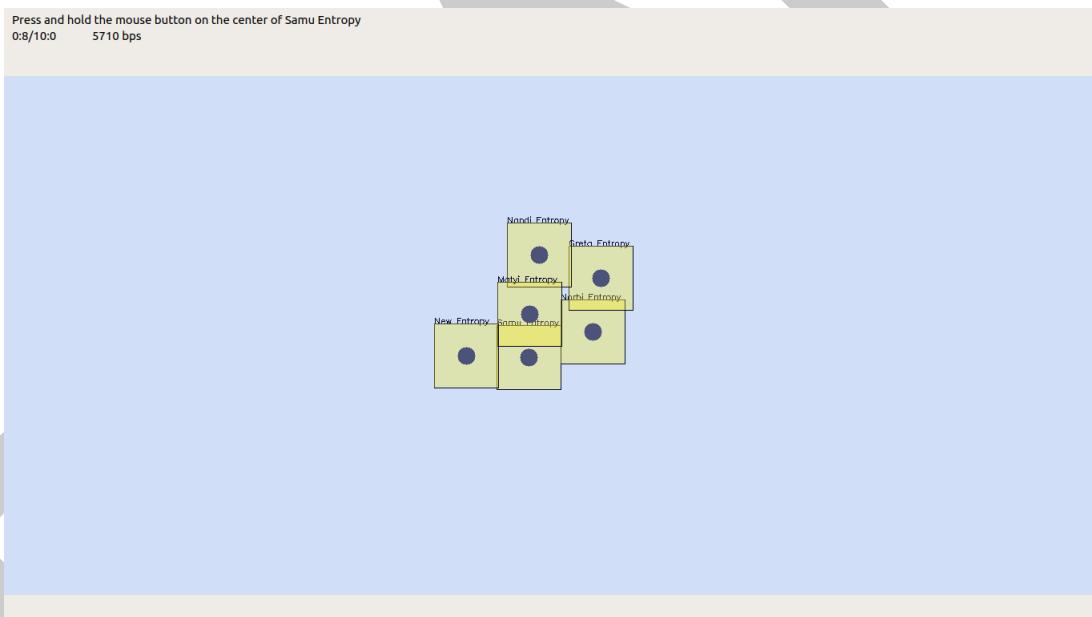
    state = found;
    noffound = 0;
    //qDebug() << "FOUND";
    //double mean = brainBThread->meanFound();
    //qDebug() << mean;

    brainBThread->incComp();
}

}

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

futtatás:



18.3. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/justine/r>

Ebben a feladatban a Robocar-os program alapján fogjuk megnézni a sscnf függvény szerepét. Az sscnf függvény formázott stringből fog adatokat beolvasni. Paraméterként először meg kell adnunk, hogy honnan fogjuk az adatokat beolvasni. Majd ezután azokat adjuk meg, hogy milyen formázással mentse el. A különböző típusokat egy % jellel és utána egy rövidítéssel tudjuk megadni. Például a %d az integert fogja jelölni, a %s egy stringet fog jelölni stb. Nézzük is, hogy hogy alakul ez a carlexer.ll fájlban.

```
std::sscanf(yytext, "<init %s %c>", name, &role);
```

Az első paraméter ugyebár a forrás lesz,hogy honan olvassuk be a dolgokat,jelen esetben a yytext-ből fogjuk.Ezután megvan adva egy %s azaz string illetve egy %c ami egy darab karaktert fog jelölni. Tehát ha beolvasáskor talál egy stringet akkor a függvény azt a name változóba fogja elmenteni, az egy darab karaktert pedig a role-ba.

```
std::sscanf(yytext, "<car %d", &m_id);
```

Ebben a sorban ugyancsak az yytext lesz a bemenet és egy integert, fog keresni a sscnf, ha talált akkor azt az m_id-ba fogja menteni. Még sok ilyen sscnf-el találkozhatunk ebben a fájlban,de nem fogjuk minden megnézni.Ebben a fájlban a sscnf szerepe az adatok feldolgozása,de szokás még adatok helyességére is használni.

18.4. OSM térképre rajzolás

Debrecen térképre dobjunk rá cuccokat, ennek mintájára, ahol én az országba helyeztem el a DEAC hekkereket: <https://www.twitch.tv/videos/182262537>(de az OOCWC Java Swinges megjelenítőjéből: <https://github.com/emulator/tree/master/justine/rcwinis> kiindulhatsz, mondjuk az komplexebb, mert ott időfejlődés is van...)

Ebben a feladatban egy egyszerű GPS trackert fogunk írni.Ehhez megintcsak az android studiot fogjuk használni.Szerencsére nagyon sok tutorial van fent a neten ezzel a témaval kapcsolatban.Az android stúdiónak van GoogleMaps alapú presetje. Először is szükségünk lesz egy API key-re amit a google developer oldaláról tudunk igényelni.Ezután ezt az API keyt bemásoljuk a google_maps_api.xml-be. Ezután már működő képes lesz a programunk. Ahhoz,hogy használni tudjuk majd a telefonunk helymeghatározását,szükségünk lesz az engedélyre. Ezt az AndroidManifest.xml fájlban tehetjük meg.

```
<uses-permission android:name="android.permission. ↵
    ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission. ↵
    ACCESS_COARSE_LOCATION" />
```

Ezzel a két sorral fogjuk megadni,hogy használni fogjuk a helymeghatározást.Ezután a többi dolgot a MapActivity.java fájlban fogjuk végezni.Először is azt szeretnénk,hogy az előbb említett engedélyeket a felhasználó engedélyezze,ezért létrehozzuk a következő sorokat:

```
ActivityCompat.requestPermissions(this, new String[] {Manifest.permission. ↵
    ACCESS_FINE_LOCATION}, PackageManager.PERMISSION_GRANTED);
ActivityCompat.requestPermissions(this, new String[] {Manifest. ↵
    permission.ACCESS_COARSE_LOCATION}, PackageManager. ↵
    PERMISSION_GRANTED);
```

Ezeket a sorokat az onCreate függvénybe kell tennünk. Ezek a sorok felelősek a felugró ablakért ami az első futáskor nyílik meg, és adhatjuk meg az engedélyt a helymeghatározáshoz. Ahhoz, hogy nyomon tudjuk követni a helyzetünket, szükségünk lesz egy LocationManagerre illetve egy LocationListenerre. Ezeket a kód elején privát változóként hoztuk létre. A LocationManager fogja figyelni a helyzetünket. A requestLocationUpdates függvény fogja meghívni a locationlisteneret. Ez a függvény a következőképpen néz ki.

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, ←  
    MIN_TIME, MIN_DIST, locationListener);
```

Paraméterként először megadjuk, hogy a GPS-től fog függni a helyváltoztatás. Majd a MIN_TIME és MIN_DIST számok jönnek, amit a kód elején deklaráltunk. A MIN_TIME azt fogja jelenteni, hogy milyen sűrűn frissítse a helyzetet, ez milisecundumban van megadva, jelen esetben 1000 ms azaz 1 másodperc. A MIN_DIST pedig a minimális távolságot fogja megadni, ezt méterben határozzuk meg, jelen esetben 5 méter. Az utolsó paraméter pedig azt jelenti, hogy megfogjuk hívni a locationlisteneret, ō fogja végezni a helyzetünk felrajzolását a térképre.

```
locationListener = new LocationListener() {  
    @Override  
    public void onLocationChanged(Location location) {  
  
        try {  
            LatLng = new LatLng(location.getLatitude(), location. ←  
                getLongitude());  
  
            mMap.addMarker(new MarkerOptions().position(latLng). ←  
                title("My Current Position").icon( ←  
                    BitmapDescriptorFactory.fromResource(R.drawable. ←  
                        marker4)));  
            mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));  
        }  
        catch (SecurityException e){  
            e.printStackTrace();  
        }  
    }  
  
    @Override  
    public void onStatusChanged(String s, int i, Bundle bundle) {  
  
    }  
  
    @Override  
    public void onProviderEnabled(String s) {  
  
    }  
  
    @Override  
    public void onProviderDisabled(String s) {  
  
    }  
};
```

Először is példányosítjuk a locationlisteneret. Majd az onLocationChanged függvényt fogjuk használni, melynek paraméterében létrejön egy location objektum ami egy földrajzi helyzetet tud reprezentálni. Az onLocationChanged függvény pedig akkor hívódik meg, hogy ha változik a helyzetünk. Ebben a függvényben értéket adunk a már a kód fenti részében létrehozott latLng típusú objektumnak. A LatLng egy koordináta párt képes tárolni. Szóval elmentjük benne a jelenlegi szélességi és hosszúsági helyzetünket. Majd az addMarker függvénnnyel egy markert helyezünk a térképre, az előbb elmentett pozícióba. A .title el megadhatjuk, hogy mit írjon ki amikor ráteszszük az ujjunkat. A .iconnal pedig saját ikonokat használhatunk. Én jelen esetben a netről szedtem le egy fehér markert, amit a projekt drawable mappájába bemásolva, majd a .icon helyes használatával meg is tudtam jeleníteni. Ezután a moveCamera-val a kamerát a helyzetünkhez visszük. Így a helyzetünk figyelése kész is van.

Lehetőségünk van a térképet különböző stílusokban megjeleníteni. Ehhez úgynévezett json fájlokat használunk. A mapstyle.withgoogle oldalon lehetőségünk van grafikusan létrehozni egy stílust majd azt json kóddá generáltatni. Én úgy csináltam meg a programot, hogy két gombot tettem rá, és válthatunk egy sötét illetve egy világos téma között. Ehhez szükség volt két json fájlra, amit a projekt raw mappájába helyeztem. Ezután az activity_maps.xml fájlban létrehoztam két gombot.

```
<Button
    android:id="@+id/Dark"
    style="@style/Widget.AppCompat.Button.Borderless.Colored"
    android:layout_width="213dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:onClick="Dark"
    android:text="Dark"
    tools:ignore="OnClick" />

<Button
    android:id="@+id/Light"
    style="@android:style/Widget.DeviceDefault.Button.Borderless. ←
        Colored"
    android:layout_width="213dp"
    android:layout_height="match_parent"
    android:layout_weight="1"
    android:background="#00FFFFFF"
    android:onClick="Light"
    android:text="Light"
    tools:ignore="OnClick" />
```

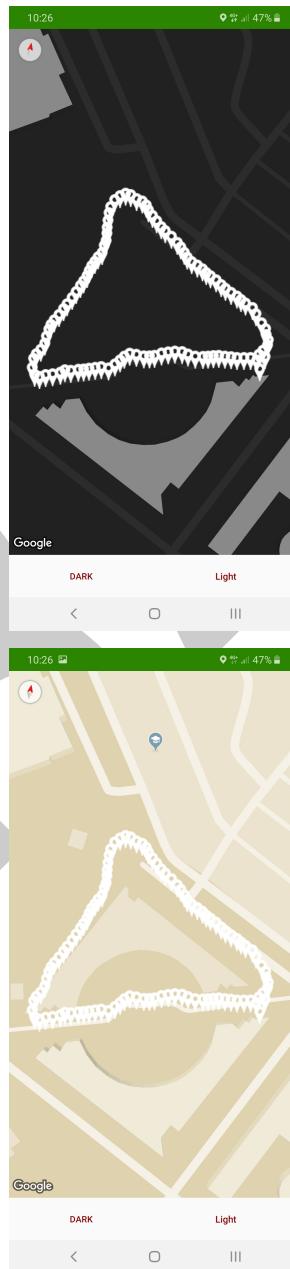
Itt megtudjuk adni a gomb stílusát, méretét, nevét valamint, hogy nyomható legyen. Az id-vel tudunk majd rá hivatkozni a MainActivityben. A MainActivityben létrehoztam két függvényt ami a gombokat figyeli, hogy mikor nyomják meg.

```
public void Dark(View view) {
    if(view.getId()== R.id.Dark)
        mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(this, R. ←
            raw.dark));
```

```
}
```

```
public void Light(View view) {
    if(view.getId()== R.id.Light)
        mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(this, R. ←
            raw.light));
```

Az első függvény a Dark gombot figyeli, és ha megnyomjuk meghívódik a setMapStyle függvény, ami a már említett dark json fájlt fogja beállítani a térkép stílusának. A Light függvény pedig a másik gombot fogja figyelni, amit ha lenyomunk akkor a Light json-t fogja beállítani stílusként.



19. fejezet

Helló, Lauda

19.1. Port scan

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

Ebben a feladatban a kivételkezeléssel fogunk foglalkozni. A kivételek a program végrehajtásakor keletkezhetnek és ezzel megszakítják a program futását. Ha egy metódusban egy ilyen hiba keletkezik, akkor egy objektum jön létre ami tartalmazza a hiba típusát, és a program állapotát. Két fajta kivétel létezik: futási illetve nem futási időben keletkezett kivétel. Futási kivételek közé tartozik például az aritmetikus kivételek (pl. 0-val osztás), referenciaival kapcsolatos kivételek és az indexeléssel kapcsolatos kivételek. Nem futási kivételek az olyak kivételek melyek a futási rendszeren kívül keletkeznek, pl I/O kivételek.

A belinkelt példa azt fogja megnézni, hogy a gépünk milyen portokat figyel. A gép nevét parancssori argumentumként fogja megkapni. A for ciklusban lévő java.net.Socket 1024 socketet fog megpróbálni megnyitni (azaz TCP kapcsolatot létrehozni), ha a megnyitás sikeres akkor az azt jelenti, hogy a gép figyeli azt, és kifogja írni, hogy figyeli. Azonban ha nem figyeli akkor egy try block segítségével kivételt fog dobni, ami majd az utána következő catch block fog elkapni, majd kiírja, hogy azt a portot nem figyeli a gép. Ha rákeresünk a java.net.Socket metódusra, megtudjuk nézni, hogy milyen kivételt dobhat és mi annak az oka.

```
Socket
public Socket(String host,
               int port)
               throws UnknownHostException,
                      IOException
Creates a stream socket and connects it to the specified port number on the named host.
If the specified host is null it is the equivalent of specifying the address as InetAddress.getByName(null). In other words, it is equivalent to specifying an address of the loopback interface.
If there is a security manager, its checkConnect method is called with the host address and port as its arguments. This could result in a SecurityException.
Parameters:
host - the host name, or null for the loopback address.
port - the port number.
Throws:
UnknownHostException - if the IP address of the host could not be determined.
IOException - if an I/O error occurs when creating the socket.
SecurityException - if a security manager exists and its checkConnect method doesn't allow the operation.
IllegalArgumentException - if the port parameter is outside the specified range of valid port values, which is between 0 and 65535, inclusive.
See Also:
setSocketImplFactory(java.net.SocketImplFactory), SocketImpl, SocketImplFactory.createSocketImpl(), SecurityManager.checkConnect(java.lang.String, int)
```

Láthatjuk, hogy többféle kivétel létrejöhet. Itt I/O exception jön létre, ha nem sikerül valamiért megnyitni a portot.

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

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

    try {

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

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

        socket.close();

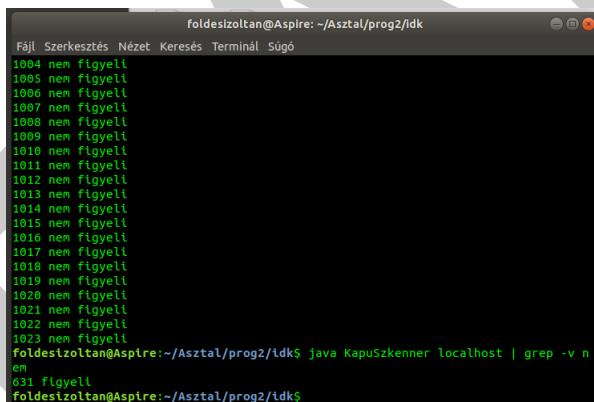
    } catch (Exception e) {

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

    }
}

}
```

futtatás:



A terminal window titled 'foldesizoltan@Aspire: ~/Asztal/prog2/idk' displays the output of a Java application. The application prints numbers from 1004 to 1023, followed by the string 'nem figyeli'. At the end of the output, there is a command: 'java KapusZkenner localhost | grep -v n', which filters out lines containing 'n'. The terminal prompt shows the user is back at the command line.

```
foldesizoltan@Aspire:~/Asztal/prog2/idk
1004 nem figyeli
1005 nem figyeli
1006 nem figyeli
1007 nem figyeli
1008 nem figyeli
1009 nem figyeli
1010 nem figyeli
1011 nem figyeli
1012 nem figyeli
1013 nem figyeli
1014 nem figyeli
1015 nem figyeli
1016 nem figyeli
1017 nem figyeli
1018 nem figyeli
1019 nem figyeli
1020 nem figyeli
1021 nem figyeli
1022 nem figyeli
1023 nem figyeli
foldesizoltan@Aspire:~/Asztal/prog2/idk$ java KapusZkenner localhost | grep -v n
em
031 figyeli
Foldesizoltan@Aspire:~/Asztal/prog2/idk$
```

19.2. Junit teszt

A https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_ely_pedatposzt kézzel számított mélységét és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

Ebben a feladatban a JUnit-al fogunk foglalkozni.A JUnit egy egységeszt keretrendszer Java nyelvhez.Ezekre az egységesztekre azért van szükség,hogy tudjuk,hogy amit leprogramoztunk az a vártnak megfelelően működik.A JUnit által a tesztelést is programként tudjukmegírni.A teszt metódusoknak létre szoktak hozni egy tesztosztályt.A feladatunk ,a belinkelt cikk alapján az,hogy a binfa java átírását teszteljük,hogy megfelelően működik-e.Bátfai Norbert tanár úr adatait használtam.Tehát írnunk kell egy JUnit teszt osztályt,amiben a megadott bemenetet beadjuk a binfának majd megadunk teszt metódusokat,amelyekkel ha a kézzel kiszámolt értékeket kapjuk, akkor a binfa megfelelően működik.Én az eclipse programban oldottam meg a feladatot.Nézzük a kódot.

```
package binfa;

import static org.junit.Assert.*;

import org.junit.Test;

public class bin {

    LZWBinFa tesztfa = new LZWBinFa();

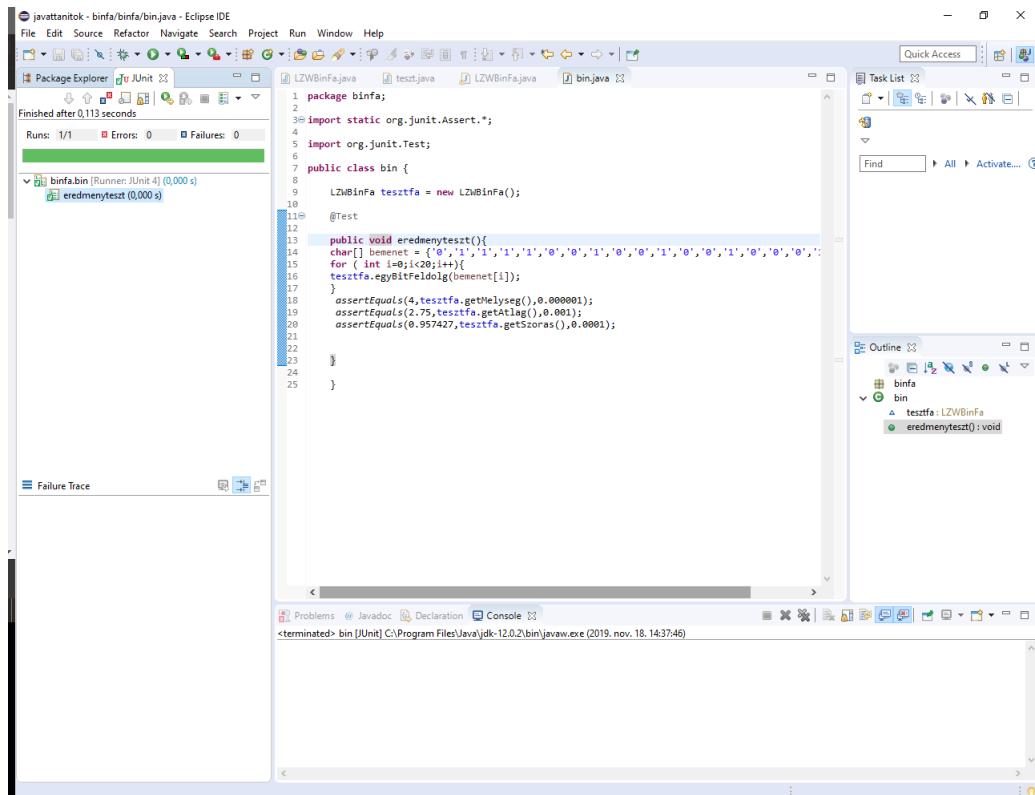
    @Test

    public void eredmenyteszt() {
        char[] bemenet = {'0','1','1','1','1','0','0','1','0','1','0','1','0','0','1' ←
            ',0','0','1','1','1'};
        for ( int i=0;i<20;i++) {
            tesztfa.egyBitFeldolg(bemenet[i]);
        }
        assertEquals(4,tesztfa.getMelyseg(),0.000001);
        assertEquals(2.75,tesztfa.getAtlag(),0.001);
        assertEquals(0.957427,tesztfa.getSzoras(),0.0001);

    }
}
```

Jelen esetünkben a bin osztály lesz a teszt osztály. Először is példányosítjuk a binfa osztályát. Majd a @Test kifejezéssel kezdjük a tesztelést, az ez alatt található összes metódus lefog futni, azonban a JUnit nem szab meg közdük sorrendet ezért úgy kell kialakítani a teszteket, hogy azok függetlenek legyenek egymástól. Ezután betöljük a linkben megadott bemenetet egy char tömbbe, majd egy for ciklus segítségével átadjuk ezeket a biteket az egyBitFeldolg függvénynek (azért ennek, mert már nem szükség átalakítani a bemenetet ugyanis már 0 ból és 1-esből áll). Majd következnek a tesztfüggvények. Jelen esetben 3 assertEquals függvényt fogunk használni. Többféle változata van az assertEquals függvénynek, mi most azt a változatát használjuk ahol három double típust vár paraméterként. Ez a függvény azt fogja megnézni, hogy a kapott illetve a várt eredmény egy bizonyos eltéréssel megegyezik-e. Az első paraméter lesz az a szám amit várunk eredményként, a második lesz a tényleges eredmény, ahol a megfelelő függvényeket adjuk meg. A harmadik paraméter pedig egy delta érték lesz, ami megadja, hogy mennyivel térhet el maximum az eredmény. Futtatás után megkapjuk, hogy volt-e hiba vagy sem, nekem tökéletesen működött a program.

futtatás:



19.3. Android játék

Ebben a feladatban egy egyszerű kis androidos játékot kellett írni. Az internet megint csak nagy segítségnek bizonyult.A játék létrehozásához az Android Studiot használtam. Egy számítálós játékot készítettem. Az alapötlet,hogy a program 0 és 100 között random generál egy számot, nekünk pedig ki kell azt találni. Ha nagyobb a szám mint amit beírunk akkor kiírja, hogy tippeljünk nagyobbat ha kisebb akkor pedig,hogy kisebbet.Ha pedig eltaláljuk akkor kiírja, hogy mi volt a szám illetve, hogy hány tippelésből tudtuk kitalálni azt.A programnak két fő fájlját fogjuk átnézni,az egyik a MainActivity,ez fog lefutni a program futásakor, ebben van a játék logikája. A másik pedig a az activity-main.xml, amiben a kinézetet tudjuk állítani. Kezdjük az utóbbival.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:background="#000000">

    <TextView
        android:id="@+id/guess"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
```

```
        android:layout_marginStart="152dp"
        android:layout_marginTop="172dp"
        android:text="Guess"
        android:textColor="#ffffffff"
        android:textColorHighlight="#4E8A1E"
        android:textSize="36sp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<EditText
    android:id="@+id/Guessed"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="96dp"
    android:layout_marginTop="280dp"
    android:ems="10"
    android:gravity="center_horizontal|fill_horizontal"
    android:inputType="number"
    android:textColor="#ffffffff"
    android:textColorHighlight="#00FFFFFF"
    android:textColorHint="#00FFFFFF"
    android:textColorLink="#00FFFFFF"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<Button
    android:id="@+id/ENTER"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="160dp"
    android:layout_marginTop="340dp"
    android:gravity="center_horizontal|center_vertical"
    android:text="ENTER"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Ebben vannak definiálva a gombok illetve szöveges felületek. Az első TextView fogja kiírni, hogy feljebb vagy lejjebb tippeljünk. A felületre az id-val lehet majd hivatkozni, aminek tetszőleges nevet tudunk megadni. A wrap content azt határozza meg, hogy a felület mérete a tartalmához fog igazodni. A textColorral a szöveg színét tudjuk beállítani. A következő az EditText, ami egy olyan mező, amibe írni tudunk. Tehát ha belekoppintunk akkor felugrik a billentyűzet és írhatunk bele. Itt is az id azonosít. Ezután egy gomb azaz Button van definiálva, amit majd az OnClickListenerrel fogunk figyelni. A text sornál adhatjuk meg, hogy mi legyen a gombra írva. Most nézzük a MainActivity fájlt.

```
package com.example.mylocation.gueesthenumber;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.text.Editable;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.util.Random;

public class MainActivity extends AppCompatActivity implements View.OnClickListener {

    public static final int MAX_NUMBER= 100;
    public static final Random RANDOM = new Random();
    private TextView msgTv;
    private EditText numberEnteredEt;
    private Button validate;
    private int numberToFind, numberTries;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        msgTv = (TextView) findViewById(R.id.guess);
        numberEnteredEt = (EditText) findViewById(R.id.Guessed);
        validate = (Button) findViewById(R.id.ENTER);
        validate.setOnClickListener(this);

        newGame();
    }

    @Override
    public void onClick(View view) {
        if (view == validate) {
            validate();
        }
    }

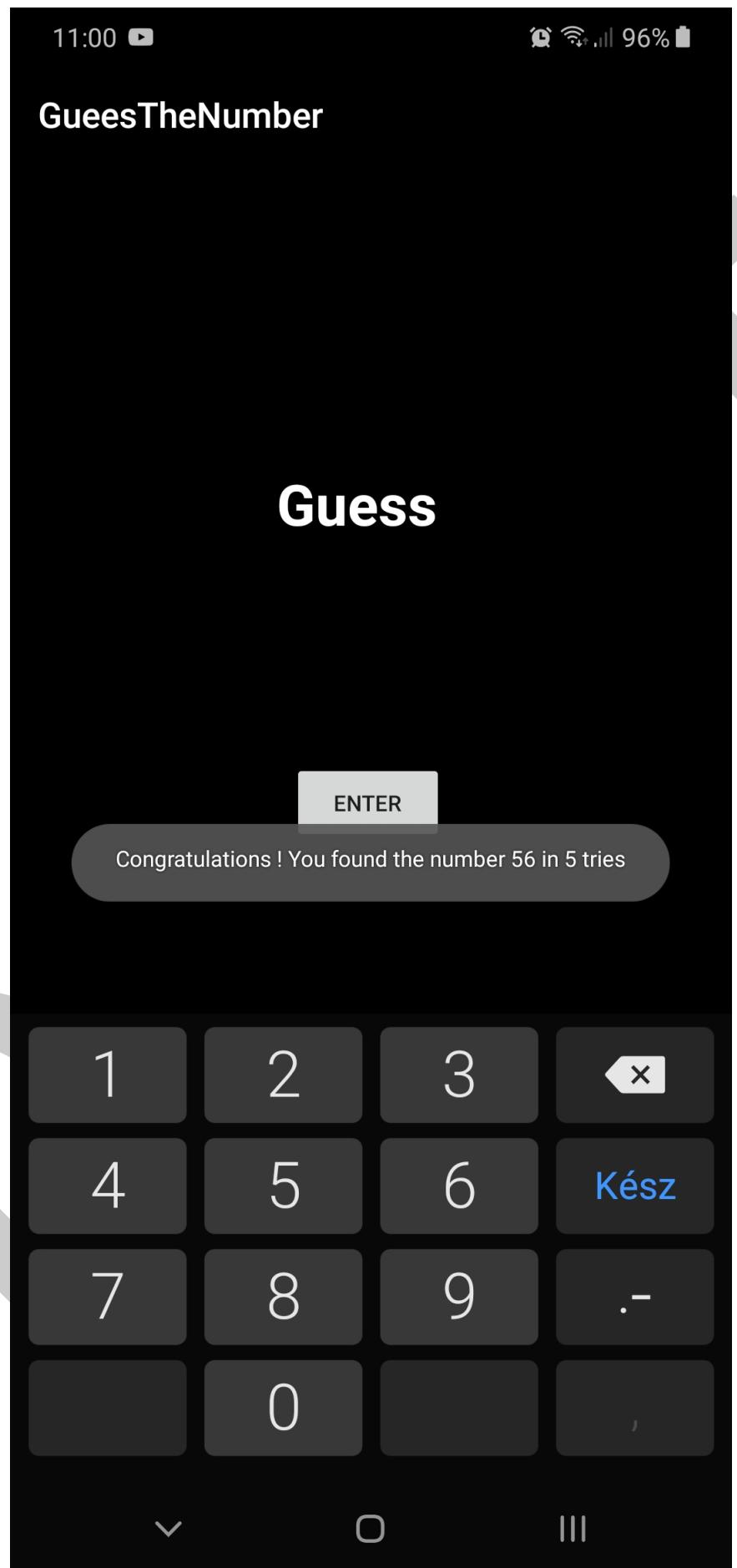
    private void validate() {
        int n = Integer.parseInt(numberEnteredEt.getText().toString());
        numberTries++;

        if (n == numberToFind) {
            Toast.makeText(this, "Congratulations ! You found the number " + numberToFind + "
```

```
        " in " + numberTries + " tries", Toast.LENGTH_SHORT) . ↵
        show() ;
    newGame() ;
} else if (n > numberToFind) {
    msgTv.setText("Guess lower");
} else if (n < numberToFind) {
    msgTv.setText("Guess higher");
}
}

private void newGame() {
    numberToFind = RANDOM.nextInt(MAX_NUMBER) + 1;
    msgTv.setText("Guess");
    numberEnteredEt.setText("");
    numberTries = 0;
}
}
```

A fájl elején deklaráljuk a változóinkat. Először is szükség lesz egy MAX NUMBER re ez fogja meghatározni a random számunk max értéket. Ezután példányosítunk egy random változót. Majd létrehozunk 3 változót amikkel majd a gombok illetve szöveges felületeket tudjuk majd kezelni. Majd ezen kívül még két integert, az egyikben a random számot fogjuk tárolni, a másik pedig a tippek számát fogja tárolni. Először az OnCreate fog lefutni a program futása során, ahol is megszabjuk a kinézetet, létrehozzuk a View-kat. Majd a végén meghívjuk a newGame függvényt, ami elindítja a játékot. Az onClick metódus fogja figyelni a gombokat, és ha rákattintunk a gombra akkor meghívódik a validate függvény ami az ellenőrzést fogja végezni. A validate metódus először eltárolja magának a az ellenőrizendő számot. Ezután növeli a tippek számát. Majd megnézi, hogy egyenlő a megadott szám a kitalálandó szám akkor kiírja, hogy Congratulations és a számot, meg a tippek számát. Ha a kapott szám nagyobb mint a tényleges szá akkor kiírja, hogy tippelj kisebbet, ha pedig kisebb akkor, hogy tippelj nagyobbat. A newGame metódus a játék indításért felelős, létrehozza a randomszámot, és nullázz a tippek számát.



19.4. AOP

Ebben a feladatban az AspectJ-vel fogunk foglalkozni. A binfa programba fogunk úgy változtatni,hogy magába a forrásba nem változtatunk semmit.Az AspectJ egy aspektus orientált programozási nyelv.Java nyelvvel működik,maga a forrása olyan mint egy java class.

Én ebben a feladatban a kiir függvényt változtattam meg.A megváltoztatott függvény mostmár inoderesen fogja kiírni a fánkat,azaz először a baloldali gyereket majd a jobboldalit majd a gyökeret.Nézzük a kódot.:

```
privileged aspect Aspect {
void around (LZWBInFa.Csomopont elem, java.io.PrintWriter os,LZWBInFa f) :
call(public void LZWBInFa.kiir(LZWBInFa.Csomopont, java.io.PrintWriter )) ←
&& args(elem,os) && target(f)

{
if (elem != null) {
    ++f.melyseg;

    if (elem != null) {
++f.melyseg;

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

}

}

}
```

Először is láthatjuk,hogy class helyett egy aspektust adunk meg,és ez privileged lesz,ez azt jelenti,hogy hozzáfog férni a binfa privát változóihoz.Az around jelenti azt,hogy ha meghívódik a kiir függvény akkor a sima kiir helyett, az ebben a fájlban megadott kiírás fog lefutni.Az around után adjuk meg magát a függvényt,amit helyettesíteni akarunk.Ez egy pointcut lesz.Az args-ban adjuk meg,az argumentumok típusát és számát,csak akkor fog lefutni ha ez megegyezik a hívott függvény argumentumaival. A targets-el adjuk meg hogy a binfára hajtódjon végre a helyettesítés. Nézzük a fordítást és a futtatást:

futtatás:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
foldesizoltan@Aspire:~/Asztal/prog2/arroway/binfa$ ajc LZWBinFa.java Aspect.aj
foldesizoltan@Aspire:~/Asztal/prog2/arroway/binfa$ java -classpath ./aspectjrt.jar: LZWBinFa testz.txt -o ki2.txt
foldesizoltan@Aspire:~/Asztal/prog2/arroway/binfa$
```

Megnyitás ki1.txt -/Asztal/prog2/arr... Mentés ⋮ ⌂ ⌂ ⌂

<pre>-----1(2) -----0(3) -----1(1) -----1(3) -----1(5) -----1(7) -----0(6) -----0(4) -----1(7) -----1(9) -----1(11) -----0(10) -----0(11) -----0(12) -----0(8) -----1(6) -----0(5) -----0(6) -----0(7) -----1(9) -----1(11) -----0(10) -----0(8) -----0(2) -----1(5) -----1(7) -----0(8) -----0(6) -----1(4) -----0(3) -----1(5) -----0(6) -----0(4) -----0(5) -----1(7) -----0(6) ---/(0) -----1(4) -----1(3) -----0(4) -----1(2) -----1(4) -----0(5) -----1(8) -----1(10) -----0(11) -----0(12)</pre>	<pre>-----1(17) -----0(16) -----1(15) -----0(14) -----1(15) -----0(14) -----1(23) -----0(22) -----0(21) -----0(21) -----0(24) -----1(23) -----0(22) -----1(23) -----1(22) -----0(21) -----0(21)</pre>
---	---

ki2.txt Mentés ⋮ ⌂ ⌂ ⌂

kiegészítés: A képen két állományt láthatunk ki1.txt és ki2.txt. Az ki1.txt fájlban a sorok között minden előző sorhoz hozzájárultak a következők: -----1(2), -----0(3), -----1(1) stb. Ez azt jelenti, hogy minden előző sorhoz hozzájárultak a következők: -----1(17), -----0(16), -----1(15) stb. Ez azt jelenti, hogy minden előző sorhoz hozzájárultak a következők: -----1(23), -----0(22), -----0(21) stb.

20. fejezet

Helló, Calvin

20.1. Minecraft Malmö

Ebben a feladatban a Minecraft Malmövel fogunk foglalkozni. A malmö a minecraft-hoz készített AI kutatást segítő platform. Először is telepítenünk kell a malmöt. Van lehetőségünk előre buildelt verziót feltelepíteni, én ilyet telepítettem fel, windows oprendszerre. A malmön kívül szükségünk lesz egy vele kompatibilis python-ra is. Majd a pythonhoz a future-t is fel kell telepítenünk. Ezekben kívül szükség van még az AdoptOpenJDK-ra, CodeSynthesis-re és FFmpeg-re. Ha felvannak telepítve, utána be kell állítanunk a környezeti változókat. Az UDP.Room facebookos csoportba belinkelt telepítési tutorial nagyon sok segítség volt. Ha mindezek megvoltak akkor futnia kell a programnak. Futtatáshoz be kell mennünk a minecraft mappába, majd powershellbe ki kell adni a ./launchClient.bat parancsot és megjelenik a főmenü. Ezután egy másik powershellben tudunk majd küldetéseket futtatni. A Python_examples mappában számos példát találunk. Nézzük is meg az alapokat.

Maga a küldetés egy while ciklusban fog futni, itt adhatunk meg Stevenek parancsokat, mozogjon előre, forduljon... stb.

```
agent_host.sendCommand(" ")
```

Egy ilyen sor fog neki megadni egy parancsot, az idézőjelek közé a következő utasítások mehetnek.

```
move [-1,1]
strafe [-1,1]
pitch [-1,1]
turn [-1,1]
jump 1/0
crouch 1/0
attack 1/0
use 1/0
```

A move-al teljes sebességgel fog mozogni, a strafe-el oldalirányba tudunk mozogni, a pitch-el a kamerát tudjuk fel le mozgatni. A turn-el fordulni tudunk jobbra balra. A jump-al ugrik, crouch-al "guggol". Az attack-al ütni fog, a use-al pedig letesz egy blokkot vagy pedig használ valamit. minden küldetéshez tartozik egy

XML fájl, ami a küldetés világát fogja leírni. Itt van lehetőségünk beállítani a játékbeli időt, időjárást, kezdő pozíciót, világ típusát..stb. Például:

```
<ServerInitialConditions>
<Time>
<StartTime>12000</StartTime>
<AllowPassageOfTime>false</AllowPassageOfTime>
</Time>
</ServerInitialConditions>

<AgentStart>
<Placement x="0" y="56" z="0" yaw="90"/>
</AgentStart>

<Weather>rain</Weather>
```

A fenti kód az időt fogja beállítani, majd azt is beállítjuk, hogy ne teljen az idő. Az alatta lévő pedig Steve kezdőhelyzetét fogja megadni. Az alatta lévővel pedig a csapadékot tudjuk állítani, itt nem tudjuk eldönthetni, hogy eső vagy hó lesz ugyanis az a biomtól függ. Lehetőségünk van még különböző dolgokat generálni, ehhez a következő parancsok állnak rendelkezésre.

```
<DrawCuboid x1, y1, z1, x2, y2, z2, type/>
<DrawLine x1, y1, z1, x2, y2, z2, type/>
<DrawBlock x, y, z, type/> <DrawSphere x, y, z, radius, type/>
<DrawItem x, y, z, type/>
```

Többféleképpen tudunk véget vetni egy küldetésnek. minden küldetés végetér ha Steve meghal. Ezenkívül megadhatunk még időkorlátot, vagy akár azt is, hogy akkor legyen vége egy küldetésnek ha Steve hozzáér egy bizonyos blokkhoz. Például itt azt adtuk meg, hogy akkor legyen vége a küldetésnek ha gyémánt blokkra lép Steve.

```
    <AgentQuitFromTouchingBlockType>
<Block type="diamond_block" />
</AgentQuitFromTouchingBlockType>
```

Lehetőségünk van figyelni Steve környezetét, ehhez a következő kódot fogjuk használni az XML-ben. Ez a tutorial_5-ben található.

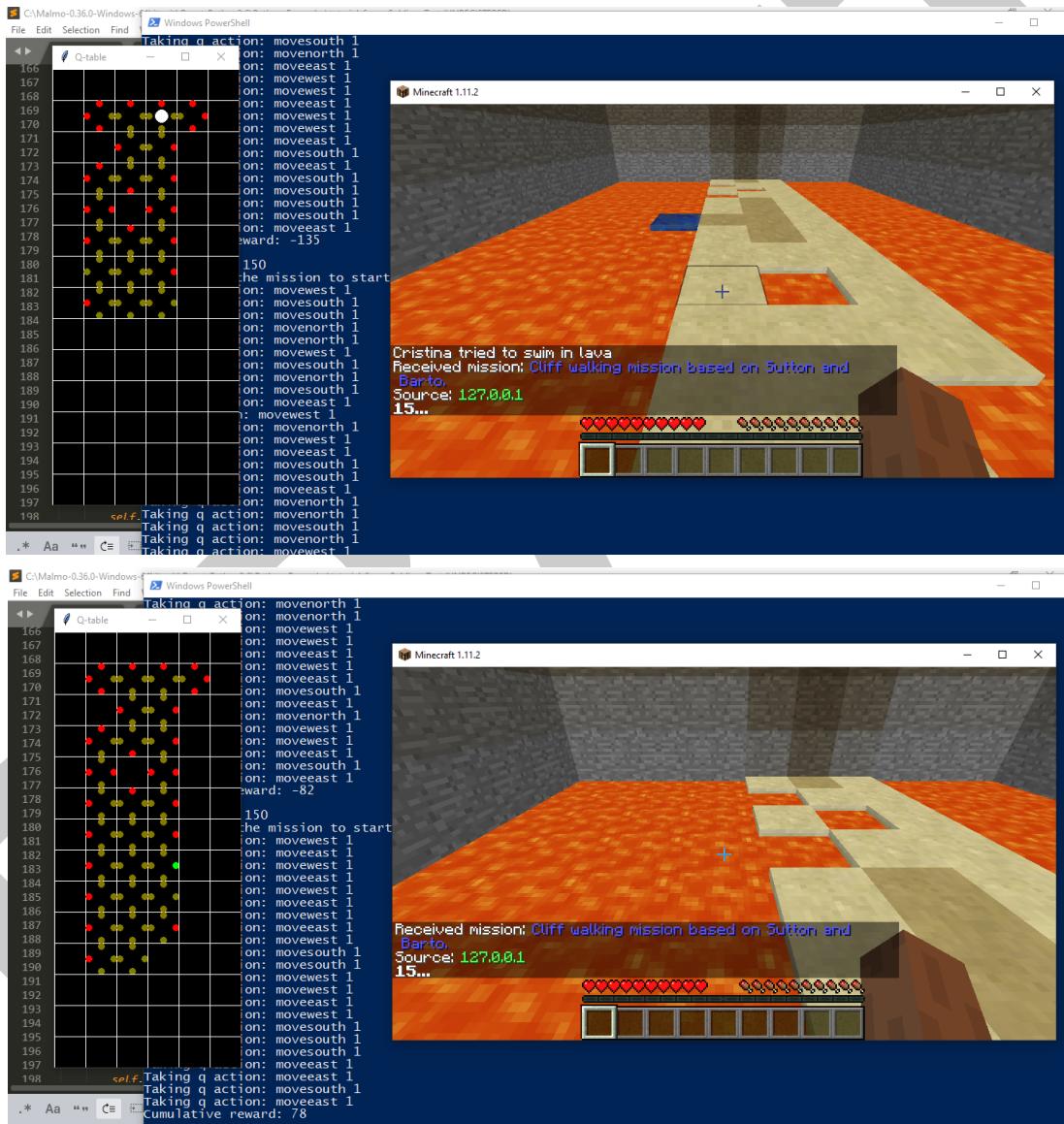
```
        <ObservationFromGrid>
<Grid name="floor3x3">
<min x="-1" y="-1" z="-1"/>
<max x="1" y="-1" z="1"/>
</Grid>
</ObservationFromGrid>
```

Ez a kód a játékos alatt lévő 3x3-as négyzet blokkjait fogja visszaadni. Ezeket egy JSON fájlba fogja kimenteni. Ezekre a blokkokra tudunk majd hivatkozni, ebben a példában arra használja, hogy ha

lávát lát maga előtt akkor ugrik egyet,hogy ne essen bele. A tutorial_6 ban láthatunk egy példát arra,hogy hogyan tudjuk betanítani Stevet,hogy megtalálja a helyes utat egy adott blokkhoz. Ehhez az XML-fájlban megadunk büntetéseket illetve jutalmakat.

```
<RewardForTouchingBlockType>
  <Block reward="-100.0" type="lava" behaviour="onceOnly"/>
  <Block reward="100.0" type="lapis_block" behaviour="onceOnly"/>
</RewardForTouchingBlockType>
```

Tehát láthatjuk,hogyha lávára lép akkor -100 pontot kap ha pedig lapisra, akkor +100-at.Ezáltal a program minél tovább fut,annál kevesebbszer fog lávába lépni.



20.2. MNIST

Ebben a feladatban az MNIST-el fogunk foglalkozni.Ahhoz,hogy működjön a programunk szükségünk lesz a tensorflow-ra. A TensorFlow egy ingyenes nyílt forrású csomag,amit adatfolyam programozás-

hoz, neuronhálókhöz, gépi tanuláshoz használnak.

Az MNIST egy adatbázis melyben összesen 70000 kézzel írott számjegyek találhatóak, ezek közül 60000-el fogjuk betanítani a hálót, a mardék 10 ezer pedig tesztelésre használható. Bátfai Norbert tanár úr kódját használtam, azonban ahhoz, hogy működjön egy kicsit át kellett írni, ugyanis már újabb verziója van a tensorflownak.

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

Ezt a sort kellett átírni, meg kellett adni a két y elő, hogy mit is adunk meg. Valamint még a kép kezelésénél kellett még megadni, hogy szürkesklálásan kezelje. Ez itt látható:

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

Saját képet is felfog ismerni a program, ez egy 28x28-as kép lesz. A reading függvény fogja ezt a saját képünket beolvasni, és ide kellett megadni, hogy szürkeskálásan kezelje. Ezután a következő sorokban létrehozzuk a modellet.

```
x = tf.placeholder(tf.float32, [None, 784])

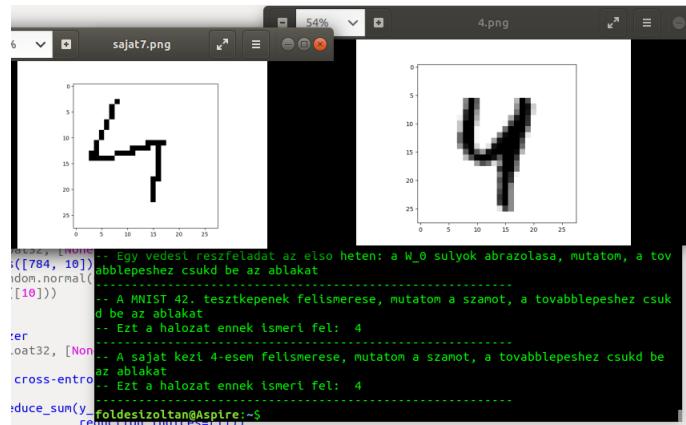
W = tf.Variable(numpy.random.normal(0, 0.001, size=(784, 10)), dtype=tf. ←
    float32)
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
```

Itt az x fogja majd tartalmazni a képünket, a W a súlyt. A súly majd a tanítás során fog beállni. A b a bias rövidítése, ez az előítéletünket fogja tartalmazni, és ez is majd a tanítás során áll be. Az y pedig ebből a három függvényből fog létrejönni. Ezután következik maga a tanítás. A képeket úgy nevezett batchenként fogjuk nézni. Azaz több képet fogunk egyszerre beolvasni, ugyanis egyesével sokkal lassabb lenne.

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

A taníás során a fent látható cross entropy átlagát szeretnénk minimalizálni, ehhez pedig a GradientDescentOptimizert fogjuk használni. Ezt 1000-szer fogjuk elvégezni, tehát 10000-szer használjuk fel a tanító képeket. Ezek után megjelenítjük a felismerni kívánt képet, majd a saját képet, és megnézzük minek ismerte fel őket.

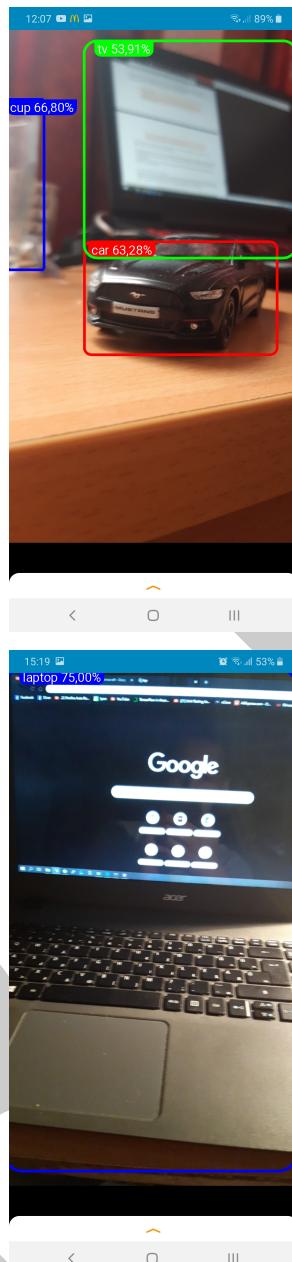
futtatás:

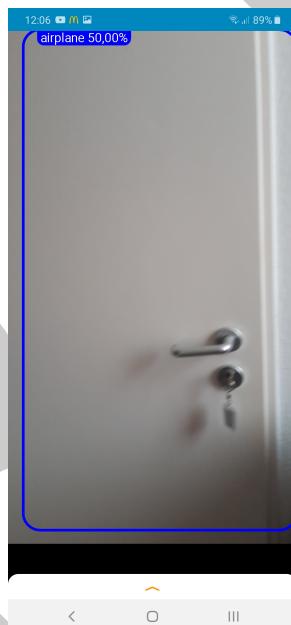


20.3. TF telefonra

Ebben a feladatban az androidra készült tensorflows tárgyfelismerőt kellett kipróbálni. Ez realtime fog tárgyat felismerni több kevesebb sikkerrel. Jöttek létre érdekes felismerések.









DRAFT

IV. rész

Irodalomjegyzék

DRAFT

20.4. Általános

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

20.5. C

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

20.6. C++

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

20.7. 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.