

Univerzális programozás

Programozás haladóknak

Ed. BHAX, DEBRECEN,
2019. Május 09, v. 1.0.0

Copyright © 2019 Rácz András István

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	Rácz, András István	2019. december 13.	

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	2019-03-04	Második fejezet befejezése	randras
0.0.5	2019-03-11	Harmadik feladatsor befejezése.	randras
0.0.6	2019-03-18	Negyedik feladatsort megoldva.	randras
0.0.7	2019-03-25	Ötödik fejezet befejezve.	randras
0.0.8	2019-04-01	Hatodik fejezet megoldva.	randras

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-04-08	Hetedik feladatsor megoldva.	randras
0.1.0	2019-04-22	Nyolcadik fejezet befejezve.	randras
0.1.1	2019-04-29	Kilencedik feladatsor megoldva.	randras
0.1.2	2019-05-06	Olvasónaplo befejezése.	randras
1.0.0	2019-04-09	Könyv befejezése.	randras

Ajánlás

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

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

Tartalomjegyzék

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?	7
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	11
2.6. Helló, Google!	11
2.7. 100 éves a Brun téTEL	13
2.8. A Monty Hall probléma	13
3. Helló, Chomsky!	15
3.1. Decimálisból unárisba átváltó Turing gép	15
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	16
3.3. Hivatalos nyelv	17
3.4. Saját lexikális elemző	17
3.5. l33t.l	18
3.6. A források olvasása	20
3.7. Logikus	21
3.8. Deklaráció	22

4. Helló, Caesar!	24
4.1. Double ** háromszög mátrix	24
4.2. C EXOR titkosító	26
4.3. Java EXOR titkosító	27
4.4. C EXOR törő	28
4.5. Neurális OR, AND és EXOR kapu	31
4.6. Hiba-visszaterjesztéses perceptron	32
5. Helló, Mandelbrot!	33
5.1. A Mandelbrot halmaz	33
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztálytalálkozásával	36
5.3. Biomorfok	39
5.4. A Mandelbrot halmaz CUDA megvalósítása	41
5.5. Mandelbrot nagyító és utazó C++ nyelven	44
5.6. Mandelbrot nagyító és utazó Java nyelven	44
6. Helló, Welch!	49
6.1. Első osztályom	49
6.2. LZW	52
6.3. Fabejárás	56
6.4. Tag a gyökér	58
6.5. Mutató a gyökér	65
6.6. Mozgató szemantika	71
7. Helló, Conway!	79
7.1. Hangyszimulációk	79
7.2. Java életjáték	80
7.3. Qt C++ életjáték	86
7.4. BrainB Benchmark	86
8. Helló, Schwarzenegger!	87
8.1. Szoftmax Py MNIST	87
8.2. Mély MNIST	87
8.3. Minecraft-MALMÖ	87

9. Helló, Chaitin!	92
9.1. Iteratív és rekurzív faktoriális Lisp-ben	92
9.2. Gimp Scheme Script-fu: króm effekt	93
9.3. Gimp Scheme Script-fu: név mandala	96
10. Helló, Gutenberg!	100
10.1. Programozási alapfogalmak	100
10.2. Programozás bevezetés	100
10.3. Programozás	101
III. Második felvonás	102
11. Helló, Arroway!	104
11.1. OO szemlélet	104
11.2. „Gagyí”	107
11.3. Yoda	108
11.4. Kódolás from scratch	108
12. Helló, Liskov!	112
12.1. Liskov helyettesítés sértése	112
12.2. Szülő-gyerek	114
12.3. Anti OO	116
12.4. Hello, Android!	122
12.5. Ciklomatikus komplexitás	124
13. Helló, Mandelbrot!	126
13.1. Reverse engineering UML osztálydiagram	126
13.2. Forward engineering UML osztálydiagram	127
13.3. BPMN	129
13.4. TeX UML	130
14. Helló, Chomsky!	133
14.1. Encoding	133
14.2. l334d1c4	134
14.3. Full screen	136
14.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció	138

15. Helló, Stroustrup!	140
15.1. JDK osztályok	140
15.2. Hibásan implementált RSA törése és összefoglaló összevonva	141
15.3. Változó argumentumszámú ctor	146
16. Helló, Gödel!	157
16.1. Gengszterek	157
16.2. STL map érték szerinti rendezése	157
16.3. Alternatív Tabella rendezése	158
16.4. GIMP Scheme hack	159
17. Helló, !	163
17.1. FUTURE tevékenység editor	163
17.2. OOCWC Boost ASIO hálózatkezelése	164
17.3. SamuCam	164
17.4. BrainB	167
18. Helló, Lauda !	169
18.1. Port Scan	169
18.2. AOP	170
18.3. Android játék	172
18.4. Junit teszt	178
19. Helló, Calvin !	179
19.1. MNIST	179
19.2. Deep MNIST	180
19.3. Android telefonra a TF objektum detektálója	182
19.4. Minecraft Malmö	184
20. Helló, Berners-Lee!	186
20.1. Python élménybeszámoló	186
20.2. C++ és Java összehasonlítása	188
IV. Irodalomjegyzék	190
20.3. Általános	191
20.4. C	191
20.5. C++	191
20.6. Lisp	191

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 más 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

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 kódcsipelek! 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.

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

A végtelen ciklus egy olyan ciklus mely nem áll le mindaddig míg valamilyen külső behatás nem éri. Végtelen ciklus felléphet valamilyen hiba esetén, de sokszor használjuk szándékosn azokat például egy programablak nyitvatartásához. A mi feladatunk az volt, hogy írunk egy végtelen ciklust amely egy szalat 100%-on pörget, egyet amely nulla százalékot használ illetve egy olyat amely minden szálat 100%-on használja.

Egy szál 100 százalékon:

Végtelen ciklust számtalan módon létrehozhatunk például while(1) vagy for(;;). Mi az utobbit fogjuk használni mivel abból egy másik programozó számára is egyértelmű, hogy az általuk létrehozott végtelen ciklus nem egy hiba vagy véletlen eredménye

```
int main ()
{
    for (;;);
    return 0;
}
```

Egy szál 0 százalékon:

```
#include <unistd.h> //Az unistd.h a sleephez szükséges
int main ()
{
    for (;;);
    sleep(1); //A sleep parancs felel a program altatásáért
    return 0;
}
```

Minden szál 100 százalékon:

```
#include <stdlib.h>

int main()
```

```
{  
    #pragma omp parallel //Ez osztja ki a programot az összes szálra ←  
        .Fordításkor gcc loop minden 100.c -o fájlnév -fopenmp  
    for (;;) ;  
    return 0;  
}
```

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

A feladatunk az volt, hogy akkora haxorok legyünk, hogy megírjuk azt programot ami ellenőriz más programokat, hogy futásuk során előfordulhat-e végtelen ciklus amely során az adott program lefagy. Ez természetesen nem lehetséges de tegyük fel, hogy lehetséges és nézük végig az alábbi kódokat. Mint láthatjuk irtunk egy lefagy függvényt ami a megadott programban végtelen ciklust keres, ha megtalálja akkor kiírja, hogy a program lefagy. Eddig nincs baj de ha a kapott program futás során nem lép végtelen ciklusba akkor a mi programunk maga fog végtelen ciklusban ragadni és ennek következtében lefagyni. Ezen ok miatt nem tudunk függvényt írni a végtelen ciklus szűrésére, legalábbis jelenleg.

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

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

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

akár önmagára

```
T100(T100)  
false
```

ezt a kimenetet adja.

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

```
Program T1000
{
    boolean Lefagy(Program P)
    {
        if(P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

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

    main(Input Q)
    {
        Lefagy2(Q)
    }
}
```

2.3. Változók értékének felcserélése

A feladatunk az volt, hogy cseréljünk fel két változót segédváltozó és logikai utasítások nélkül. Mint alább látható ezt egy egyszerű matematikai módszere megoldható.

```
#include <stdio.h>          //ez a printf és a scanf-hez szükséges

int main()
{
    int a = 0;
    int b = 0;
    printf("Adja meg az a szamot: ");
    scanf("%d", &a);
    printf("Adja meg a b szamot: ");
    scanf("%d", &b);
    b = b-a;
    a = a+b;
    b = a-b;
    printf("a=%d%s", a, "\n");
```

```
    printf("b=%d%s",b,"\\n");  
}
```

2.4. Labdapattogás

A feladatunk az volt, hogy standard outputon, labdához hasonlóan, patogtasunk egy karaktert. A könyebb érthetőség érdekében a magyarázat közvetlenül a kód melett lesz kommentek formájában.

Labdapatogás if-el

```
#include <stdio.h> //printf-et tartalmaza  
#include <curses.h> //curses.h szükséges a getmaxyx-hez, a WINDOW *-hoz, az ←  
    initscr-hez, mvprintw-hez és a refresh-hez  
#include <unistd.h> //Ez az usleep-hez szükséges  
  
int  
main ( void )  
{  
    WINDOW *ablak; //létrehozza az ablakot  
    ablak = initscr (); //ez határoza meg a ablak méretét majd át lesz adva ←  
        a getmaxyx-nek  
  
    int x = 0;  
    int y = 0;  
  
    int xnov = 1;  
    int ynov = 1;  
  
    int mx; //Az ablak magassága  
    int my; //Az ablag szélessége  
  
    for ( ; ; ) {  
  
        getmaxyx ( ablak, my , mx ); //az ablak méreteit adja az my-nak ←  
            és az mx-nek  
  
        mvprintw ( y, x, "o" ); // kiírja a labdát  
  
        refresh (); //tényleges kimenetet ad, meg kell hívni hogy ←  
            kimenetet kapjunk  
        usleep ( 50000 ); // a labda sebességét határoza meg a program ←  
            altatásával (az esetünkben 50000 microsscundum)  
        clear();  
  
        x = x + xnov; //a labda vízszintes koordinatája  
        y = y + ynov; //a labda függőleges koordinatája
```

```
    if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
        xnov = xnov * -1;
    }
    if ( x<=0 ) { // elerte-e a bal oldalt?
        xnov = xnov * -1;
    }
    if ( y<=0 ) { // elerte-e a tetejet?
        ynov = ynov * -1;
    }
    if ( y>=my-1 ) { // elerte-e a aljat?
        ynov = ynov * -1;
    }
}
}
```

Labdapatogás if nélkül

```
#include <stdio.h> //printf-et tartalmaza
#include <stdlib.h> //Ez tartalmaza az abs függvényt
#include <unistd.h> //Ez az usleep-hez szükséges

//Az ablak méretnek megadása
#define SZEL 78 //fordításkor SZEL 78-al lesz egyenlő
#define MAG 22 //fordításkor MAG 22-vel lesz egyenlő

int putX(int x,int y) //putX függvény
{
    int i;

    for(i=0;i<x;i++) // a függőleges koordinátát határozza meg
        printf("\n");

    for(i=0;i<y;i++) // a vízszintes koordinátát határozza meg
        printf(" ");

    printf("o\n"); // a Labda

    return 0;
}

int main()
{
    int x=0,y=0;

    while(1)
    {
        system("clear"); // Ez törli az előző labdát az új kiírása előt
        putX(abs(MAG-(x++%(MAG*2))),abs(SZEL-(y++%(SZEL*2)))); // ez határoza ←
            meg a labda pozicióját amihez a putX függvényt hívja meg
    }
}
```

```
    usleep(50000); // a labda sebességét határoza meg a program altatásával
}

return 0;
}
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Olyan programot kellet írnunk amely megadja hogy hány biten tárolunk egy adott integer számot.Ezt a bitek shiftelésével oldotuk meg egy while ciklusban ami addig fut mig el nem éri a nullát.A ciklus minden egyes periodusában növeljük az n-t ami a végén a bitek számát fogja megadni.Ennek megvalósítását lentebb láthatjuk.

```
#include <stdio.h>
int main()
{
    int a=1;
    int n=1;
    while((a<<=1))
    {
        n+=1;
    }
    printf("Megoldas:%d%s",n, "\n");
}
```

2.6. Helló, Google!

A feladatunk az volt, hogy újraalkosuk azt a page rank algoritmust amire a google épült.Az algoritmus feladata az, hogy meghatározza az oldalak presztízsét és az alapján rangsorolja azokat.Egy megadott oldal presztízsét úgy határozza meg, hogy összeszámolja azt, hogy más oldalakról hány link mutat a mi altalunk megadott oldalra.De az oldal pontszámába az is beleszámít, hogy a mi oldalunkra mutató weboldalakra hány link mutat.Miután a programunk végzet az összes megadot oldalal a kapott presztízs pontokat rangsorolja majd kiírja a standard outputra.

PageRank:

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

void kiir (double tomb[], int db) //A kiir függvény felel a kiíratásért.Az ←
    int db adja meg, hogy hányszor fut le a függvényben lévő ciklus.
{
    int i;
```

```
for (i=0; i<db; i++)
    printf("PageRank [%d]: %lf\n", i, tomb[i]);
}

double tavolsag(double pagerank[], double pagerank_temp[], int db)
{
    double tav = 0.0;
    int i;
    for(i=0;i<db;i++)
        if((pagerank[i] - pagerank_temp[i])<0)
    {
        tav +=(-1*(pagerank[i] - pagerank_temp[i]));
    }
    else
    {
        tav +=(pagerank[i] - pagerank_temp[i]);
    }
    return tav;
}

int main(void)
{
    double L[4][4] = {           // L= a link mátrix
    {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}; //Ebben lesz benne a végeredmény
    double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0}; // Az ←
        oldatok presztízs

    long int i,j;
    i=0; j=0;

    for (;;)
    {
        for(i=0;i<4;i++)
            PR[i] = PRv[i]; //Átadja PRv-t a PR-nek
        for (i=0;i<4;i++)
        {
            double temp=0;
            for (j=0;j<4;j++)
                temp+=L[i][j]*PR[j]; //Az L mátrixot összeszorozza a PR vektorral.
            PRv[i]=temp;
        }

        if ( tavolsag(PR,PRv, 4) < 0.00001) //A tavolság függvényt meghívva ←
            határoza meg, hogy mikor áll le a ciklus.
    }
}
```

```
        break;
    }
    kiir (PR, 4); //Meghívjuk a kiir függvényt a kiiratáshoz
    return 0;
}
```

2.7. 100 éves a Brun téTEL

A feladatunk a Brun-tétel demonstrálása volt.A Brun téTEL a prímszámokkal kapcsolatos és nélkülözhetetlen tudni, hogy mik is azok a prím számok és hogy melyeket nevezzük ikerprímeknek. Prímszámoknak nevezzük azokat a számokat amelyeknek pontosan két osztolyuk van, az egy és önmaga.Minden szám felirható prímszámok szorzataira és Euklidész Elemek című munkája óta azt is tudjuk, hogy végtelen prímszám létezik. Ami viszontv még nincs bebizonyítva, hogy az ikerprímek - azok a prím számpárok amelyek különbössége pontosan kettő - száma is végtelen-e. Ezekhez az ikerprímekhez kapcsolódik a Brun-tétel ami azt mondja ki, hogy az ikerprímek reciprokosszege egy Brun-konstans névvel ellátott értékhez konvergál.Ennek demonstrálására készült az alábbi R kód.

```
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")
```

2.8. A Monty Hall probléMA

A Monty Hall-probléMA egy valószínűségi paradoxon.Ez a paradoxon egy amerikai vetélkedőn alapul, amiben a játékosnak három ajtó közül kellett választania.Két ajtó mögött egy-egy kecske volt a harmadik mögött pedig egy autó.A játékosnak rá kellett mutatnia az egyik ajtóra.Ezután a műsorvezető a másik két ajtó közül kinyitotta az egyiket ami mögött a kecske volt (a műsorvezető tudja, hogy melyik ajtó mögöt mi található) és megkérdezte a játékost, hogy akar-e változtatni a döntésén.A Monty Hall-probléMA ezen kérdésből jött létre, hogy érdemes-e a játékosnak változtatni vagy sem, illetve számít-e.Egyeszerű valószínűségszámitással megmutatható, hogy igen, számít mivel elsőnek 3 ajtó közül választhatunk így annak az esélye,

hogy az autót választjuk 1/3, annak pedig, hogy kecskét 2/3. Miután a műsorvezető kinyitotta az egyik ajtót annak a valószínűsége, hogy az általunk választott ajtó mögött van a kocsi továbbra is 1/3. Ezen a ponton viszont a másik két ajtó közül az egyik kinyílt ezért a másik csukott ajtó mögöt 2/3 valószínűséggel a kocsi van. Ez azonban annyira ellentmond a józan észnek, hogy a problémát paradoxonnak tekintjük és ennek a szemléltetésére készült az alábbi kód R-ben

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

for (i in 1:kiserletek_szama) {

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

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

  }else{

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


  }

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


}

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

for (i in 1:kiserletek_szama) {

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


}

valtoztatesnyer = which(kiserlet==valtoztat)

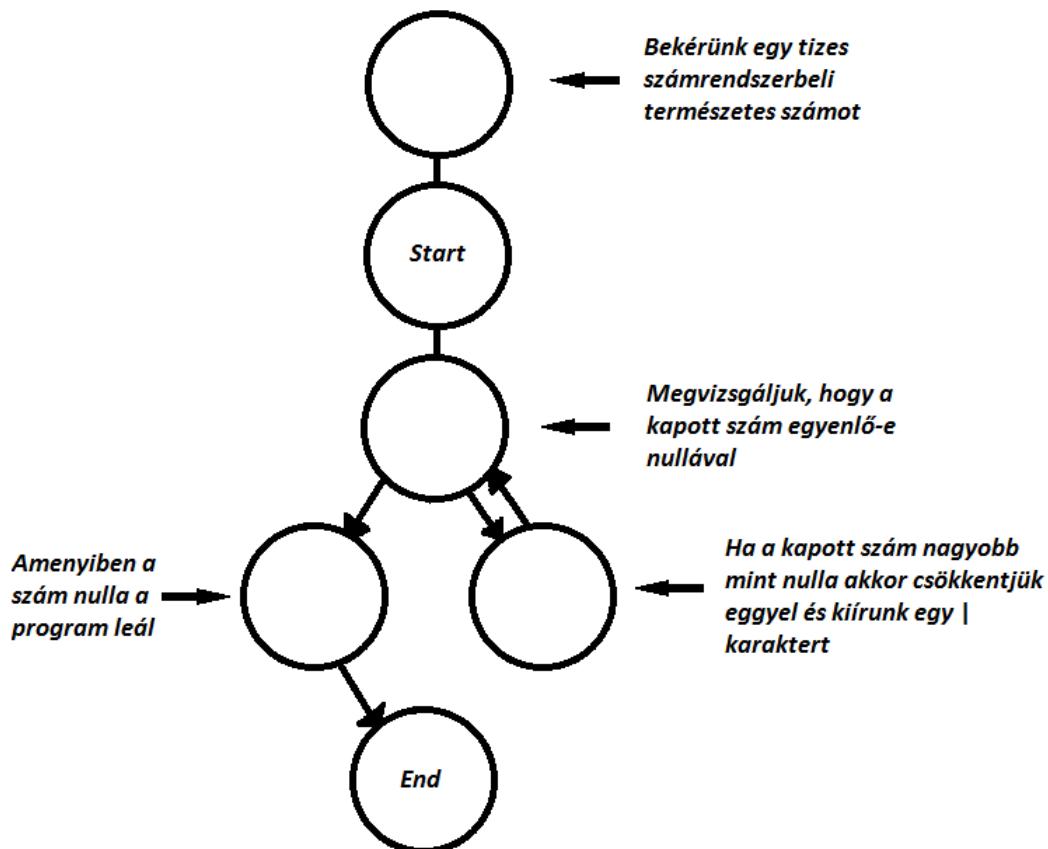
sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Az unáris számrendszer a legegyszerűbb számrendszer.Csak természetes számokat lehet velük ábrázolni.Ha például az 5 számot akarjuk unárisban ábrázolni akkor először is be kell vezetnünk egy szimbólumot amivel az 1 számot foja jelenteni.Nekünk most ez a | karakter lesz.Szóval az 5 unáris számrendszerbe így fog kinézni: |||||.A mi feladatunk az, hogy irunk egy olyan turing gépet ami egy decimális számot unárisá alakit.Ennek a programnak az általapmenet gráfja lejjebb látható.



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

Ebben a feladatban két környezetfüggő generatív grammakat kellett bemutatnunk ami az $a^n b^n c^n$ nyelvet generálja. A generatív nyelvtan elméletét Noam Chomsky amerikai nyelvész alkota meg. Chomsky a generatív nyelveket négy csoportra osztotta, amiből mi a környezetfüggő nyelvtanokkal fogunk foglalkozni. A környezetfüggetlen nyelvtanokkal ellentétben itt a képzési szabályok minden oldalán allhatnak terminális szimbólumok. A szabályokat a \rightarrow jelrel lehet megadni de ez a lentebbi megoldásunkból is látható.

```
S, X, Y "változók"
a, b, c "konstansok"
S → abc, S → aXbc, Xb → bX, Xc → Ybcc, bY → Yb, aY → aaX, aY → aa
S-ből indulunk ki
-----
S (S → aXbc)
aXbc (Xb → bX)
abXc (Xc → Ybcc)
abYbcc (bY → Yb)
aYbcc (aY → aa)
aabcc
-----
S (S → aXbc)
aXbc (Xb → bX)
abXc (Xc → Ybcc)
abYbcc (bY → Yb)
aYbcc (aY → aaX)
aaXbcc (Xb → bX)
aabXbcc (Xb → bX)
aabbXcc (Xc → Ybcc)
aabYbcc (bY → Yb)
aabYbbccc (bY → Yb)
aaYbbbccc (aY → aa)
aaabbccc
```

```
A, B, C "változók"
a, b, c "konstansok"
A → aAB, A → aC, CB → bCc, cB → Bc, C → bc
S-ből indulunk ki
-----
A (A → aAB)
aAB (A → aC)
aaCB (CB → bCc)
aabCc (C → bc)
aabcc
-----
A (A → aAB)
aAB (A → aAB)
aaABB (A → aAB)
aaaABBB (A → aC)
aaaaACBBB (CB → bCc)
aaaaabCccBB (cB → Bc)
```

```
aaaabCBcB (cB -> Bc)
aaaabCBBc (CB -> bCc)
aaaabbCcBc (cB -> Bc)
aaaabbCBcc (CB -> bCc)
aaaabbbCccc (C -> bc)
aaaabbbbcccc
```

3.3. Hivatkozási nyelv

A C egy általános célú programozási nyelv és mint minden nyelv a C is változik, bővül bizonyos dolgokkal. Így ennek következtében előfordulhat az, hogy egy régebbi szabványal nem fog lefordulni a programunk. Erre példa az alábbi for ciklus amit ha c89 szabvánnyal próbálunk lefordítani (ezt a -std=c89 kapcsolával tudjuk megtenni) akkor egy hibaüzeneten belül a fordító közli velünk, hogy mely szabványokkal tudjuk lefordítani a programunkat.

```
//gcc fordule.c -o teszt -std=c89 -el nem fordul le
//gcc fordule.c -o teszt le fog fordulni
int main()
{
    for(int i; i>0; ++i)
    {}
}
```

3.4. Saját lexikális elemző

A feladat az volt, hogy írunk egy olyan lexikális ellemzőt ami felismeri a bemeneten megjelenő valós számokat és megszámolja azokat majd kiírja, hogy pontosan hány darab számot tartalmazott a szöveg. A feladat érdekesége az volt, hogy a lexer lexikális elemzőjét használjuk. Ezzel is szokva, hogy munkánk során gyakran fogjuk más programjait használni, azaz "óriások válnán állni". A kód maga azt csinálja, hogy bekér egy szöveget, majd ha ütünk egy enterrel akkor láthatjuk, hogy a szövegen lévő számokat kiírja először stringként majd float formátumban. Amikor ki akarunk lépni a programból, ez ctrl+D lenyomásaval tehetjük meg, akkor ki fogja írni, hogy a szövegen pontosan hány valós számot találtunk.

```
% { /*
Fordítás:
$ lex -o realnumber.c realnumber.l

Futtatas:
$ gcc realnumber.c -o realnumber -lfl
(kilépés az input vége, azaz Ctrl+D)
*/
#include <stdio.h>
int realnumbers = 0;
%
digit [0-9]
```

```
%%
{digit}*(\.{digit}+)?    {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

3.5. l33t.l

A leet egy olyan írásmód amely szavakban egyes betüket átír más karakterekre (pl.haxor - h4x0r).Amellett, hogy az így leírt szavak sokkal rajosabbak, még hasznuk is van mivel általába az online világban egyre gyakrabban megjelenő AI szűrők nincsenek felkészítve az e módon leírt szavakra.A program hossza ellenére egyáltalán nem bonyolult.Elsőnek szokásosan includeoljuk az includeolnivalokat majd definiáljuk a L337SIZE-ot.Ezután bevezetünk egy c változót egy leet nevű négy elemű tömböt és megadjuk l337d1c7 tömböt amely megadja hogy mely karaktereket mire cserélhetünk ki.A program végső része pedig egy for ciklus ami végig megy a kapott szón és if-ek segítségével eldönti, hogy mit mire cseréljen.

```
/*
Forditas:
$ lex -o l337d1c7.c l337d1c7.l

Futtatas:
$ gcc l337d1c7.c -o l337d1c7 -lfl
(kilépés az input vége, azaz Ctrl+D)
*/
%{
#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"} }  
  
// https://simple.wikipedia.org/wiki/Leet  
};  
  
%}  
%%  
. {  
  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
    {  
  
        if(l337d1c7[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));  
  
            if(r<91)
```

```
    printf("%s", 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;
}
```

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelő)==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 jelkezelő függvény kezelje. (Miután a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)



Bugok

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

i.

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

A legelső kóddal ellentétben a SIGINT jel kezelése nem let figyelmen kívül hagyva.Ezt leszámitva a két kód megegyezik de ez az apró különbözőség is elegendő ahhoz, hogy a két kód egymás ellenetjei legyenek.

ii.

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

Ez egy for ciklus ami nullától indul és egyesével léptetve 5-ig tart.A ++i miatt az i az előtt fog eggyel nőni, hogy bármilyen művelet végrehajtódna.

iii.

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

Ez egy for ciklus ami nullától indul és egyesével léptetve 5-ig tart.A i++ miatt az i az után fog eggyel nőni, hogy bármilyen művelet végrehajtódna.

iv.

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

Itt szintén egy for ciklust láthatunk ami egy tömbben akarjuk 0-tól 4-ig tárolni a számokat.De ez a megvalósítás ilyen formán kerülendő mivel a tomb[i] = i++ első alkalommal nem kerül végrehajtásra a tömb 0 indexű eleme valamilyen memória szemét lesz.

v.

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

Ez egy feltételes for ciklus ahol i kisebb mint n és *d++ megegyezik *s++ -al.Visszont ez a feltétel rosszul van megírva mivel a (*d++=*s++) nem egy logikai értéket fog visszaadni így az and logikai művelet nem lesz értelmezhető.

vi.

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

A printf az f() függvény értékét adja vissza amelynek paraméterei sorrendje nincs megadva.

vii.

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

Két számot iratunk ki az egyik f() által visszadott érték és az a értéke.

viii.

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

A printf, két számot, az a változót és f() függvény által, a memoriacímiből meghatározott értéket írja ki.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

\text = szöveg kiíratása

\forall = univerzális kvantor

\exists = egzisztenciális kvantor

\supset = implikáció

\wedge = konjunkció

\neg = negáció

\vee = diszjunkció

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

Végtelen sok prímszám létezik

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

Végtelen sok ikerprím létezik

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

Véges sok prímszám létezik

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

Véges sok prímszám létezik

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- ```
int a; // integer típusú változó bevezetése
```
- ```
int *b = &a;      // egy pointer ami a memoriacímére mutatt
```
- ```
int &r = a; // referencia változó ami megkapja az a értékét
```
- ```
int c[5];        // 5 elemű tömb ami integer értékeket tárol
```
- ```
int (&tr)[5] = c; // egészek töbjének referenciája
```
- ```
int *d[5];        //egészre mutató mutatók tömbje
```
- ```
int *h (); // egészre mutató mutatót visszaadó függvény
```
- ```
int *(*l) ();      //egészre mutató mutatót visszaadó függvényre mutató ↔ mutató
```

- ```
int (*v (int c)) (int a, int b) // egészet visszaadó és két ↪
 egészet kapó függvényre mutató mutatót visszaadó, egészet kapó ↪
 függvény
```
- ```
int (*(*z) (int)) (int, int);      //függvénymutató egy egészet visszaadó ↪
    és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó ↪
    függvényre
```

4. fejezet

Helló, Caesar!

4.1. Double ** háromszögmátrix

Ebben a feladatban háromszögmátrixokkal fogunk foglalkozni ezért fontos letisztázni, hogy mi is az a mátrix és mikor beszélünk háromszögmátrixról.A mátrixok m darab sorral és n darab oszloppal rendelkező táblázatok.Ha egy mátrix négyzetes, azaz a sorok és oszlopok száma megegyezik és főátlója alatt vagy felett csupa 0 elem található, akkor alsó vagy felső háromszög mátrixról beszélünk. Az alsó háromszögmátrix elemeit sorfolytonosan bejárva el tudjuk helyezni egy tömbben vagy vektorban. A malloc függvénytelével képesek vagunk lefoglalni a dinamikus területen egy, addot méretű területet,amit paraméterként kapunk meg.Végül a lefoglalt területet a free függvény felszabadítjuk.

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

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

    printf("%p\n", &tm);

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

    printf("%p\n", tm);

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

```
}

printf("%p\n", tm[0]);

for (int i = 0; i < nr; ++i)
    for (int j = 0; j < i + 1; ++j)
        tm[i][j] = i * (i + 1) / 2 + j;

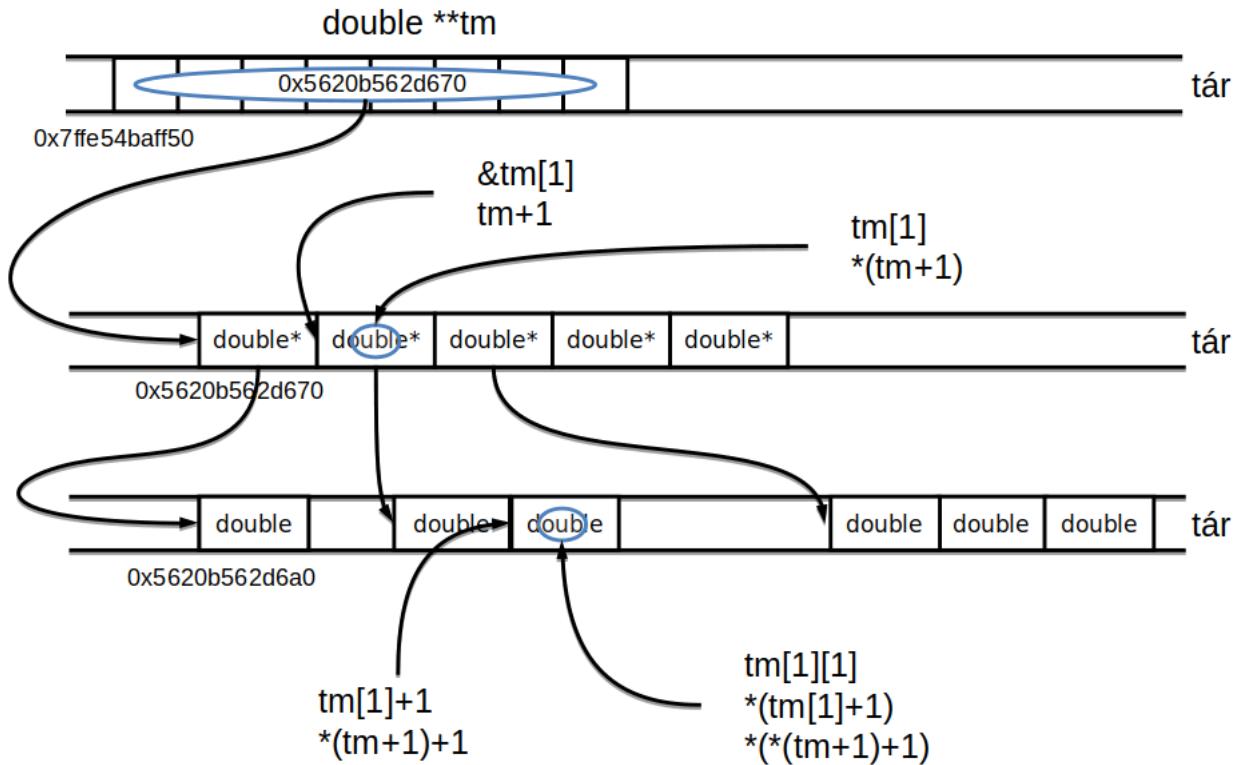
for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
}

for (int i = 0; i < nr; ++i)
{
    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;
}
```



4.2. C EXOR titkosító

A feladatunk az volt, hogy készítsünk egy EXOR-os titkosítót C-ben. A program maga egy fájlból kapott szöveget fog titkosítani amit aztán kiír egy cél fájlba. A kódot fordítás után(gcc titkosito.c -o titkosito) úgy lehet futtatni Linux terminálban, hogy a következő sort begépeljük:

```
./titkosito kód < titkositandó.szöveg > titkosított.szöveg
```

A futtáshoz szükséges parancssorban észrevehetjük, hogy szükségünk lesz egy bemeneti fájlra ami tartalmazza a titkosítani kívánt szöveget, egy kimeneti fájlra ahova majd a titkosított szöveg fog kerülni illetve egy kódra ami alapján titkosítani fogunk. Az általunk megadott kóddal úgy fog titkosítani, hogy a kód bitjeit kizáró vaggyal (EXOR-al) összeküti. Az így létrejőt szöveget pedig kiírja a megadott output fájlunkba.

A kódban láthatjuk, hogy a szokásos includeok után láthatunk két definet az egyik a maximális kulcsmérettől a másik a buffer méretét rögzíti. A main függvényben láthatunk pár változó deklarálást illetve egy while ciklust. Utóbbinak a feladata, hogy a kapott szöveg bájtjait a kulcsal elshiftelje. Az így kapott szöveget végül kiírjuk egy fájlba.

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

int
```

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

4.3. Java EXOR titkosító

Az előző feladatban megírt C kizáro vagyos titkositonkat át fogjuk írni java-ba. A program működési elve úgyan az, azt leszámítva, hogy itt a bemeneti fájl helyett a standard inputon keresztül visszük be a titkosítandó szöveget.

A kód attékintése után láthatjuk, hogy itt is egy kulccsal fogjuk bájtonként elshiftelni a szöveget és így fog létrejönni a titkosított szövegünk. Ez ugy fogjuk megtenni, hogy indítunk egy while ciklust ami adig fog tartani ameddig az éppen bejövő szövegen végig nem ment bájtonként.

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcsSzöveg,
                          java.io.InputStream bejövőCsatorna,
                          java.io.OutputStream kimenőCsatorna)
        throws java.io.IOException {

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

```
int olvasottBájtok = 0;

while((olvasottBájtok =
bejövőCsatorna.read(buffer)) != -1) {

for(int i=0; i<olvasottBájtok; ++i) {

buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
kulcsIndex = (kulcsIndex+1) % kulcs.length;

}

kimenőCsatorna.write(buffer, 0, olvasottBájtok);

}

}

public static void main(String[] args) {

try {

new ExorTitkosító(args[0], System.in, System.out);

} catch(java.io.IOException e) {

e.printStackTrace();

}

}

}
```

4.4. C EXOR törő

Az előző két feladatban titkosítotunk és most a titkositott fájlunkat fogjuk feltörni. A mi EXOR törőnk egyöt, az angol abc betűiből álló kódöt fogja feltörni illetve dekodolja a szöveget. A program végigmegy a kódolt szövegen és egymásba ágyazott ciklusokkal legyártja az összes lehetséges kulcsot és megpróbálja dekódolni a szöveget. A tiszta lehet függvény magyar nyelvben gyakran előforduló szavakat megkeresi és azok alapján próbálja megtippelni, hogy a szöveg tiszta-e. A program működésének részletesebb elemzése a kódban kommentek formájában található meg.

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 5
#define _GNU_SOURCE
```

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

double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
    int sz = 0;
    for (int i = 0; i < titkos_meret; ++i)
        if (titkos[i] == ' ')
            ++sz;

    return (double) titkos_meret / sz;
}

int
tiszta_lehet (const char *titkos, int titkos_meret)
{
    // a tiszta szöveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

    return szohossz > 6.0 && szohossz < 9.0
        && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
        && strcasestr (titkos, "az") && strcasestr (titkos, "ha");

}

void
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{

    int kulcs_index = 0;

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

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

int
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{
```

```
exor (kulcs, kulcs_meret, titkos, titkos_meret);

return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

char kulcs[KULCS_MERET];
char titkos[MAX_TITKOS];
char *p = titkos;
int olvasott_bajtok;
char kod[28] = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o' ←
    ',p','q','r','s','t','u','v','w','x','y','z'};

// titkos fajt berantasa
while ((olvasott_bajtok =
    read (0, (void *) p,
    (p - titkos + OLVASAS_BUFFER <
     MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
    p += olvasott_bajtok;

// maradek hely nullazasa 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 <= 28; ++ii)
    for (int ji = 0; ji <= 28; ++ji)
        for (int ki = 0; ki <= 28; ++ki)
            for (int li = 0; li <= 28; ++li)
                for (int mi = 0; mi <= 28; ++mi)
{
    kulcs[0] = kod[ii];
    kulcs[1] = kod[ji];
    kulcs[2] = kod[ki];
    kulcs[3] = kod[li];
    kulcs[4] = kod[mi];

    if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
    {
        printf
("Kulcs: [%c%c%c%c]\nTiszta szoveg: [%s]\n",
        kod[ii], kod[ji], kod[ki], kod[li], kod[mi], titkos);
        return 0;
    }
}
```

```
// ujra EXOR-ozunk, így nem kell egy masodik buffer  
exor (kulcs, KULCS_MERET, titkos, p - titkos);  
}  
  
return 0;  
}
```

4.5. Neurális OR, AND és EXOR kapu

Az emberi agyban található neuronok hálózata felel az információk feldolgozásáért. A neuron felelős az elektromos jelek fogadásáért, feldolgozásáért és továbbadásáért. Neurális hálók nevezük azt az információfeldolgoző eszközt, amely nagyszámú, hasonló típusú rendelkező adat feldolgozására képes. Illetve tanulási algoritmussal is rendelkezik így képes előhívni a korábban megtanult információkat. Ezt fogjuk tapasztalni a lentebb található R kódban is. A program azt fogja csinálni, hogy az általunk megadot szabályok alapján elkezdi megtanulni az alap logikai műveleteket.

```
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])  
  
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])
```

```
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])


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])
```

4.6. Hiba-visszaterjesztéses perceptron

Ebben a feladatban továbbra is a neurális hálókkal, pontosabban a perceptronokkal fogunk foglalkozni. Ez az algoritmus a számítógépnek megtanítja a binári osztályzást. A bináris osztályzást legegyszerűbben úgy érhetjük meg ha elképzelünk egy vonalat ami fölött fehér, alatta pedig fekete pontok vannak. Ha perceptronnak megadok egy fehéret és egy feketét onnan tól kezdve el fogja tudni dönteni, hogy az adott pont fehér vagy fekete-e. Ezt az osztályzást azért nevezzük bináris osztályzásnak mivel vannak a vonal feletti és a vonal alatti pontok, tehát két választási lehetőségünk van. A mi általunk használ program egy általunk megadot képen fogja végrehajtani a bináris osztályozást.

Megoldás forrása: <https://github.com/nbatfai/nahshon/blob/master/ql.hpp#L64>

5. fejezet

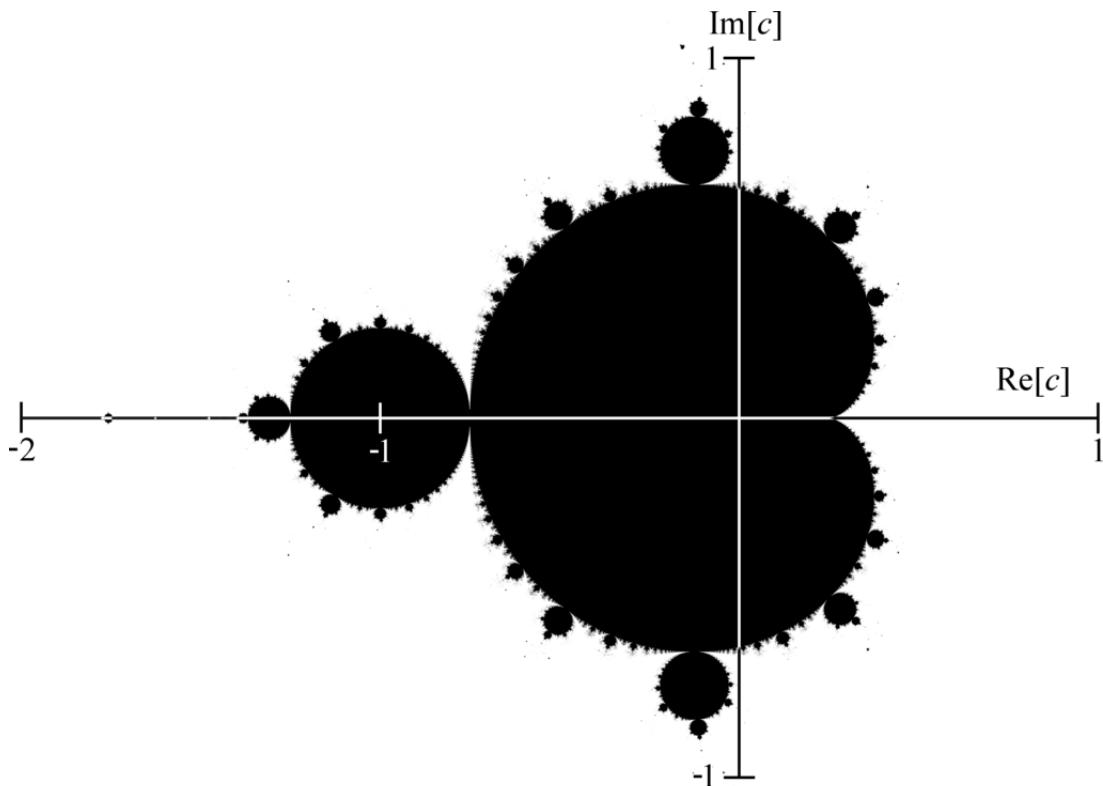
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

A Mandelbrot halmazt Benoit Mandelbrot fedezte fel komplex számsíkon. A komplex számok azok a számok, amelyek megoldást jelenetnek az olyan gyökös kifejezések meghatározásában amikor a gyök alatt negatív szám található. Ezeket a számokat az imaginárius egység segítségével fejezik ki. Imaginárius egységek nevezik azt a komplex számot amelynek négyzete -1. A Mandelbrot-halmaz azokból a komplex számokból áll melyekre az alábbi rekurzív sorozat:

$$x_{n+1} = (x_n)^2 + c$$

nem tart végtelenbe, azaz abszolút értékében korlátos. A Mandelbrot-halmaz a komplex számsíkon ábrázolva, egy nevezetes fraktálalakzat adódik.



A Mandelbrot-halmazt kirajzoló program:

```
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
#define ITER_HAT 32000

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

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

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

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértek, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            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;  
  
}  
  
kepadat[j][k] = iteracio;  
}  
}  
  
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;  
}  
  
int  
main (int argc, char *argv[])  
{  
  
    if (argc != 2)  
    {  
        std::cout << "Hasznalat: ./mandelpng fajlnev";  
        return -1;  
    }  
  
    int kepadat[MERET][MERET];  
  
    mandel(kepadat);  
  
    png::image < png::rgb_pixel > kep (MERET, MERET);  
  
    for (int j = 0; j < MERET; ++j)  
    {  
        //sor = j;  
        for (int k = 0; k < MERET; ++k)  
        {  
            kep.set_pixel (k, j,  
                           png::rgb_pixel (255 -  
                                         (255 * kepadat[j][k]) / ITER_HAT ←  
                                         ,  
                                         255 -
```

```
(255 * kepadat[j][k]) / ITER_HAT ←  
,  
255 -  
(255 * kepadat[j][k]) / ITER_HAT ←  
));  
}  
}  
  
kep.write(argv[1]);  
std::cout << argv[1] << " mentve" << std::endl;  
}
```

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

Ebben a feladatban egy C++ programot írtunk amely meghatározza a Mandelbrot-halmazt az `std::complex` osztály segítségével. A komplex számmolkkal való számításhoz szükséges a `#include <complex>` beincul-deolása. A program gyakorlati megvalósítása lejebb található.

```
// Forditas:  
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2  
// Futtatas:  
// ./3.1.2 mandel.png 1920 1080 2040 ←  
-0.01947381057309366392260585598705802112818 ←  
-0.0194738105725413418456426484226540196687 ←  
0.7985057569338268601555341774655971676111 ←  
0.798505756934379196110285192844457924366  
// ./3.1.2 mandel.png 1920 1080 1020 ←  
0.4127655418209589255340574709407519549131 ←  
0.4127655418245818053080142817634623497725 ←  
0.2135387051768746491386963270997512154281 ←  
0.2135387051804975289126531379224616102874  
// Nyomtatas:  
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1 --left-footer=" ←  
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←  
color  
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf  
  
#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";

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

    for ( int k = 0; k < szelesseg; ++k )
    {

        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam

        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;

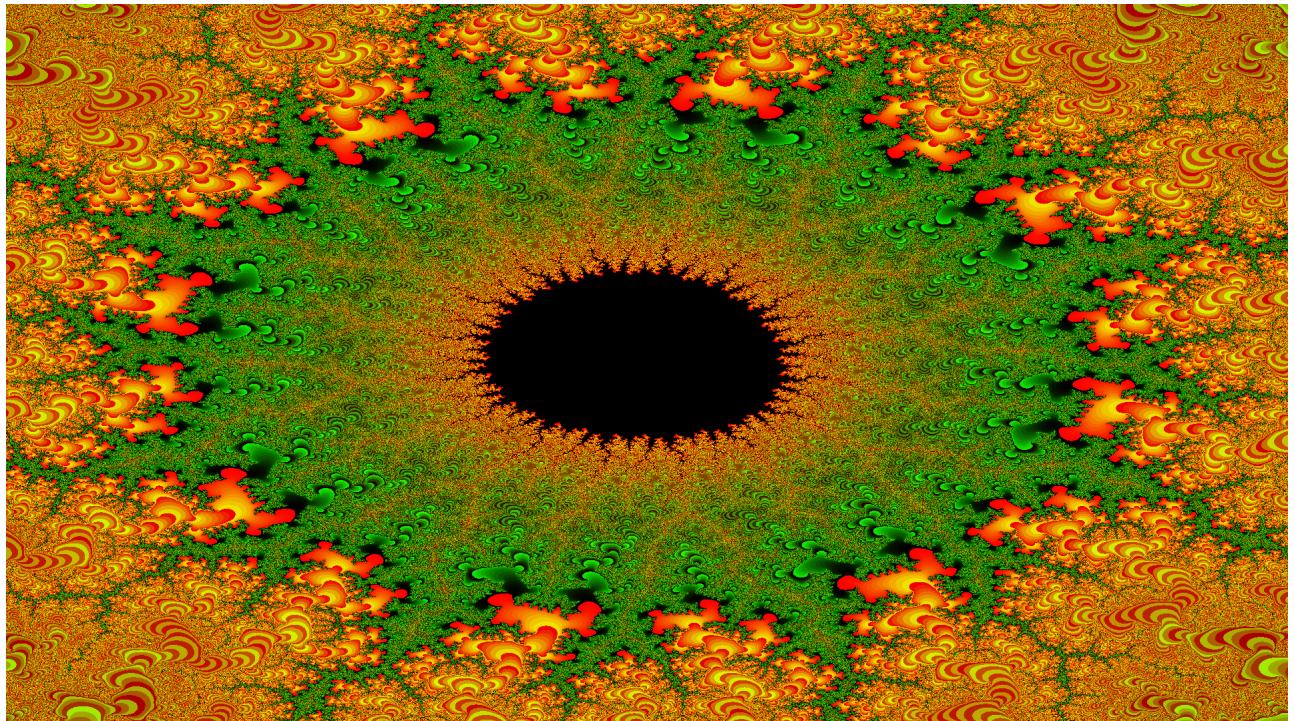
    ++iteracio;
}

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;
}
```

Futatás után, a megadott számuktól függően, ezt vagy egy ehhez hasonló képet kapunk:



5.3. Biomorfok

A biomorfokat Clifford Pickover fedezte fel, amikor a Julia halmazokat rajzoló programjában hiba lépet fel.A különbség a Julia és a Mandelbrot halmazok között, hogy utóbiban a c változó. Ennek egy programban való megvalósításához az előző feladatban elkészített programot átírni mivel eleje teljesen megegyezik így elegendő csak a végét változtatnunk.Ebben az esetben a felhasználótól már a cc konstanst és a küszüöbszámot kérjük be.

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatás:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -l --line-numbers=1 --left-footer="←
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
// color

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

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

double dx = ( xmax - xmin ) / szelessseg;
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 < szelessseg; ++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;
}

}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

Ebben a feladatban az Nvidia CUDA technológiáját fogjuk igénybe venni, amelyel jelentősen fel tudjuk gyorsítani a Mandelbrotos programunk kép generálását.A technológia lényege, hogy a program futását párhuzamosítjuk a CUDA magok között.Ehhez termlézesztesen Nvidia grafikus kártyára van szükséges.

```
#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)
{
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:
    // most eppen a j. sor k. oszlopaban vagyunk

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

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

    // c = (reC, imC) a rács csomópontjainak
    // megfelelő komplex szám
    reC = a + k * dx;
    imC = d - j * dy;
    // z_0 = 0 = (reZ, imZ)
    reZ = 0.0;
    imZ = 0.0;
    iteracio = 0;
    // z_{n+1} = z_n * z_n + c iterációk
```

```
// számítása, amíg |z_n| < 2 vagy még
// nem értük el a 255 iterációt, ha
// viszont elértek, akkor úgy vesszük,
// hogy a kiinduláció komplex számra
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteracionsHatar)
{
    // 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])

```



```
    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;

}
```

5.5. Mandelbrot nagyító és utazó C++ nyelven

A feladatunk az volt, hogy készítsünk egy 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! A mi programunk a QT GUI-t használja, mivel ez az egyik legnépszerűbb grafikus interfézs a C++ nyelvben. Ennek a megoldását itt találhatjuk: https://github.com/AndrasIstvanRacz/Prog1/tree/master/P_K%C3%B6nyv/source/Mandelbrot/MandelMozgatoC%2B%2B

Fordításhoz először fel kell telepítenünk a libqt4-dev-et (sudo apt-get install libqt4-dev). Ez követően gépeljük be sorba a következő utasításokat:

```
qmake -project
qmake -mandelmozgatoc++.pro
make
Futatás:
./mandelmozgatoc++
```

5.6. Mandelbrot nagyító és utazó Java nyelven

Ebben a feladatban az előző C++ programot kellett átírni Javába. Ennek a megoldását itt találhatjuk: https://github.com/AndrasIstvanRacz/Prog1/tree/master/P_K%C3%B6nyv/source/Mandelbrot/NagyitoJava

Fordításhoz először fel kell telepítenünk egy java fordítót amit úgy lehetünk meg, hogy linux terminálba beírjuk a következő parancsot: sudo apt-get install openjdk-8-jdk. Ez követően gépeljük be sorba a következő utasításokat:

```
javac MandelbrotHalmazNagyító.java
```

Futatás:

java MandelbrotHalmazNagyító

```
/*
 * MandelbrotHalmazNagyító.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /**
     * A nagyítandó kijelölt területet bal felső sarka.
     */
    private int x, y;
    /**
     * A nagyítandó kijelölt terület szélessége és magassága.
     */
    private int mx, my;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nyígtani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     *
     * @param a           a [a,b]x[c,d] tartomány a koordinátája.
     * @param b           a [a,b]x[c,d] tartomány b koordinátája.
     * @param c           a [a,b]x[c,d] tartomány c koordinátája.
     * @param d           a [a,b]x[c,d] tartomány d koordinátája.
     * @param szélesség   a halmazt tartalmazó tömb szélessége.
     * @param iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
                                     int szélesség, int iterációsHatár) {
        // Az ős osztály konstruktörának hívása
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        // Egér kattintó események feldolgozása:
        addMouseListener(new java.awt.event.MouseAdapter() {
            // Egér kattintással jelöljük ki a nagyítandó területet
            // bal felső sarkát:
            public void mousePressed(java.awt.event.MouseEvent m) {
                // A nagyítandó kijelölt területet bal felső sarka:
                x = m.getX();
                y = m.getY();
                mx = 0;
                my = 0;
                repaint();
            }
        });
    }
}
```

```
// Vonszolva kijelölünk egy területet...
// Ha felengedjük, akkor a kijelölt terület
// újraszámítása indul:
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;
    // Az új Mandelbrot nagyító objektum elkészítése:
    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);
}
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // A nagyítandó kijelölt terület szélessége és magassága:
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
/**
 * Pillanatfelvételek készítése.
 */
public void pillanatfelvétel() {
    // Az elmentendő kép elkészítése:
    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();
        // A pillanatfelvétel képfájl nevének képzése:
        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("MandelbrotHalmazNagyitas_");
        sb.append(++pillanatfelvételszámláló);
        sb.append("_");
        // A fájl nevébe belelevesszük, hogy melyik tartományban
        // találtuk a halmazt:
        sb.append(a);
        sb.append("_");
        sb.append(b);
        sb.append("_");
        sb.append(c);
        sb.append("_");
        sb.append(d);
        sb.append(".png");
        // png formátumú képet mentünk
        try {
            javax.imageio.ImageIO.write(mentKép, "png",
                new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * A nagyítandó kijelölt területet jelző négyzet kirajzolása.
     */
    public void paint(java.awt.Graphics g) {
        // A Mandelbrot halmaz kirajzolása
        g.drawImage(kép, 0, 0, this);
        // Ha éppen fut a számítás, akkor egy vörös
        // vonallal jelöljük, hogy melyik sorban tart:
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
        // A jelző négyzet kirajzolása:
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
    }
    /**
     * Példányosít egy Mandelbrot halmazt nagyító obektumot.
     */
    public static void main(String[] args) {
        // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
        // tartományában keressük egy 600x600-as hálóval és az
```

```
// aktuális nagyítási pontossággal:  
new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);  
}
```

6. fejezet

Helló, Welch!

6.1. Első osztályom

Ebben a feladatban az volt a dolgunk, hogy kódoljuk le a polártranszformációs algoritmust C++-ban illetve javában. A C++-os verzió három fájlóból fog állni egy main.cpp-ből, egy polargen.cpp-ből és apolargen.h-ból.

A main.cpp-ben először beincludeoljuk az iostreamet illetve az általunk létrehozott polargen.h-t utóbbita később még viszatérünk. A main függvényben declaráljuk a PolarGen osztályú pg változót és egy 1-től 10-ig indít egy for ciklust amiben pg-re meghívja a kovetkezo függvényt és kiírja a pg-ből kapott számot.

```
#include <iostream>
#include "polargen.h"

int
main ()
{
    PolarGen pg;

    for (int i = 0; i < 10; ++i)
        std::cout << pg.kovetkezo () << std::endl;

    return 0;
}
```

A polargen.cpp-be szintén be kell includeoljuk a polargen.h-t. Később majd láthatjuk, hogy erre azért van szükség mivel polargen.h-ban lévő, PolarGen osztályban declaráljuk a kovetkezo függvényt amit itt a polargen.cpp-ben fejtünk ki. Ha a nincsTarolt igaz, azaz nincs tárolt adat akkor a viszatérítési érték r*v1 lesz. Elenkező esetben a tarolt változót fogjuk visszakapni.

```
#include "polargen.h"

double
PolarGen::kovetkezo ()
{
```

```
if (nincsTarolt)
{
    double u1, u2, v1, v2, w;
    do
    {
        u1 = std::rand () / (RAND_MAX + 1.0);
        u2 = std::rand () / (RAND_MAX + 1.0);
        v1 = 2 * u1 - 1;
        v2 = 2 * u2 - 1;
        w = v1 * v1 + v2 * v2;
    }
    while (w > 1);

    double r = std::sqrt ((-2 * std::log (w)) / w);

    tarolt = r * v2;
    nincsTarolt = !nincsTarolt;

    return r * v1;
}
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}
```

Először beincludeoljuk a cstdlib-et, a cmath-ot és a ctime-ot majd declaráljuk a PolarGen osztályt. Deffiniáljuk a nincsTarolt logikai változót igazra, declaráljuk a kovetkezo függvényt és a tarolt double típusú változót.

```
#ifndef POLARGEN__H
#define POLARGEN__H

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

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

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

Illetve végül itt láthatjuk a polárgen programunk javás átírását. A kód működési elve úgyan az mint a c++-os verziójé csak a java letisztult egyszerűsége miatt sokkal tisztább és átláthatóbb. Például a random szám generálása itt jelentősen egyszerűbben történik meg.

```
public class PolárGenerátor {  
    boolean nincsTárolt = true;  
    double tárolt;  
  
    public PolárGenerátor() {  
        nincsTárolt = true;  
    }  
  
    public double következő() {  
        if (nincsTárolt) {  
            double u1, u2, v1, v2, w;  
            do {  
                u1 = Math.random();  
                u2 = Math.random();  
                v1 = 2*u1-1;  
                v2 = 2*u2-1;  
                w = v1*v1+v2*v2;  
            } while (w > 1);  
            double r = Math.sqrt((-2*Math.log(w)) / w);  
            tárolt = r*v2;  
            nincsTárolt = !nincsTárolt;  
            return r*v1;  
        } else {  
            nincsTárolt = !nincsTárolt;  
            return tárolt;  
        }  
    }  
  
    public static void main(String[] arps) {  
        PolárGenerátor g = new PolárGenerátor();  
        for (int i = 0; i < 10; ++i) {  
            System.out.println(g.következő());  
        }  
    }  
}
```

6.2. LZW

A feladatunk az volt, hogy Lempel-Ziv-Welch tömörítési algoritmust - amit Terry Welch amerikai informatikus publikált 1984-ben - írjuk meg C-ben. Ez az algoritmus az adatokat egy fa strukturában ábrázolja. A binfában megadjuk a fa bal és jobb oldali mutatóját. Ez a gyökér jobb és bal gyerekére fog mutatni, majd ez után létrehozzuk a binfa típusra mutatót. Ha az általunk beolvasott érték 0, akkor ez azt jelenti, hogy a fa mutatónak nincs bal oldali gyereke, ezért lefoglalunk neki helyet és nullára állítjuk az értékét. Ezután nullára állítjuk az új gyermeknek a jobb és a bal mutatóját és a fa mutatóját ráállítjuk az új gyökérre. A bal gyermek lesz a gyökér ha a fa bal gyermeke nem 0-ra mutat. A jobb oldali gyermeket 1 esetén vizsgáljuk.

```
//>gcc z.c -lm -o z           fordítása

#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;           // definiáljuk a binfa típust

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;
    int egy_e;
    int i;
```

```
unsigned char c;

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

printf ("\n");
kiir (gyoker);
```

```
extern int max_melyseg, atlagosszeg, melyseg, atlagdb;
extern double szorasosszeg, atlag;

printf ("melyseg=%d\n", max_melyseg - 1);

/* Átlagos ághossz kiszámítása */

atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
ratlag (gyoker);
atlag = ((double) atlagosszeg) / atlagdb;

/* Ághosszak szórásának kiszámítása */

atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
szorasosszeg = 0.0;

rszoras (gyoker);

double szoras = 0.0;

if (atlagdb - 1 > 0)
    szoras = sqrt (szorasosszeg / (atlagdb - 1));
else
    szoras = sqrt (szorasosszeg);

printf ("atlag=%f\nszoras=%f\n", atlag, szoras);

szabadit (gyoker);
}

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);

    // ez a postorder bejáráshoz képest
    // 1-el nagyobb mélység, ezért -1
}
```

```

        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;
    }
}

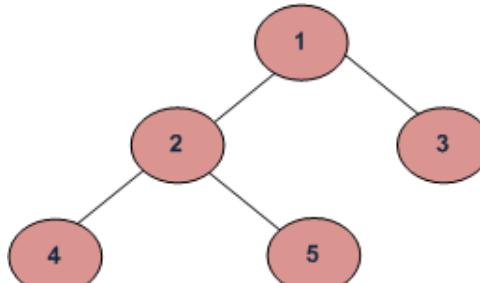
void
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal nulla);
        free (elem);
    }
}

```

6.3. Fabejárás

Egy bináris fának három bejáraási módja van: inorder, postorder, preorder. Az eredeti LZW binfa építőnk a fát inorder módon építi fel.

Inorder bejárás: 4, 2, 5, 1, 3



```

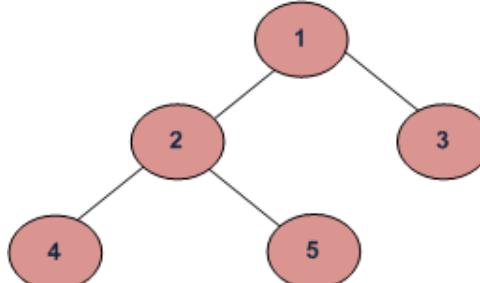
void kiir (Csomopont* elem, std::ostream& os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyernek(), os);
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu() << "(" << melyseg - 1 << ")" << std::endl ↔
        ;
        kiir (elem->>nullasGyernek(), os);
        --melyseg;
    }
}

```

```

    }
}
```

Postorder bejárás: 1, 2, 4, 5, 3



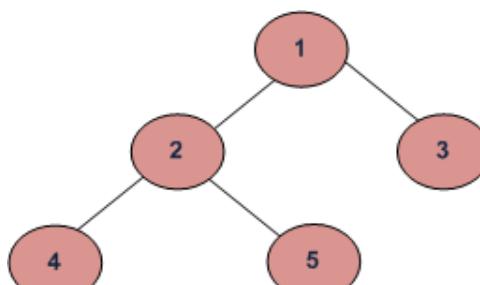
```

void kiir (Csomopont* elem, std::ostream& os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermekek(), os);

        kiir (elem->>nullasGyermekek(), os);
        --melyseg;

        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu() << "(" << melyseg - 2 << ")" << std::endl <-
    ;
    }
}
```

Preorder bejárás: 4, 5, 2, 3, 1



```

void kiir (Csomopont* elem, std::ostream& os)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    // leállítása
    if (elem != NULL)
    {

        for (int i = 0; i < melyseg; ++i)
            os << "---";
```

```
    os << elem->getBetu() << "(" << melyseg - 2 << ")" << std::endl <-
        ;
    ++melyseg;
    kiir (elem->egyesGyermek(), os);
    kiir (elem->>nullasGyermek(), os);
    --melyseg;
}
}
```

6.4. Tag a gyökér

A feladatunk az volt, hogy az LZW algoritmust ültessük át egy C++ osztályba. Az osztály definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ez lesz a beágyazott Csomopont osztály.Csak a fa részeként fogunk számolni vele, egyéb szerepet nem szánunk neki.A a << operátort tagfüggvényként túlterheljük, ezzel fogjuk a fába nyomni az elemeket.Az alapértelmezett paraméter nélküli konstruktur a '/-el hozza létre a csomópontot.A programot futatni ezzel a sorral lehet ./lzwtree be_file -o ki_file, amennyiben valamelyik tag hiányzik a program kiírja, hogy hogyan tudjuk futatni a programot.

```
// z3a7.cpp

#include <iostream>      // mert olvassuk a std::cin, írjuk a std::cout
                        // csatornákat
#include <cmath>         // mert vonunk gyököt a szóráshoz: std::sqrt
#include <fstream>       // fájlból olvasunk, írunk majd

class LZWBInFa          //itt hozzuk létre LZWBInFa osztályt
{
public:

    LZWBInFa () :fa (&gyoker)
    {
    }
    ~LZWBInFa ()
    {
        szabadit (gyoker.egyesGyermek ());
        szabadit (gyoker.nullasGyermek ());
    }

    /* Tagfüggvényként túlterheljük a << operátort amely után így
       nyomhatjuk a fába az inputot: binFa << b; ahol a b
       egy '0' vagy '1'-es betű lesz.*/

    void operator<< (char b)
    {

        if (b == '0')
        {
```

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

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

void kiir (void)           // Ez a függvény felel a kiíratásért.
{
    melyseg = 0;

    kiir (&gyoker, std::cout);
}

{
    szabadit (gyoker.egyesGyermek ());
    szabadit (gyoker.nullasGyermek ());
}

int getMelyseg (void);
double getAtlag (void);
double getSzoras (void);

friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)
{
```

```
        bf.kiir (os);
        return os;
    }
    void kiir (std::ostream & os)
    {
        melyseg = 0;
        kiir (&gyoker, os);
    }

private:
    class Csomopont
    {
public:

    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
    {
    };
    ~Csomopont ()
    {
    };

    Csomopont *nullasGyermek () const
    {
        return balNulla;
    }

    Csomopont *egyesGyermek () const
    {
        return jobbEgy;
    }

    void ujNullasGyermek (Csomopont * gy)
    {
        balNulla = gy;
    }

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

    char getBetu () const
    {
        return betu;
    }

private:
    char betu;
```

```
Csomopont *balNulla;
Csomopont *jobbEgy;
Csomopont (const Csomopont &);
Csomopont & operator= (const Csomopont &);

};

Csomopont *fa;
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
LZWBinFa (const LZWBinFa &);
LZWBinFa & operator= (const LZWBinFa &);

void kiir (Csomopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->>nullasGyermek (), os);
        --melyseg;
    }
}
void szabadit (Csomopont * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek ());
        szabadit (elem->>nullasGyermek ());
        delete elem;
    }
}

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

    void rmelyseg (Csomopont * elem);
    void ratlag (Csomopont * elem);
    void rszoras (Csomopont * elem);

};

int
LZWBinFa::getMelyseg (void)
```

```
{  
    melyseg = maxMelyseg = 0;  
    rmelyseg (&gyoker);  
    return maxMelyseg - 1;  
}  
  
double  
LZWBinFa::getAtlag (void)  
{  
    melyseg = atlagosszeg = atlagdb = 0;  
    ratlag (&gyoker);  
    atlag = ((double) atlagosszeg) / atlagdb;  
    return atlag;  
}  
  
double  
LZWBinFa::getSzoras (void)  
{  
    atlag = getAtlag ();  
    szorasosszeg = 0.0;  
    melyseg = atlagdb = 0;  
  
    rszoras (&gyoker);  
  
    if (atlagdb - 1 > 0)  
        szoras = std::sqrt (szorasosszeg / (atlagdb - 1));  
    else  
        szoras = std::sqrt (szorasosszeg);  
  
    return szoras;  
}  
  
void  
LZWBinFa::rmelyseg (Csomopont * elem)  
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        if (melyseg > maxMelyseg)  
            maxMelyseg = melyseg;  
        rmelyseg (elem->egyesGyermek ());  
        rmelyseg (elem->>nullasGyermek ());  
        --melyseg;  
    }  
}  
  
void  
LZWBinFa::ratlag (Csomopont * elem)  
{  
    if (elem != NULL)
```

```
{  
    ++melyseg;  
    ratlag (elem->egyesGyermek ());  
    ratlag (elem->>nullasGyermek ());  
    --melyseg;  
    if (elem->egyesGyermek () == NULL && elem->>nullasGyermek () == NULL ↔  
        )  
    {  
        ++atlagdb;  
        atlagosszeg += melyseg;  
    }  
}  
}  
  
void  
LZWBinFa::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));  
        }  
    }  
}  
  
void  
usage (void)  
{  
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;  
}  
  
int  
main (int argc, char *argv[])  
{  
    if (argc != 4)  
    {  
        usage ();  
  
        return -1;  
    }
```

```
char *inFile = *++argv;

if ((*((++argv) + 1) != 'o')
{
    usage ();
    return -2;
}

std::fstream beFile (inFile, std::ios_base::in);

if (!beFile)
{
    std::cout << inFile << " nem létezik..." << std::endl;
    usage ();
    return -3;
}

std::fstream kiFile (*++argv, std::ios_base::out);

unsigned char b;
LZWBinFa binFa;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
    if (b == 0x0a)
        break;

bool kommentben = false;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
{
    if (b == 0x3e)
    {
        kommentben = true;
        continue;
    }

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

    if (kommentben)
        continue;

    if (b == 0x4e)
        continue;
```

```
for (int i = 0; i < 8; ++i)
{
    if (b & 0x80)
        binFa << '1';
    else
        binFa << '0';
    b <<= 1;
}

kiFile << binFa;

kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;

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

return 0;
}
```

6.5. Mutató a gyökér

Az előző feladatal elentétben itt a gyökeret nem egy objektumként fogjuk kezelni. Ezt úgy fogjuk megvalósítani, hogy a famutót ráállítjuk a gyökérre a konstruktorban, amit lent hozzunk létre. Ebben az esetben a gyökér is mutató lesz. Ez azt vonja maga után, hogy ahol referencia volt a gyökér ott referencia nélkülivé kell tennünk. Ezen kívül helyet kell foglalnunk a memóriában. Szabadításkor mivel a mutató mutatóit kell elérnünk nyilat fogunk használni.

```
#include <iostream>
#include <cmath>
#include <fstream>

class LZWBinFa
{
public:
    LZWBinFa () {
        gyoker = new Csomopont ('0');
        fa = gyoker;
```

```
};

void operator<<(char b)

{
    if (b == '0')
    {
        if (!fa->nullasGyermek ())
        {
            Csomopont *uj = new Csomopont ('0');
            fa->ujNullasGyermek (uj);
            fa = gyoker;
        }
        else
        {
            fa = fa->nullasGyermek ();
        }
    }
    else
    {
        if (!fa->egyesGyermek ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyesGyermek (uj);
            fa = gyoker;
        }
        else
        {
            fa = fa->egyesGyermek ();
        }
    }
}

void kiir (void)
{
    melyseg = 0;
    kiir (gyoker, std::cout);
}

void szabadit (void)
{
    szabadit (gyoker->egyesGyermek ());
    szabadit (gyoker->nullasGyermek ());
}

int getMelyseg (void);
double getAtlag (void);
double getSzoras (void);

friend std::ostream& operator<< (std::ostream& os, LZWBInFa& bf)
{
    bf.kiir(os);
```

```
        return os;
    }
    void kiir (std::ostream& os)
    {
        melyseg = 0;
        kiir (gyoker, os);
    }

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

    private:
        char betu;
        Csomopont *balNulla;
        Csomopont *jobbEgy;
        Csomopont (const Csomopont &);
        Csomopont & operator=(const Csomopont &);
    };

    Csomopont *fa;
    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;
    LZWBinFa (const LZWBinFa &);
    LZWBinFa & operator=(const LZWBinFa &);

    void kiir (Csomopont* elem, std::ostream& os)
    {
        if (elem != NULL)
        {
            ++melyseg;
            kiir (elem->egyesGyermek(), os);
        }
    }
```

```
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu() << "(" << melyseg - 1 << ")" << std::endl <>
            ;
        kiir (elem->nullasGyermek(), os);
        --melyseg;
    }
}

void szabadit (Csomopont * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek());
        szabadit (elem->>nullasGyermek());
        delete elem;
    }
}

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

    void rmelyseg (Csmopont* elem);
    void ratlag (Csmopont* elem);
    void rszoras (Csmopont* elem);

};

int LZWBInFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (gyoker);
    return maxMelyseg-1;
}
double LZWBInFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (gyoker);
    atlag = ((double)atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBInFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;
    rszoras (gyoker);
```

```
if (atlagdb - 1 > 0)
    szoras = std::sqrt( szorasosszeg / (atlagdb - 1));
else
    szoras = std::sqrt (szorasosszeg);

return szoras;
}
void LZWBinFa::rmelyseg (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek());
        rmelyseg (elem->>nullasGyermek());
        --melyseg;
    }
}
void
LZWBinFa::ratlag (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyesGyermek());
        ratlag (elem->>nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->>nullasGyermek() == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}
void
LZWBinFa::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));
        }
    }
}
```

```
}

void usage(void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    if (argc != 4) {
        usage();
        return -1;
    }

    char *inFile = *++argv;

    if (*((*++argv)+1) != 'o') {
        usage();
        return -2;
    }

    std::fstream beFile (inFile, std::ios_base::in);
    std::fstream kiFile (*++argv, std::ios_base::out);

    unsigned char b;
    LZWBinFa binFa;

    while (beFile.read ((char *) &b, sizeof (unsigned char))) {
        for (int i = 0; i < 8; ++i)
        {
            int egy_e = b & 0x80;
            if ((egy_e >> 7) == 1)
                binFa << '1';
            else
                binFa << '0';
            b <<= 1;
        }
    }

    kiFile << binFa;

    kiFile << "depth = " << binFa.getMelyseg () << std::endl;
    kiFile << "mean = " << binFa.getAtlag () << std::endl;
    kiFile << "var = " << binFa.getSzoras () << std::endl;

    binFa.szabadit ();

    kiFile.close();
```

```
beFile.close();

return 0;
}
```

6.6. Mozgató szemantika

Ebben a feladatban a Tag a gyökér fejezetben használt kódhoz fogunk mozgató konstruktort és értékkopírást. Ennek az a lényege, hogy felesleges másolatok létrehozását nélkül képesek leszünk a binfánkat átmozgatni egyik fájljból a másikba. Ezt a programunk úgy fogja megoldani, hogy a létrehozott binfát átrakja egy másik fájlba majd az eredetit törli. Ennek a gyakorlati megvalósítása a lentebb található kód.

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>

class LZWBInFa
{
public:
    LZWBInFa ()
    {
        gyoker = new Csomopont ('/');
        fa = gyoker;
    }
    ~LZWBInFa ()
    {
        if (gyoker != nullptr)
        {
            szabadit (gyoker->egyesGyermekek ());
            szabadit (gyoker->>nullasGyermekek ());
            delete(gyoker);
        }
    }
    LZWBInFa (LZWBInFa&& eredeti)
    {
        std::cout<<"Move ctor\n";
        gyoker = nullptr;
        *this = std::move(eredeti);

    }
    LZWBInFa& operator= (LZWBInFa&& eredeti)
    {
        std::cout<<"Move assignment ctor\n";
        std::swap(gyoker, eredeti.gyoker);
```

```
        return *this;
    }

    void operator<< (char b)
    {
        if (b == '0')
        {
            if (!fa->nullasGyermek ())
            {
                Csomopont *uj = new Csomopont ('0');
                fa->ujNullasGyermek (uj);
                fa = gyoker;
            }
            else
            {
                fa = fa->nullasGyermek ();
            }
        }
        else
        {
            if (!fa->egyesGyermek ())
            {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }

    void kiir (void)
    {
        melyseg = 0;
        kiir (gyoker, std::cout);
    }

    int getMelyseg (void);
    double getAtlag (void);
    double getSzoras (void);

    friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)
    {
        bf.kiir (os);
        return os;
    }
    void kiir (std::ostream & os)
    {
```

```
melyseg = 0;
    kiir (gyoker, os);
}

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

private:
    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &); //másoló konstruktor
    Csomopont & operator= (const Csomopont &);
};

Csomopont *fa;
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
LZWBinFa (const LZWBinFa &);
LZWBinFa & operator= (const LZWBinFa &);

void kiir (Csomopont * elem, std::ostream & os)
```

```
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        kiir (elem->egyesGyermek (), os);  
        for (int i = 0; i < melyseg; ++i)  
            os << "---";  
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;  
        kiir (elem->>nullasGyermek (), os);  
        --melyseg;  
    }  
}  
void szabadit (Csomopont * elem)  
{  
    if (elem != NULL)  
    {  
        szabadit (elem->egyesGyermek ());  
        szabadit (elem->>nullasGyermek ());  
        delete elem;  
    }  
}  
  
protected:  
    Csomopont *gyoker;  
    int maxMelyseg;  
    double atlag, szoras;  
  
    void rmelyseg (Csomopont * elem);  
    void ratlag (Csomopont * elem);  
    void rszoras (Csomopont * elem);  
  
};  
  
int  
LZWBinFa::getMelyseg (void)  
{  
    melyseg = maxMelyseg = 0;  
    rmelyseg (gyoker);  
    return maxMelyseg - 1;  
}  
  
double  
LZWBinFa::getAtlag (void)  
{  
    melyseg = atlagosszeg = atlagdb = 0;  
    ratlag (gyoker);  
    atlag = ((double) atlagosszeg) / atlagdb;  
    return atlag;  
}
```

```
double
LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (gyoker);

    if (atlagdb - 1 > 0)
        szoras = std::sqrt (szorasosszeg / (atlagdb - 1));
    else
        szoras = std::sqrt (szorasosszeg);

    return szoras;
}

void
LZWBinFa::rmelyseg (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek ());
        rmelyseg (elem->>nullasGyermek ());
        --melyseg;
    }
}

void
LZWBinFa::ratlag (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyesGyermek ());
        ratlag (elem->>nullasGyermek ());
        --melyseg;
        if (elem->egyesGyermek () == NULL && elem->>nullasGyermek () == NULL -->
            )
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}
```

```
void
LZWBinFa::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));
        }
    }
}

LZWBinFa move(LZWBinFa&& eredeti)
{
    LZWBinFa b(std::move(eredeti));
    return b;
}
void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    if (argc != 4)
    {
        usage ();
        return -1;
    }

    char *inFile = *++argv;

    if (*((*++argv) + 1) != 'o')
    {
        usage ();
        return -2;
    }

    std::fstream beFile (inFile, std::ios_base::in);

    if (!beFile)
```

```
{  
    std::cout << inFile << " nem letezik..." << std::endl;  
    usage();  
    return -3;  
}  
  
std::fstream kiFile (*++argv, std::ios_base::out);  
  
unsigned char b;  
LZWBinFa binFa;  
  
while (beFile.read ((char *) &b, sizeof (unsigned char)))  
    if (b == 0x0a)  
        break;  
  
bool kommentben = false;  
  
while (beFile.read ((char *) &b, sizeof (unsigned char)))  
{  
  
    if (b == 0x3e)  
    {  
        kommentben = true;  
        continue;  
    }  
  
    if (b == 0x0a)  
    {  
        kommentben = false;  
        continue;  
    }  
  
    if (kommentben)  
        continue;  
  
    if (b == 0x4e)  
        continue;  
  
    for (int i = 0; i < 8; ++i)  
    {  
        if (b & 0x80)  
            binFa << '1';  
        else  
            binFa << '0';  
        b <<= 1;  
    }  
  
    }  
  
    kiFile << binFa;
```

```
kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;

LZWBinFa binFa2 = std::move(binFa);

kiFile << binFa2;

kiFile << "depth = " << binFa2.getMelyseg () << std::endl;
kiFile << "mean = " << binFa2.getAtlag () << std::endl;
kiFile << "var = " << binFa2.getSzoras () << std::endl;

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

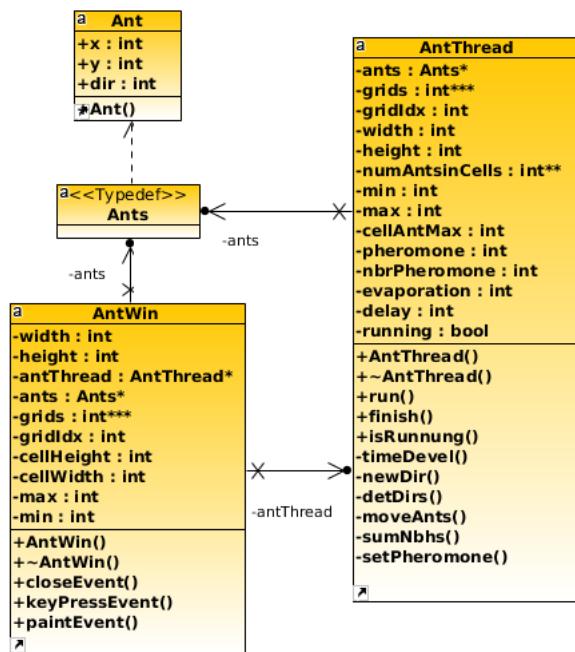
return 0;
}
```

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

Ebben a feladatban a hangya szimulácival fogunk foglalkozni. A szimulció, ahogy a nevéből is kitalálhatjuk a hangyákkal kapcsolatos, mégghozzá a hangyák azon tulajdonságával kapcsolatos, hogy két pont között minden megkeresik a legoptimálisabb utvonalat és miután megtalálták csak azt fogják használni. A mi programunk is ezt fogja csinálni. A mi esetünkben a kis pontok lesznek a hangyák amelyek először a különböző pontokba különböző utakon fognak eljutni. Egy kis idő után azonban észrevehetjük, hogy egy-egy pontba már csak egy vonalon fognak haladni a hangyáink. Ez azt jelenti, hogy megtalálták a legrövidebb utat az adott pontba.



Megoldás forrása: https://github.com/AndrasIstvanRacz/Prog1/tree/master/P_K%C3%B6nyv/source/Conway/Myrmecologist

7.2. Java életjáték

Ebben a feladatban a John Conway-féle életjátékot kellet megvalósítsuk egy java programban azon belül is a sikló-kilövőt.

John Conway, a Cambridge Egyetem egyik matematikusa találta ki az életjátékot. Ez matematikai szempontból az úgynevezett sejtautomaták közé tartozik (Az olyan diszkrét modellek, amiket a számelméletben mikrostruktúrák modellezésére használnak sejtautomatáknak neveznek). A program egy négyzetrácsot fog megjeleníteni. A négyzetrács fekete cellákat sejteknek neveznek. Egy cellához legközelebb lévő 8 cellát a cella környezetének nevezik. Az életjáték körökre van osztva. Az első körben elhelyezünk tetszőleges számú sejtet. Ez után a játékosnak nincs beleszólása a játékmenetbe. Egy sejttel ezután három dolog történhet:

- 1) A sejt túléli a kört, ha két vagy három szomszédja van.
- 2) A sejt elpusztul, ha kettőnél kevesebb (elszigetelődés), vagy háromnál több (túlnépesedés) szomszédja van.
- 3) Új sejt születik minden olyan cellában, melynek környezetében pontosan három sejt található.

Fontos, hogy ezek a változások csak kör végén következnek be. Ez azt jelenti, hogy az elhalálozók nem befolyásolják a túlélést és a születéseket illetve a születők sem befolyásolják az elhalálozókat.

Az életjáték eredményezhet speciális alakzatokat (alkzatoknak nevezik a sejtek egy halmazát), ezek közül mi a sikló-kilövővel fogunk foglalkozni. Ennek az az érdekesége, hogy négy fázis után önmagába alakul vissza miközben egy kockányit minden elmozdul, tehát átlós mozgásal végez. Ez az alakzat a hackerek hivatalos emblémája is.

```
public class Sejtautomata extends java.awt.Frame implements Runnable {  
    public static final boolean ÉLŐ = true;  
    public static final boolean HALOTT = false;  
    protected boolean [][] [] rácsok = new boolean [2] [] [];  
    protected boolean [] [] rács;  
    protected int rácsIndex = 0;  
    protected int cellaSzélesség = 20;  
    protected int cellaMagasság = 20;  
    protected int szélesség = 20;  
    protected int magasság = 10;  
    protected int várakozás = 1000;  
    private java.awt.Robot robot;  
    private boolean pillanatfelvétel = false;  
    private static int pillanatfelvételSzámláló = 0;  
  
    public Sejtautomata(int szélesség, int magasság) {  
        this.szélesség = szélesség;  
        this.magasság = magasság;  
        rácsok[0] = new boolean[magasság] [szélesség];  
        rácsok[1] = new boolean[magasság] [szélesség];  
        rácsIndex = 0;  
        rács = rácsok[rácsIndex];  
        for(int i=0; i<rács.length; ++i)  
            for(int j=0; j<rács[0].length; ++j)  
                rács[i][j] = HALOTT;
```

```
siklóKilövő(rács, 5, 60);
addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent e) {
        setVisible(false);
        System.exit(0);
    }
});

addKeyListener(new java.awt.event.KeyAdapter() {
    public void keyPressed(java.awt.event.KeyEvent e) {
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            cellaSzélesség *= 2;
            cellaMagasság *= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel = !pillanatfelvétel;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
            várakozás /= 2;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
            várakozás *= 2;
        repaint();
    }
});

addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});

addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});
```

```
cellaSzélesség = 10;
cellaMagasság = 10;
try {
    robot = new java.awt.Robot(
        java.awt.GraphicsEnvironment.
        getLocalGraphicsEnvironment().
        getDefaultScreenDevice());
} catch(java.awt.AWTException e) {
    e.printStackTrace();
}

setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
          magasság*cellaMagasság);
setVisible(true);
new Thread(this).start();
}

public void paint(java.awt.Graphics g) {
    boolean [][] rács = rácsok[rácsIndex];
    for(int i=0; i<rács.length; ++i) {
        for(int j=0; j<rács[0].length; ++j) {
            if(rács[i][j] == ÉLŐ)
                g.setColor(java.awt.Color.BLACK);
            else
                g.setColor(java.awt.Color.WHITE);
            g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                      cellaSzélesség, cellaMagasság);
            g.setColor(java.awt.Color.LIGHT_GRAY);
            g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                      cellaSzélesség, cellaMagasság);
        }
    }
}

if(pillanatfelvétel) {
    pillanatfelvétel = false;
    pillanatfelvétel(robot.createScreenCapture
        (new java.awt.Rectangle
            (getLocation().x, getLocation().y,
             szélesség*cellaSzélesség,
             magasság*cellaMagasság)));
}
}

public int szomszédokSzáma(boolean [][] rács,
                           int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 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 = szélesség-1;  
            else if(o >= szélesség)  
                o = 0;  
  
            int s = sor + i;  
            if(s < 0)  
                s = magasság-1;  
            else if(s >= magasság)  
                s = 0;  
  
            if(rács[s][o] == állapot)  
                ++állapotúSzomszéd;  
        }  
  
    return állapotúSzomszéd;  
}  
  
public void időFejlődés() {  
  
    boolean [][] rácsElőtte = rácsok[rácsIndex];  
    boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];  
  
    for(int i=0; i<rácsElőtte.length; ++i) {  
        for(int j=0; j<rácsElőtte[0].length; ++j) {  
  
            int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);  
  
            if(rácsElőtte[i][j] == ÉLŐ) {  
  
                if(élők==2 || élők==3)  
                    rácsUtána[i][j] = ÉLŐ;  
                else  
                    rácsUtána[i][j] = HALOTT;  
            } else {  
  
                if(élők==3)  
                    rácsUtána[i][j] = ÉLŐ;  
                else  
                    rácsUtána[i][j] = HALOTT;  
            }  
        }  
    }  
    rácsIndex = (rácsIndex+1)%2;  
}  
  
public void run() {
```

```
while(true) {  
    try {  
        Thread.sleep(várakozás);  
    } catch (InterruptedException e) {}  
  
    időFejlődés();  
    repaint();  
}  
}  
  
public void sikló(boolean [][] rács, int x, int y) {  
  
    rács[y+ 0][x+ 2] = ÉLŐ;  
    rács[y+ 1][x+ 1] = ÉLŐ;  
    rács[y+ 2][x+ 1] = ÉLŐ;  
    rács[y+ 2][x+ 2] = ÉLŐ;  
    rács[y+ 2][x+ 3] = ÉLŐ;  
  
}  
  
public void siklóKilövő(boolean [][] rács, int x, int y) {  
  
    rács[y+ 6][x+ 0] = ÉLŐ;  
    rács[y+ 6][x+ 1] = ÉLŐ;  
    rács[y+ 7][x+ 0] = ÉLŐ;  
    rács[y+ 7][x+ 1] = ÉLŐ;  
  
    rács[y+ 3][x+ 13] = ÉLŐ;  
  
    rács[y+ 4][x+ 12] = ÉLŐ;  
    rács[y+ 4][x+ 14] = ÉLŐ;  
  
    rács[y+ 5][x+ 11] = ÉLŐ;  
    rács[y+ 5][x+ 15] = ÉLŐ;  
    rács[y+ 5][x+ 16] = ÉLŐ;  
    rács[y+ 5][x+ 25] = ÉLŐ;  
  
    rács[y+ 6][x+ 11] = ÉLŐ;  
    rács[y+ 6][x+ 15] = ÉLŐ;  
    rács[y+ 6][x+ 16] = ÉLŐ;  
    rács[y+ 6][x+ 22] = ÉLŐ;  
    rács[y+ 6][x+ 23] = ÉLŐ;  
    rács[y+ 6][x+ 24] = ÉLŐ;  
    rács[y+ 6][x+ 25] = ÉLŐ;  
  
    rács[y+ 7][x+ 11] = ÉLŐ;  
    rács[y+ 7][x+ 15] = ÉLŐ;  
    rács[y+ 7][x+ 16] = ÉLŐ;  
    rács[y+ 7][x+ 21] = ÉLŐ;  
    rács[y+ 7][x+ 22] = ÉLŐ;
```

```
rács[y+ 7][x+ 23] = ÉLŐ;
rács[y+ 7][x+ 24] = ÉLŐ;

rács[y+ 8][x+ 12] = ÉLŐ;
rács[y+ 8][x+ 14] = ÉLŐ;
rács[y+ 8][x+ 21] = ÉLŐ;
rács[y+ 8][x+ 24] = ÉLŐ;
rács[y+ 8][x+ 34] = ÉLŐ;
rács[y+ 8][x+ 35] = ÉLŐ;

rács[y+ 9][x+ 13] = ÉLŐ;
rács[y+ 9][x+ 21] = ÉLŐ;
rács[y+ 9][x+ 22] = ÉLŐ;
rács[y+ 9][x+ 23] = ÉLŐ;
rács[y+ 9][x+ 24] = ÉLŐ;
rács[y+ 9][x+ 34] = ÉLŐ;
rács[y+ 9][x+ 35] = ÉLŐ;

rács[y+ 10][x+ 22] = ÉLŐ;
rács[y+ 10][x+ 23] = ÉLŐ;
rács[y+ 10][x+ 24] = ÉLŐ;
rács[y+ 10][x+ 25] = ÉLŐ;

rács[y+ 11][x+ 25] = ÉLŐ;

}

public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételszámláló);
    sb.append(".png");
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}

public void update(java.awt.Graphics g) {
    paint(g);
}

public static void main(String[] args) {
    new Sejtautomata(100, 75);
}
```

7.3. Qt C++ életjáték

Ebben a feladatban is John Conway életjátékát fogjuk megvalósítani, csak ezutal C++-ban Qt gui segítsével. A program működési elve úgyan az, illetve a szabályok is megegyeznek az előző feladatban lévőkkel.

Megoldás forrása: https://github.com/AndrasIstvanRacz/Prog1/tree/master/P_K%C3%B6nyv/source/Conway/Gol_Qt_C%2B%2B

7.4. BrainB Benchmark

Ebben a feladatban a Brain Benchmarkkal kellett foglalkoznunk, ami az agy koncentrációs késégét méri. A Brain Benchmark meghatároza, hogy az agyunk milyen gyorsan képes reagálni, menyire képes összpontosítani egy adott feladatra illetve teszteli a memóriát. Az áltunk használt program ezt úgy teszi meg, hogy a képernyőn megjelenít kis négyzeteket bennük egy körrrel. A felhasználó feladata az, hogy válaszon egyet a négyzetek közül és a benne lévő körre kattintson, és nyomva tartva az egeret kövese azt. Eközben újabb és újabb négyzetek fognak megjelenni. Abban az esetben ha az egeret elengedjük vagy az általunk kiválasztott kört elveszítjük a négyzetek elkezdenek eltünni. A program ezen a datok alapján ki fogja számolni a reakcióidőket. Ez az esprtolok körében eléggé elterjedt mivel azon a területen fontos, hogy az agy a lehető leggyorsabban reagáljon.

Megoldás forrása: https://github.com/AndrasIstvanRacz/Prog1/tree/master/P_K%C3%B6nyv/source/Conway/BrainB

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Ezen feladat helyet az SMNIST-et csináltam.

8.2. Mély MNIST

Ezen feladat helyet az SMNIST-et csináltam.

8.3. Minecraft-MALMÖ

A MALMÖ project a microsoft álltal létrehozott olyan platform amely segítségével a mesterséges intelligenciával végezhetünk kutatásokat, méghozá az igen nagy népszerűséget élvező Minecraft játékon keresztül. A megoldáshoz nagy segítséget nyújt a Malmo Platform Tutorial amit a következő címen találhatunk meg https://microsoft.github.io/malmo/0.17.0/Python_Examples/Tutorial.pdf.

A feladat elkezdése előtt meg kell látogatnunka a következő linket <https://github.com/Microsoft/malmo>. Itt láthatjuk, hogy a Malmö telepítése elvégezhető Python Pip-el is illetve használhatjuk a pre-built verziókat is. Én a Linux pre-built verziót használtam a feladat megoldásához. Bármelyik verzió használásához szükségnünk lesz az open JDK 8-ra. Ezután megnyitotuk a letöltött Malmo mappát és az abban található Minecraft file-t megnyitjuk terminálban. Majd ./launchClient.sh parancssal elindítjuk a clientet.

Mostmár, hogy fut a cliensünk elkezdhetünk foglalkozni a feladatunkkal ami az volt, hogy Steve képes legyen mozogni a neki megadot világban anélkül, hogy megakadna. Ehhez először ismernünk kell az alap mozgási parancsokat.

```
agent_host.sendCommand("turn 1") // Jobbra fordul, teljes sebességgel. -1 ←  
    esetén ellenkező irányba fog fordulni. agent_host.sendCommand("move 1") ←  
    // 1-el előre -1-el hátrafelé indul teljes sebeséggel  
  
agent_host.sendCommand("pitch 1") // 1-el lefelé tekintünk -1-el pedig az ←  
    ellenkező irányba
```

```
agent_host.sendCommand("strafe -1") //bal oldalra mozdul, teljes ←  
sebességgel  
  
agent_host.sendCommand("jump 1") //ugrálás  
  
agent_host.sendCommand("crouch 1") //guggolás  
  
agent_host.sendCommand("attack 1") //támadás
```

MalmoPython.MissionSpec() függvény a missionXML-ből meghatároza, hogy milyen biomot akarunk létrehozni, milyen játékmodba legyünk állítva illetve meddig fusson egy adott mission. Ezt majd átadja a my_mission változonak.

Példa a missionXML-re

```
'''<?xml version="1.0" encoding="UTF-8" standalone="no" ?>  
    <Mission xmlns="http://ProjectMalmo.microsoft.com" xmlns:xsi="←  
        http://www.w3.org/2001/XMLSchema-instance">  
  
        <About>  
            <Summary>gitlab.com/whoisZORZ</Summary>  
        </About>  
  
        <ServerSection>  
            <ServerHandlers>  
                <FlatWorldGenerator generatorString="3;7,220*1,5*3,2;3;, ←  
                    biome_1"/>  
                <DrawingDecorator>  
                    <DrawSphere x="-27" y="70" z="0" radius="30" type="←  
                        air"/>  
                </DrawingDecorator>  
                <ServerQuitFromTimeUp timeLimitMs="360000"/>  
                <ServerQuitWhenAnyAgentFinishes/>  
            </ServerHandlers>  
        </ServerSection>  
  
        <AgentSection mode="Survival">  
            <Name>ZORZBot</Name>  
            <AgentStart/>  
            <AgentHandlers>  
                <ObservationFromFullStats/>  
                <ContinuousMovementCommands turnSpeedDegs="180"/>  
            </AgentHandlers>  
        </AgentSection>  
    </Mission>'''
```

Mielőt bármit is csinálnánk az agentnek meg kel adjunk, hogy folyamatosan előre haladjon és declarálunk kel néhány változót: a pozíció koordinátákat, irány változót, a nézetet pedig a stevepitch változó fogja

tárolni.A yaw értéke a Minecraft koordinátarendszeréhez igazodik a 180 északot,a 0 délt,a 90 nyugatot és a -90 pedig keletet.

```
# Loop until mission starts:  
print("Waiting for the mission to start ", end=' ')  
world_state = agent_host.getWorldState()  
while not world_state.has_mission_begun:  
    print(".", end="")  
    time.sleep(0.1)  
    world_state = agent_host.getWorldState()  
    for error in world_state.errors:  
        print("Error:",error.text)  
  
print()  
print("Mission running ", end=' ')  
  
agent_host.sendCommand( "move 1" )  
  
Sx = 0  
Sz = 0  
sy = 0  
Syaw = 0  
Spitch = 0  
eidx = 0  
eidxj = 0  
eidxb = 0  
Barrier = 0
```

Steveet mozgás közben blokkok fogják körül venni ezért előfordulhat,hogy megakad valamelyen akadályba például nekimegy egy fának.Ennek elkerülése érdekében a json függvényel információt fogunk gyűjteni a Stevet körülvevő blokkokról.Ezeket az információkat terminálra is kiírathatjuk.

```
if world.state.number_of_observations_since_last_state > 0:  
    msg = world_state.observations[-1].text  
    observations = json.loads(msg)  
    nbr = observations.get("nbr3x3", 0)  
    print("Mit latok: ", nbr)  
  
    if "Yaw" in observations:  
        Syaw = observations["Yaw"]  
    if "Pitch" in observations:  
        Spitch = observations["Pitch"]  
    if "XPos" in observations:  
        Sx = observations["XPos"]  
    if "ZPos" in observations:  
        Sz = observations["ZPos"]  
    if "YPos" in observations:  
        Sy = observations["YPos"]  
  
    print ("Pozicio koordinatak: ", Sx, Sz, Sy)  
    print ("Irany: ", Syaw)
```

```
print ("Nezet: ", Spitch)

if Syaw >= 180-22.5 and Syaw <= 180+22.5 :
    eidx = 1
    eidxj = 2
    eidxb = 0

if Syaw >= 180+22.5 and Syaw <= 270-22.5 :
    eidx = 2
    eidxj = 5
    eidxb = 1

if Syaw >= 270-22.5 and Syaw <= 270+22.5 :
    eidx = 5
    eidxj = 8
    eidxb = 2

if Syaw >= 270+22.5 and Syaw <= 360-22.5 :
    eidx = 8
    eidxj = 7
    eidxb = 5

if Syaw >= 360-22.5 or Syaw <= 0+22.5 :
    eidx = 7
    eidxj = 6
    eidxb = 8

if Syaw >= 0+22.5 and Syaw <= 90-22.5 :
    eidx = 6
    eidxj = 3
    eidxb = 7

if Syaw >= 90-22.5 and Syaw <= 90+22.5 :
    eidx = 3
    eidxj = 0
    eidxb = 6

if Syaw >= 90+22.5 and Syaw <= 180-22.5 :
    eidx = 0
    eidxj = 1
    eidxb = 3
```

Amikor Stevenek nem lesz szabad utja,akkor arra fogjuk utasítani, hogy kezdjen el fordulni és növeljük a Barrier változót eggyel.Amenyiben szabad az út Stevnek nem kell csinálnia semmit csak előre haladnia.Egyes akadályokat Steve ugrással is kivédhet.

```
if nbr[eidx+9]!="air" or nbr[eidxj+9]!="air" or nbr[eidxb+9]!="air":
    print ("Nincs szabad utam, elottem: ", nbr[eidx+9])
    agent_host.sendCommand ("turn" + str(turn))
```

```
        Barrier = Barrier + 1
    else:
        print ("Szabad az ut!")
        agent_host.sendCommand ("turn 0")
        agent_host.sendCommand ("jump 0")
        agent_host.sendCommand ("attack 0")
        Barrier = 0

    if Barrier > 8:
        agent_host.sendCommand ("jump 1")

    lepes = lepes + 1
    if lepes > 100:
        lepes = 0
    if tav < 20:
        prevSx = Sx
        prevSz = Sz
        prevSy = Sy
        turn = turn * -1
        agent_host.sendCommand ("attack 1")

    time.sleep(1)

print()
print("Mission ended")
# Mission has ended.
```

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Ebben a feladatban egy olyan programot kellett készítenünk, amely meghatározza egy szám faktoriálisát. Faktoriálisnak nevezzük a matematikában egy n nemnegatív egész szám, n -nél kisebb vagy egyenlő pozitív egész számok szorzatát (Jelölése: $n!$). Ezt a programot iteratív és rekurzív módon is meg kellett valósítani Lisp-ben.

Factoriális iteratívan

Elsönek itt láthatjuk az iteratív megoldásunkat amit egy for loop-al oldottunk meg. Láthatjuk, hogy z -nek bekérünk egy számot aminek a faktoriálisát meg akarjuk tudni. Itt még definiáltunk egy y -t is amiben a megoldásunkat fogjuk tárolni. Ezután indítunk egy for ciklust ami 1-től z -ig össze fogja szorozni a számokat amit eltárolunk y -ban. Végül princ parancsal kiíratjuk y -t ami a bekért számuk faktoriálisa lesz.

```
(princ "Szam: ")
(setq z (read))
(setq y 1)
(loop for x from 1 to z
      do (setq y (* y x)))
(princ y)
```

Factoriális rekurzívan

A rekurzív megvalósításban elsőnek írtunk egy factorial nevű függvényt. Ez maga nem olyan bonyolult, minden össze egy if-es eldöntésből áll. ha a szám kisebb mint 2 akkor a faktoriálisunk 1 lesz. Ellenkező esetben visszont a szám változót megszorozza azzal a számal amit a factorial függvény fog vissza adni a (szám - 1)-re. Ezen a ponton láthatjuk, hogy ez egy rekurzív függvény mivel a végrehajtás során önmagát hívja meg. A függvény megírása után nincs más dolgunk csak bekérünk egy számot z néven, majd egy kiíratáson belül meghívjuk rá a factorial függvényt. Ez vissza is fogja adni a bekért szám faktoriálisát.

```
(defun factorial(szam)
  (if (< szam 2)
    1
    (* szam (factorial (- szam 1)))))
```

```
)  
(princ "Szam: ")  
(setq z (read))  
  
(princ (factorial z))
```

9.2. Gimp Scheme Script-fu: króm effekt

A feladatunk az volt, hogy írunk olyan script-fukiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

Először létrehozunk egy 8 elemből álló tömböt egy függvény segítségével amelynek beállítjuk az értékeit.

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

Ennek a definénak a segítségével tudjuk majd elérni a lista x-edik elemét.

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

A következő kodrész felel a kép megformázásért.

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

```
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))

(list text-width text-height)
)
)

(define (script-fu-bhax-chrome-border text font fontsize width height new- ←
    width color gradient border-size)
(let*
(
    (text-width (car (text-wh text font fontsize)))
    (text-height (elem 2 (text-wh text font fontsize)))
    (image (car (gimp-image-new width (+ height (/ text-height 2)) 0)))
    (layer (car (gimp-layer-new image width (+ height (/ text-height 2) ←
        ) RGB-IMAGE "bg" 100 LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (layer2)
)
)

(gimp-image-insert-layer image layer 0 0)

(gimp-image-select-rectangle image CHANNEL-OP-ADD 0 (/ text-height 2) ←
    width height)
(gimp-context-set-foreground '(255 255 255))
(gimp-drawable-edit-fill layer FILL-FOREGROUND )

(gimp-image-select-rectangle image CHANNEL-OP-REPLACE border-size (+ (/ ←
    text-height 2) border-size) (- width (* border-size 2)) (- height ←
    (* border-size 2)))
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-edit-fill layer FILL-FOREGROUND )

(gimp-image-select-rectangle image CHANNEL-OP-REPLACE (* border-size 3) ←
    0 text-width text-height)
(gimp-drawable-edit-fill layer FILL-FOREGROUND )

(gimp-selection-none image)

;step 1
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (* border-size 3) 0)

(set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←
```

```
LAYER) )))

;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 25 TRUE TRUE)

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .18 .38 TRUE 1 0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)

;step 6
(set! layer2 (car (gimp-layer-new image width (+ height (/ text-height ↵
    2)) RGB-IMAGE "2" 100 LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)

;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT- ↵
    LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width 0 width (+ height (/ text-height 2)))

;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ↵
    0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-image-scale image new-width (/ (* new-width (+ height (/ text- ↵
    height 2))) width))

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

A következő kodrészletben a felhasználó használhatja az általunk alapértelmezetté tett beállításokat illetve lehetőséget biztosítunk neki, hogy más értékeket használjon.

```
(script-fu-register "script-fu-bhax-chrome-border"
    "Chrome3-Border2"
    "Creates a chrome effect on a given text."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 19, 2019"
    "")
```

```
SF-STRING      "Text"        "Rácz András István"
SF-FONT        "Font"         "Sans"
SF-ADJUSTMENT "Font size"   '(160 1 1000 1 10 0 1)
SF-VALUE       "Width"       "1920"
SF-VALUE       "Height"      "1080"
SF-VALUE       "New width"   "400"
SF-COLOR       "Color"        '(255 0 0)
SF-GRADIENT    "Gradient"    "Crown molding"
SF-VALUE       "Border size"  "7"
)
(script-fu-menu-register "script-fu-bhax-chrome-border"
  "<Image>/File/Create/BHAX"
)
```

9.3. Gimp Scheme Script-fu: név mandala

Ebben a feladatban egy olyan scriptet kellet írunk a GIMP-hez, amely egy névből mandalát készít. Ezt a kapott szöveg többszöros elforgatásával tehetjük meg. Ennek megvalósítását szemlélteti a következő kód.

Tanulságok, tapasztalatok, magyarázat...

```
(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-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"          "Rácz Andras István"
    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"
)
```

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

1. Fejezet: Bevezetés

Ebben a fejezetben megismerkedhetünk az alapvető fogalmakkal. Továbbá olvashatunk arról, hogy mi alapján osztályozzuk a programnyelveket. Megismerjük a programnyelvek két csoportjait: imperatív, deklaratív illetve, tudatja velünk hogy mik sorolhatók az egyéb nyelvek közé.

2. Fejezet: Alapelemek

A második fejezetben egy programozási nyelv alapeszközeit, alapfogalmait ismerjük meg. Részletesen tár-gyalunk a karakterkészletről illetve a többkarakteres szimbólumokról. A második fejezet negyedik részétől kezdve a különböző adattípusokról, konstansokról, mutatókról és változókról olvashatunk.

3. Fejezet: Kifejezések

Ebben a fejezetben a kifejezésekről beszélünk. Megnézük, hogy egy kifejezés milyen konponensekből te-vődnek össze (operandusok, operátorok, zárojelek). Megtudjuk, hogy egy kifejezésnek három alakja lehet infix, prefix és postfix. Ezután rátérünk a C nyelvben használt operátorok megismerésére.

10.2. Programozás bevezetés

1. Fejezet: Alapismeretek

Az első fejezetben arról olvashatunk, hogy, hogyan kell elkezdeni egy számunkra új programozási nyelv tanulását. Továbbá ebben a fejezetben megismerkedhetünk a C alapjaival mint például a kiíratás, változók és konstansok megadása, ciklusok. Ezeken kívül bevezet minket a tönbök és függvények világába is, illetve röviden összefoglalta, hogy mivel is fog foglalkozni a könyv a továbbiakban.

2. Fejezet: Típusok, operátorok és kifejezések

Ebben a részben a különboző declarációkról részletesen olvashatunk. Arról, hogy milyen módon érdemes a változó neveket létrehozni illetve, hogy vannak a C nyelv által lefoglalt kifejezések, illetve, hogy meg kell adnunk a változók típusát is. Továbbá ebben a részben olvashatunk a különböző (aritmetikai, relációs, logikai, inkrementáló, dekrementáló és értékadó) operátorokról is.

3. Fejezet: Vezérlési szerkezetek

Itt beszélünk az if-else utsításal való döntésekről, hogy hol adhatjuk meg a feltételt illetve, hogy hova kell írjuk a végrehajtani kívánt utasítást. Beszélünk továbbá a switch utasításról ahol megemlíti a break parancs amit majd részletezni fog. Ez után rátérunk a while, a for és a do while ciklusok részletezésére. Ezt a fejezetet a ritkán használt continue-val és goto-val zárjuk.

10.3. Programozás

1. Fejezet: Bevezetés

2. Fejezet: A C++ nem objektumorientált újdonságai

A második fejezetben a C és a C++ nyelv összehasonlítását láthatjuk. Először a függvényparaméterek megadását hasonlitjuk össze. Ezzutá olvashatunk, hogy milyen módon declarálhatunk változókat, illetve itt még megemlítiük a változótipusokat is. Ezek után visszatérünk a függvényekre, hogy, hogyan terhelhetünk túl őket. A fejezetet a praméterátadással és referenciatípusokkal zárjuk.

3. Fejezet: Objektumok és osztályok

Ebben a fejezetben az osztályokról olvashatunk, arról, hogy mit nevezzünk osztálynak, mire használjuk azokat, illetve megemlíti a még az objektumok fogalma, illetve az ezeken belüli adatvédelem és adatrejtés is. Itt továbbá részletes betekintést kapunk a konstruktorkról és destruktorkról világába és beszélünk a dinamikus adatkezelésről. Ez után visszatérünk a konstruktorkra nevezetesen a másoló konstruktort elemzük ki benne és olvashatunk a tagfüggvényekről is. Továbbá a könyv ezen részében olvasunk a frien függvényekről, a statikus tagokról és a beágyazott definíciókról.

4. Fejezet: Konstansok és inline függvények

A könyv negyedik fejezete először bemutatja nekünk, hogy a konstansokat hol és hogyan érdemes használni. Ezután részletesen olvashatunk a konstansok fontosabb fajtáiról például a konstans változókról, pointerekről, függvényparaméterekről, tagváltozókról stb. Információt kapunk arról, hogy abban az esetben ha egy konstans tagfüggvény egy adot változóját meg akarjuk változtatni akkor ezt úgy tehetjük meg, hogy a változót mutable-ként kel definiálnunk. Ezen fejezet második és egyben utolsó részében az inline függvényekről beszünk.

III. rész

Második felvonás

**Bátf41 Haxor Stream**

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

11. fejezet

Helló, Arroway!

11.1. 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)

A módosított polártranszformációs normális generátorral képesek vagyunk pseudorandom számok generálására. Pseudorandom számok olyan számok amelyek valamilyen algoritmus alapján lettek véletlenszerűen generálva, más szóval egy adott szám újra létrehozható úgyan azon algoritmus használatával. Az általunk használt algoritmus egyszerre két véletlen számot fog majd generálni a jobb optimalizálás érdekébe.

Java implementáció

A java kódot két főbb részre oszthatjuk. Az első maga a plártranszformációs algoritmusunk és egy main függvényre ami az algoritmus tesztelésére hozzunk léter.

```
public class polargen {  
  
    boolean nincsTarolt = true;  
    double tarolt;  
  
    public polargen() {  
        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;  
            } while(v1 * v1 + v2 * v2 >= 1);  
            nincsTarolt = false;  
            tarolt = Math.sqrt(-2 * Math.log(v1 * v1 + v2 * v2)) *  
                    Math.tan(2 * Math.PI * u1);  
        }  
        return tarolt;  
    }  
}
```

```
        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;
}
}
```

A polargen osztályban az első amit láthatunk az a nincsTarolt boolean és a tarolt double típusú változók ami a hatékonyabb tárolást fogja segíteni. A nincsTarolt fogja megadni, hogy van-e már számunk és a tarolt fogja tartalmazni az értéket. A következő metódus hozza létre a random számokat a Math osztály segítségével.

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

A második rész mint már feljebb említetem a main metódus ami csak a tesztelés szempontjából szükséges, ez 10 számot fog generálni de mivel egy futás alatt 2 számot ad a polargen algoritmus ezért a program minden ötször fog lefutni.

C++ implementáció

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
class polargen
{
public:
    polargen();
    double kovetkezo();
    ~polargen() {}
private:
    bool nincsTarolt;
    double tarolt;
};
polargen::polargen()
{
```

```
nincsTarolt = true;
std::srand (std::time(NULL));
};

double polargen::kovetkezo()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1 = std::rand () / (RAND_MAX + 1.0);
            u2 = std::rand () / (RAND_MAX + 1.0);
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);
        double r = std::sqrt ((-2 * std::log (w)) / w);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
        return r * v1;
    }
    else
    {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
};

int main()
{
    polargen rnd;
    for (int i = 0; i < 10; ++i)
        std::cout << rnd.kovetkezo() << std::endl;
}
```

A Java Sun-os implementáció és az általunk megírt algoritmus nagyon hasonlóak. Az az álapot van megoldva a JDK-ban hogy ameddig egy szál éppen a metódust veszi igénybe addig a többinek várnia kell. Erre a synchronized-ot használja. Illetve itt a StrictMath osztály kerül meghívásra ami minden különbözik a Math osztálytól, hogy környezetfüggetlenül ugyna azt az eredményt fogja visszaadni.

```
private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

synchronized public double nextGaussian() {
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    }
}
```

```
        else {
            double v1, v2, s;
            do {
                v1 = 2 * nextDouble() - 1;    // between -1.0 and 1.0
                v2 = 2 * nextDouble() - 1;    // between -1.0 and 1.0
                s = v1 * v1 + v2 * v2;
            }
            while (s >= 1 || s == 0);
            double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s) / ←
                s);
            nextNextGaussian = v2 * multiplier;
            haveNextNextGaussian = true;
            return v1 * multiplier;
        }
    }
}
```

11.2. „Gagyi”

Az ismert formális while ($x \leq t \&& x \geq t \&& t \neq x$); 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 forrására 3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Ez a feladat azon alapul, hogy javában megkülönböztetünk érték szerintit és az identitáson alapuló egyenlőséget. A feladatnak megfelelően mi a megoldásunkat a Integer osztályra fogjuk építeni. Nézzük is meg a példa programjainkat:

```
D:\> Prog > Prog2Konyv > source > MasodikFejezet > Arroway > Gagyi > Gagyi.java
1
2 class Gagyi
3 {
4     public static void main(String[] args)
5     {
6         Integer t = -129;
7         Integer x = -129;
8         while(x <= t && x>=t && t != x)[]
9             System.out.println("---");
10    }
11 }
12
andrásracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Arroway/Gagyi
--andrásracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Arroway/Gagyi$ java Gagyi
---
```

Láthatjuk, hogy a programunk végtelen ciklusba fog lépni, mivel $x \leq t$, $x \geq t$ és $t \neq x$ is igaz lesz mivel egyszere, ugyanabba az időben és helyen a memoriában nem létezhet egy osztályból két példány. Visszont ha átirjuk a -129-et -128-ra akkor más lesz a helyzet:

D:\> Prog > Prog2Konyv > source > MasodikFejezet > Arroway > Gagyi > Gagyi.java

```
1
2 class Gagyi
3 {
4     public static void main(String[] args)
5     {
6         Integer t = -128;
7         Integer x = -128;
8         while(x <= t && x>= t && t != x){
9             System.out.println("---");
10        }
11    }
12 }
```

andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Arroway/Gagyi\$ javac Gagyi.java
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Arroway/Gagyi\$ java Gagyi
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Arroway/Gagyi\$

Mint láthatjuk a program le fog álni. Ez azért van mert az Integer osztály gyorsítótárazza a [-128,127] intervalumot így azok hashCodeja megegyezik. Így ha ebből a tartományból választunk egy számot, akkor $t \neq x$ hamis lesz mivel x egyenlő lesz t-vel. Amennyiben mégis érték alapján szeretnénk összehasonlítani x-et t-vel akkor erre az equals metódust kell használnunk.

11.3. Yoda

Írunk olyan Java programot, ami java.lang.NullPointerException-t! https://en.wikipedia.org/wiki/Yoda_conditions

A Yoda conditions egy programozási stílus ahol egy kifejezés két részét felcseréljük. Ennek a stílusnak a neve a Star Wars Yoda nevű karakteréből származik aki nem szabványosan beszél angolul. A Yoda feltétel használatával elkerülhetjük az összehasonlításoknál felmerülő lehetséges hibát, azt hogy akaratlanul rendelnünk hozzá, nem pedig feltételezett határozunk meg.

Illetve elkerülhetjük a nem biztonságos null viselkedés egyes típusit.

```
String myString = null;
if (myString.equals("foobar")) { /* ... */ }
```

Ez a kód NullPointerException hibát adja

```
String myString = null;
if ("foobar".equals(myString)) { /* ... */ }
```

Igy pedig hamis értéket fogunk viszakapni, ahogyn az várható. Ez azért történik mivel null objektumra nem hívhatunk meg metódust.

11.4. Kódolás from scratch

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

Ezuttal a Bailey-Borwein-Plouff algoritmussal fogjuk kiszámolni a π -nek az értékét egy tetszőleges hexadecimális számjegyétől kezdődően.

```
public class PiBBP {

    String d16PiHexaJegyek;

    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();
    }

    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);

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

```
}
```

Ezen részeben láthatjuk a BBP algoritmus implementációját. Először is láthatunk egy d16PiHexaJegyek változót amibne majd az eredményünk tárolodni fog. A PiBBP metódusban meghívjuk a d16Sj metódust ami paraméterül megkapja a d-t, ami azt adja meg hogy hanyadik tizedesjegytől kivánjuk meghatározni a π értékét illetve második paraméterbe megkapja a BBP algoritmusbna meghatározott számok egyikét. Illetve miután meghatározta a számot hexadecimálissá kell alakítania. Ezt úgy teszi meg, hogy a szám minden egyes számjegyet megvizsgálja és ha az nem kisebb mint tíz akkor az adott számjegyből kivonja a tizet és a hexaJegyek tömbből azt az elmet szurja be helyére amelynek indexe megegyezik a kapott számmal.

```
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;  
}
```

n16modk a d16Sj metódus használja, ebben a részben számolja ki a program a a $16^n \text{ mod } k$ -t.

```
public String toString() {  
  
    return d16PiHexaJegyek;  
}  
  
public static void main(String args[]) {  
    System.out.print(new PiBBP(1000000)+"\n");  
}
```

```
}
```

A program végső része már csak a kiszámolt hexa jegy visszaadását láthatjuk illetve egy `main` ami példá-nyosít egy BBP algoritmust implementáló objektumot.

12. fejezet

Helló, Liskov!

12.1. Liskov helyettesítés sértése

A feladat megoldásához először a Liskov helyettesítési elvről (LSP-ről) kell ejtenünk pár szót. Az LSP szerint az alaposztályokra történő hivatkozásokat használó funkcióknak képesnek kell lenniük arra, hogy a származtatott osztály objektumait felhasználják anélkül, hogy tudnák. Másszóval a származtatott osztályoknak helyettesítenie kell az alaposztályt.

A Liskov helyettesítési elvet a négyzet és a téglalap példájával lehet talán a legjobban szemléltetni, és a jobb átlathatóság itt láthatjuk a példát javában megvalósítva:

```
class Rectangle {
    private int length;
    private int breadth;
    public int getLength() {
        return length;
    }
    public void setLength(int length) {
        this.length = length;
    }
    public int getBreadth() {
        return breadth;
    }
    public void setBreadth(int breadth) {
        this.breadth = breadth;
    }
    public int getArea() {
        return this.length * this.breadth;
    }
};

class Square extends Rectangle {

    public void setBreadth(int breadth) {
```

```
        super.setBreadth(breadth);
        super.setLength(breadth);
    }

    public void setLength(int length) {
        super.setLength(length);
        super.setBreadth(length);
    }
};

class LSP {

    public void calculateArea(Rectangle r) {
        r.setBreadth(2);
        r.setLength(3);

        assert r.getArea() == 6 : printError("area", r);
        assert r.getLength() == 3 : printError("length", r);
        assert r.getBreadth() == 2 : printError("breadth", r);
    }

    private String printError(String errorIdentifier, Rectangle r) {
        return "Unexpected value of " + errorIdentifier + " for instance of " +
               r.getClass().getName();
    }

    public static void main(String[] args) {
        LSP lsp = new LSP();
        lsp.calculateArea(new Rectangle());
        lsp.calculateArea(new Square());
    }
};
```

Az elv szerint az alaposztályokra történő hivatkozásokat használó funkcióknak képesnek kell lenniük arra, hogy a származtatott osztály objektumait felhasználják anélkül, hogy tudnák. Tehát a példában látható CalcArea függvénynek képesnek kell lennie arra, hogy a származtatott osztály objektumait, például a Négyzetet, felhasználja, és teljesítse a Téglalap által meghatározott követelményt. A probléma akkor merül fel amikor ugyanezt a Négyzetre akarjuk megesinálni mivel a square osztálynak nincs szüksége olyan módszerre, mint a setBreadth vagy a setLength, mivel a négyzet oldalai egyenlőek.

LSP sértése C++ nyelvben ezuttal a madaras példán keresztül.

```
class Madar {
public:
    virtual void repul() {};
};
```

```
class Program {
public:
    void fgv ( Madar &madar ) {
        madar.repul();
    }
};

// itt jönnek az LSP-s osztályok
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 );
}
```

12.2. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek!

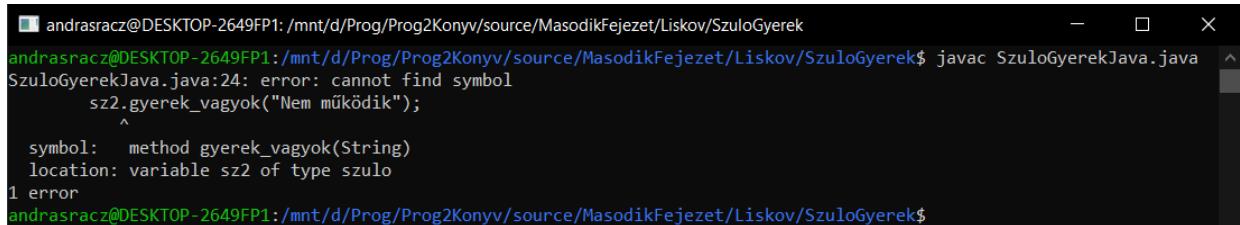
Ebben a feladatban minden össze azt kellet szemléltetni hogy a szülő referencián keresztül nem lehetséges meghívni a gyermek egy olyan metódusát amit ő maga nem határozott meg.

Javás példa

```
class szulo
{
    public void szulo_vagyok()
    {
        System.out.println("Én vagyok én.");
    }
};
```

```
class gyerek extends szulo
{
    public void gyerek_vagyok(String x)
    {
        System.out.println(x);
    }

    public static void main(String[] args)
    {
        szulo sz = new szulo();
        szulo sz2 = new gyerek();
        System.out.println("A szülő meghívó módszere");
        sz.szulo_vagyok();
        System.out.println("A gyermek meghívása a szülőn referencián ←
            keresztül ");
        sz2.gyerek_vagyok("Nem működik");
    }
};
```



The screenshot shows a terminal window with the following text:

```
andrasracz@DESKTOP-2649FP1:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Liskov/SzuloGyerek
andrasracz@DESKTOP-2649FP1:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Liskov/SzuloGyerek$ javac SzuloGyerekJava.java
SzuloGyerekJava.java:24: error: cannot find symbol
    sz2.gyerek_vagyok("Nem működik");
           ^
      symbol:   method gyerek_vagyok(String)
      location: variable sz2 of type szulo
1 error
andrasracz@DESKTOP-2649FP1:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Liskov/SzuloGyerek$
```

C++ példa

```
class szulo
{
    public void szulo_vagyok()
    {
        System.out.println("Én vagyok én.");
    }
};

class gyerek extends szulo
{
    public void gyerek_vagyok(String x)
    {
        System.out.println(x);
    }

    public static void main(String[] args)
    {
        szulo sz = new szulo();
        szulo sz2 = new gyerek();
```

```

        System.out.println("A szülő meghívó módszere");
        sz.szulo_vagyok();
        System.out.println("A gyermek meghívása a szülőn referencián ←
            keresztül ");
        sz2.gyerek_vagyok("Nem működik");

    }
}

```

```

andrasracz@DESKTOP-2649FP1:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Liskov/SzuloGyerek
andrasracz@DESKTOP-2649FP1:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Liskov/SzuloGyerek$ g++ SzuloGyerekC++.cpp -o t ^
eszt
SzuloGyerekC++.cpp:27:18: warning: character constant too long for its type
    std::cout << 'A szülő meghívó módszere';
                  ^~~~~~
SzuloGyerekC++.cpp: In function 'int main(int, char**)':
SzuloGyerekC++.cpp:30:9: error: 'class szulo' has no member named 'gyerek_vagyok'; did you mean 'szulo_vagyok'?
    p2->gyerek_vagyok("Nem működik");
                  ^~~~~~
          szulo_vagyok
andrasracz@DESKTOP-2649FP1:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Liskov/SzuloGyerek$
```

12.3. Anti OO

Ebben a feladatban az Arroway fejezetben már megismert BBP algoritmust fogjuk használni. Annyi különböző lesz hogy még az Arroway fejezetben csak a javas verziót használtuk itt most megalkotjuk a C, C++ és a C# verzióját és összehasonlítottuk a futási idejüket.

	Java	C	C++	C#
10⁶	1.3	1.43	1.42	1.33
10⁷	15.35	16.71	16.65	15.49
10⁸	175.32	192.72	192.67	177.19

Mint ahogyan a táblázatban látható a java lett a leggyorsabb. Ez azért törtnént mert a javában már alapból egy csomó minden meg van írva amik jobbak amit mi C/C++-ba megírnánk, illetve futás közben is egy csomó optimalizálás történik ez abbol is látható hogy 10⁸-nál a java már jeletösen gyorsabb volt.

Forráskódok

```

//Java kód

public class PiBBPBench {

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

        double d16Sj = 0.0d;

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

```
        return d16Sj - Math.floor(d16Sj);
    }

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

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

        long r = 1;

        while(true) {

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

            t = t/2;

            if(t < 1)
                break;

            r = (r*r) % k;
        }

        return r;
    }

    public static void main(String args[]) {

        double d16Pi = 0.0d;

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

        int jegy = 0;

        long delta = System.currentTimeMillis();

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

            d16Pi = 0.0d;
```

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

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

d16Pi = d16Pi - Math.floor(d16Pi);

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

}

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

```
// C# kód

public class PiBBPBench {

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

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

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

    public static long n16modk(int n, int k) {
        int t = 1;
        while(t <= n)
            t *= 2;

        long r = 1;

        while(true) {

            if(n >= t) {
```

```
        r = (16*r) % k;
        n = n - t;
    }

    t = t/2;

    if(t < 1)
        break;

    r = (r*r) % k;

}

return r;
}

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

    double d16Pi = 0.0d;

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

    int jegy = 0;

    System.DateTime kezd = System.DateTime.Now;

    for(int d=100000000; d<100000001; ++d) {

        d16Pi = 0.0d;

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

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

        d16Pi = d16Pi - System.Math.Floor(d16Pi);

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

    }

    System.Console.WriteLine(jegy);
    System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
    System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
}
```

```
}
```

```
// C kód

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

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

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

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

        t = t / 2;

        if (t < 1)
            break;

        r = (r * r) % k;
    }

    return r;
}

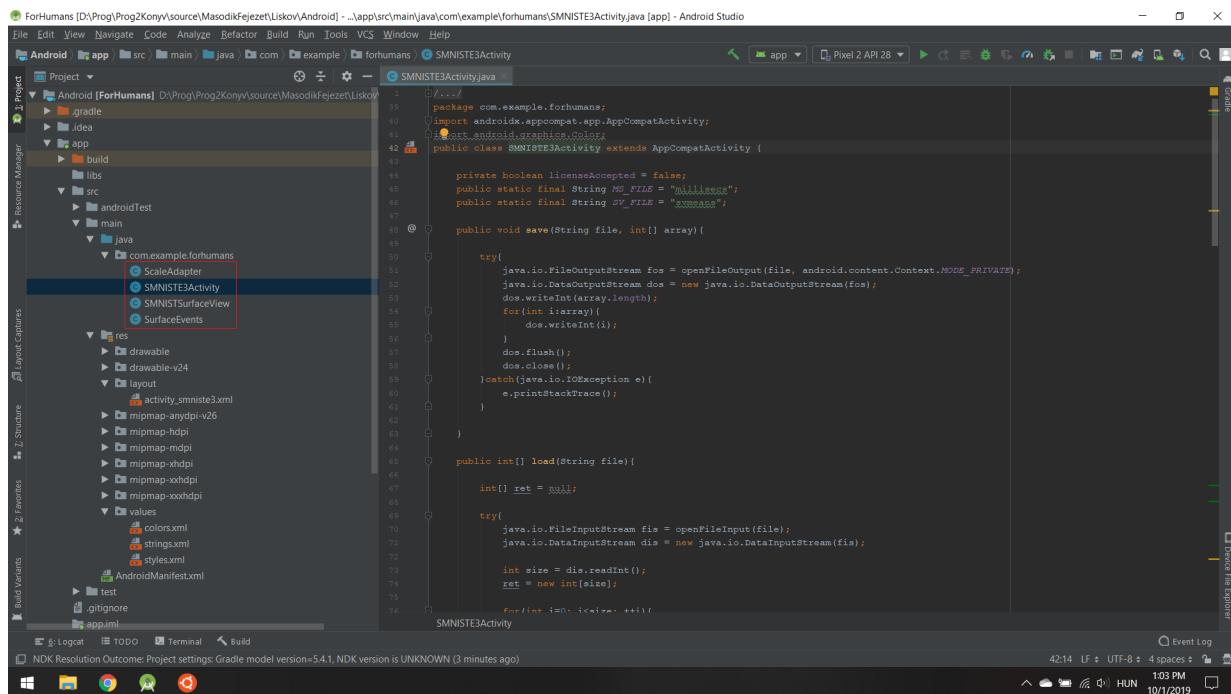
double
d16Sj (int d, int j)
{
    double d16Sj = 0.0;
    int k;

    for (k = 0; k <= d; ++k)
```

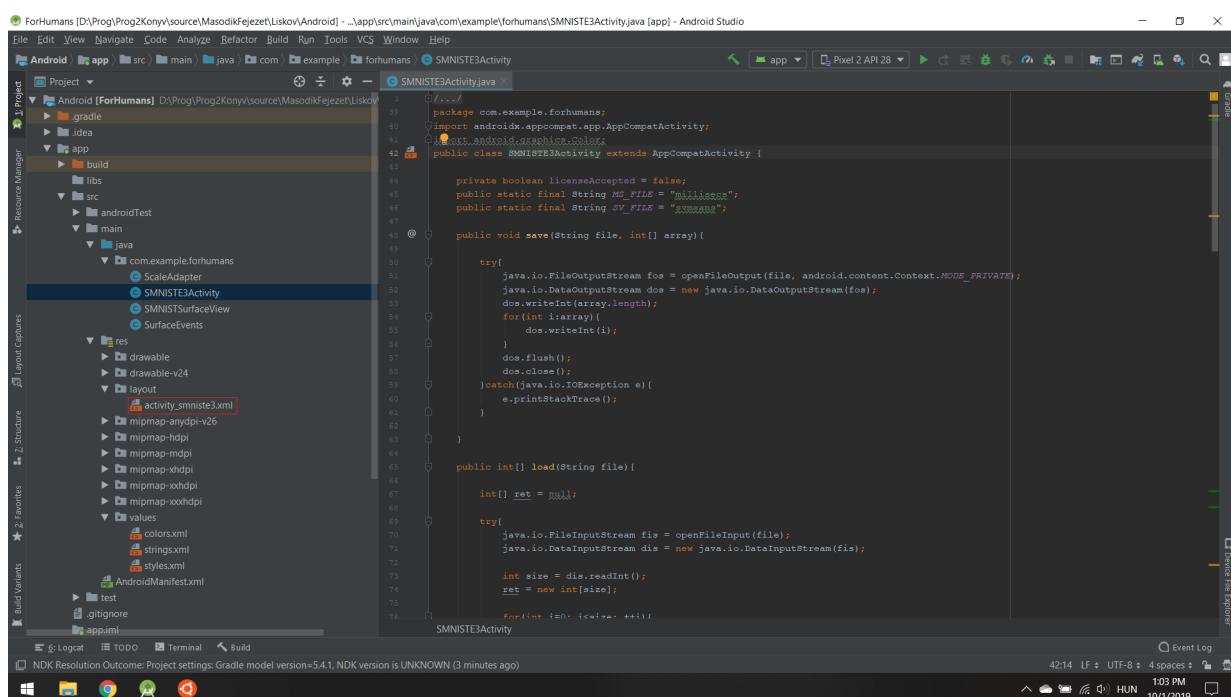
```
d16Sj += (double) n16modk (d - k, 8 * k + j) / (double) (8 * k + j);  
return d16Sj - floor (d16Sj);  
}  
  
main ()  
{  
  
    double d16Pi = 0.0;  
  
    double d16S1t = 0.0;  
    double d16S4t = 0.0;  
    double d16S5t = 0.0;  
    double d16S6t = 0.0;  
  
    int jegy;  
    int d;  
  
    clock_t delta = clock ();  
  
    for (d = 1000000; d < 1000001; ++d)  
    {  
  
        d16Pi = 0.0;  
  
        d16S1t = d16Sj (d, 1);  
        d16S4t = d16Sj (d, 4);  
        d16S5t = d16Sj (d, 5);  
        d16S6t = d16Sj (d, 6);  
  
        d16Pi = 4.0 * d16S1t - 2.0 * d16S4t - d16S5t - d16S6t;  
  
        d16Pi = d16Pi - floor (d16Pi);  
  
        jegy = (int) floor (16.0 * d16Pi);  
  
    }  
  
    printf ("%d\n", jegy);  
    delta = clock () - delta;  
    printf ("%f\n", (double) delta / CLOCKS_PER_SEC);  
}
```

12.4. Hello, Android!

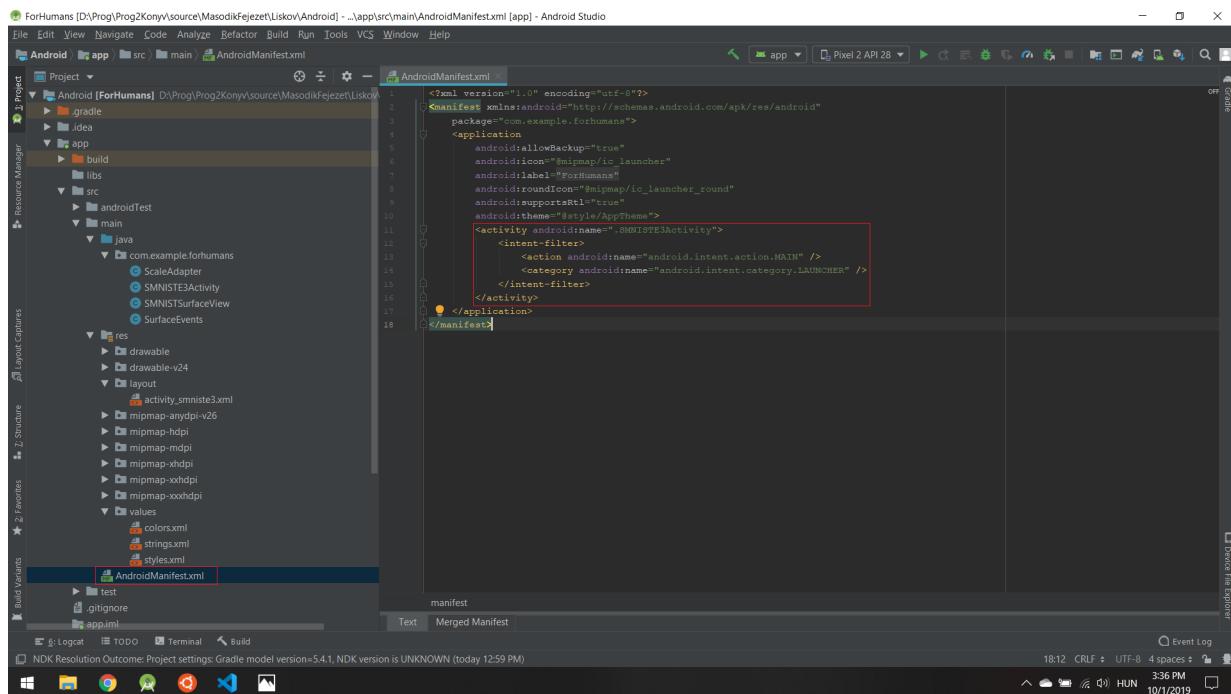
Feladatunk az volt, hogy éleszük fel az SMNIST alkalamazást és tegyünk benne aprób változtatásokat. Először is ehhez fel kellet telepitenünk az android studio alkalamazást amit a feladat megoldásához fogunk használni. Amiután ezzel végeztünk létrehozunk egy új projectet. A <https://gitlab.com/nbatfa/smnist/tree/master/> repórból letölthjük az SMNIST fájait majd ebből a ScaleAdapter, SMNISTE3Activity, SMNISTSurfaceView és a SurfaceEvents átmásoljuk a mi projectünk megfelelő mappába.



Ezután az activity_smniste3.xml-t bemásoljuk a layout mappába.

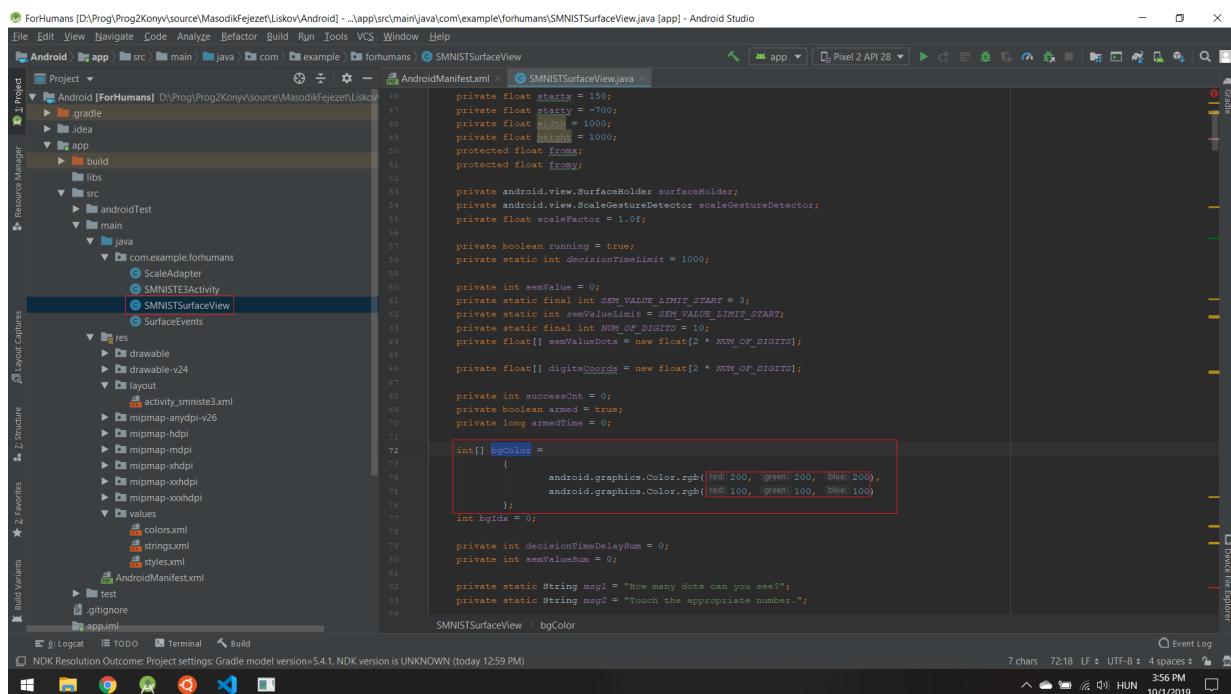


Ahhoz, hogy ez működjön az AndroidManifest.xml-be be kell írnunk néhány sort.

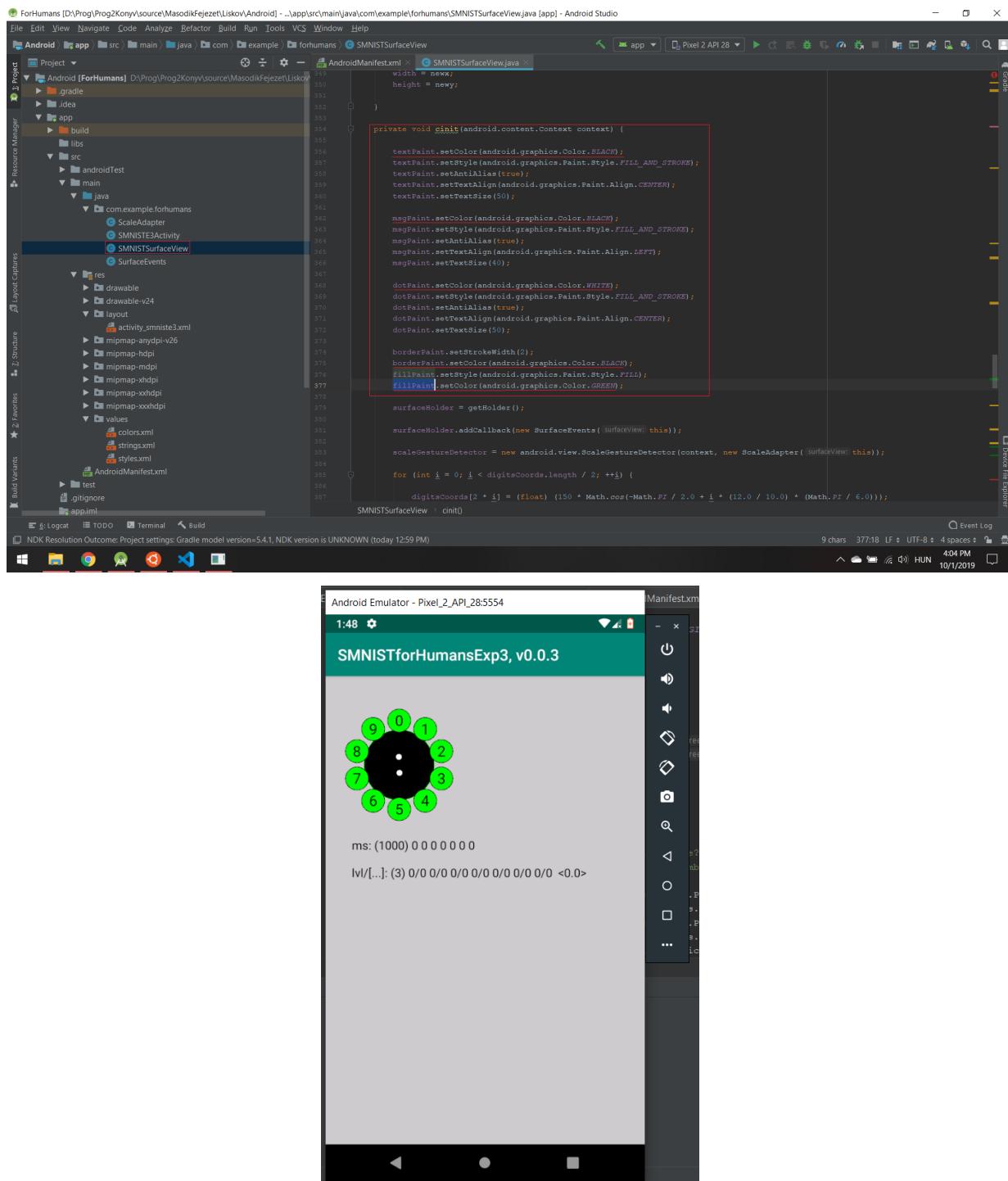


Ha ezeket végeztünk akkor már csak annyi van hátra, hogy átírunk pár dolgot a kódban. Erre az Android JDK gyors változása miatt van szükség. Nem csak újítások kerülnek be te változik a szintaxisa és a strukturája is.

Most, hogy a fájlaink a helyükön vannak és minden működik átérhetünk a változtatásokra. Az első amit meg fogunk tenni az, hogy megváltoztatjuk a háttér színét. Ezt úgy tehetjük meg, hogy az SMNISTSurfaceView fájlban átírjuk a `bgColor`-ban az `rgb` kódokat úgy, hogy a nekünk tetsző színeket kapjuk.



Következőnek a `private void` cíntben található `textPaint`-et, `msgPaint`, `dotPaint`, `borderPaint` és a `fillPaint` fogjuk módosítani szintén saját tetszésünknek megfelelően.



Az én változtatásaimmal például ez az eredmény született

12.5. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását!

A ciklomatikus komplexitás egy szoftvermetrikai ami egy szoftver komplexitását határozza meg. A komplexitás kisszámítása a gráfelméleten alapul, a kódban lévő elágazásokból felépülő gráf pontjai, és a köztük található élekből számítható ki a következőképpen:

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

Ahol E a gráf éleinek, N a gráfban lévő csúcsok és P pedig az összefüggő komponensek száma. A képlet alapján kapott M érték pedig a ciklomatikus komplexitás lesz aminek nem lenne szabad meghaladnia a 10-et, mivel e felet a kód nehezen értelmezhetővé, javithatóvá válik.

Mi a LZWBInfa.cpp kódnak néztük meg a ciklomatikus komplexitását és ez az eredmény született:

Function Name	NLOC	Complexity	Token #
LZWBInFa::LZWBInFa	4	1	17
LZWBInFa::operator <<	29	4	106
LZWBInFa::kiir	5	1	19
LZWBInFa::szabadít	5	1	24
LZWBInFa::operator <<	5	1	26
LZWBInFa::kiir	5	1	21
LZWBInFa::Csomopont::Csmopont	1	1	24
LZWBInFa::Csmopont::~Csmopont	1	1	5
LZWBInFa::Csmopont::nullasGyermek	3	1	9
LZWBInFa::Csmopont::egyesGyermek	3	1	9
LZWBInFa::Csmopont::ujNullasGyermek	3	1	12
LZWBInFa::Csmopont::ujEgyesGyermek	3	1	12
LZWBInFa::Csmopont::getBetu	3	1	9
LZWBInFa::kiir	13	3	94
LZWBInFa::szabadít	9	2	37
LZWBInFa::getMelyseg	6	1	24
LZWBInFa::getAtlag	7	1	35
LZWBInFa::getSzoras	12	2	67
LZWBInFa::rmelyseg	12	3	52
LZWBInFa::ratlag	15	4	69
LZWBInFa::rszoras	15	4	81
usage	4	1	18
main	35	6	248

Mint láthatjuk a legmagasabb értékünk is mindössze csak 6 volt, átlagában pedig a program ciklomatikus komplexitása valahol 1-2 között mozog, ami azt jelenti, hogy az LZWBInfa.cpp elég jól átlátható.

13. fejezet

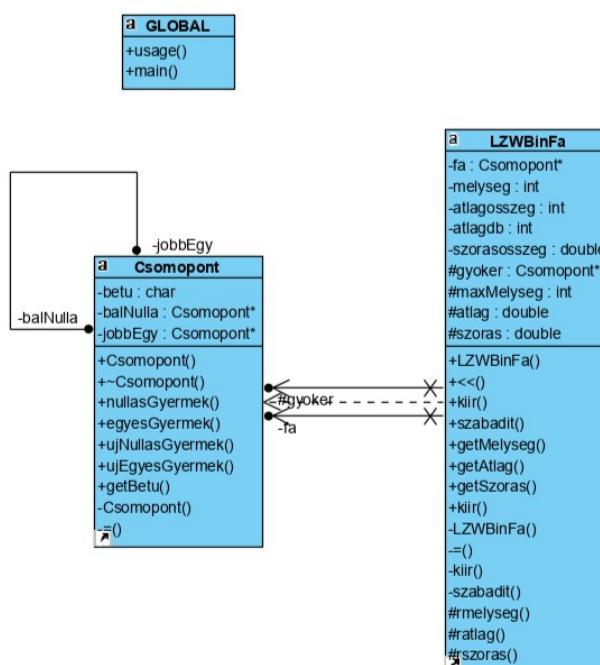
Helló, Mandelbrot!

13.1. 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.

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_6.pdf (28-32 fólia)

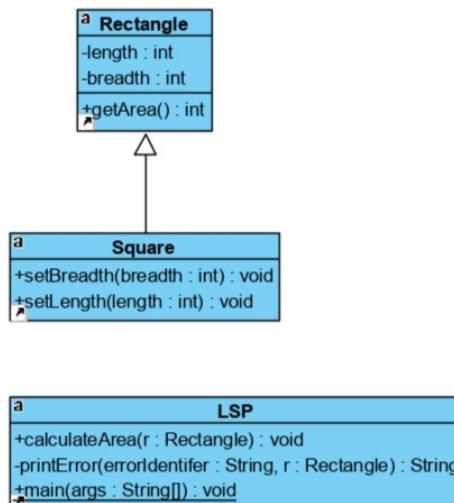
Ebben a feladatban UML osztálydiagramot készítettünk az első védési C++ programhoz. Az uml diagrammal az objektumorientált rendszerünkben lévő osztályokat és azok között fennálló kapcsolatokat modellezzi. Mi erre a feladatra a Visual Paradigma nevű szoftvert használtuk. Ez képes arra hogy a már meglévő, mi esetünkbe C++ fájlból automatikusan C++ fájt generáljon. Ezt a programon belül a Tools fül Code menüt lenyitva az Instan revers menüponttal tehetjük meg. Ha erre rágatunk akkor megnyilik egy új ablak ott megadjuk a C++ fájlt amit reversálni kívánunk és rákatintunk az ok gombra. Miután ezt az első védési C++ programmal megesináltuk ezt az eredményt kaptuk:



Most, hogy a UML diagramot elkészítetük nézzük is meg a kompozíció és aggregáció kapcsolatát. Az aggregáció egy tartalmazást jelölő aszociáció. A kompozíciójánál a tartalmazott és a tartalmazó együtt jön létre és együtt szünik meg. Nálunk például a Csomopont gyoker elem egy kompozíció.

13.2. Forward engineering UML osztálydiagram

Ez a feladat hasonló az előzőhez anyi külömlbség van, hogy ezuttal egy UML diagramból fogjuk legenerálni a mi esetünkben java fájlokat. Ehhez szintén a Visual Paradigmát fogjuk alkalmazásb venni először készítenünk kell egy osztály diagrammot. Én a Liskov felyezetben megírt LSP.java foráskódhoz fogok UML Class diagrammot létrehozni.



Most, hogy az UML diagrammunk megvan át is convertálhatjuk kóddá. Az átalakítás elvégeztével láthatjuk hogy négy darab új fájlt hozott létre az UML diagramunkból, egyet az LPS, egyet a Rectangle és egyet a Square osztálynak illetve egy build fájlt. Ezeke lent láthatjuk.

```
class Rectangle {
    private int _length;
    private int _breadth;

    public int getArea() {
        throw new UnsupportedOperationException();
    }

    public void setLength(int aLength) {
        this._length = aLength;
    }

    public int getLength() {
        return this._length;
    }

    public void setBreadth(int aBreadth) {

```

```
    this._breadth = aBreadth;
}

public int getBreadth() {
    return this._breadth;
}
}
```

```
class Square extends Rectangle {

    @Override
    public void setBreadth(int aBreadth) {
        throw new UnsupportedOperationException();
    }

    @Override
    public void setLength(int aLength) {
        throw new UnsupportedOperationException();
    }
}
```

```
class LSP {

    public void calculateArea(Rectangle aR) {
        throw new UnsupportedOperationException();
    }

    private String printError(String aErrorIdentifier, Rectangle aR) {
        throw new UnsupportedOperationException();
    }

    public static void main(String[] aArgs) {
        throw new UnsupportedOperationException();
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<project basedir=". " default="build">
    <path id="build.classpath">
        <pathelement location="classes"/>
    </path>
    <target name="init">
        <mkdir dir="classes"/>
    </target>
```

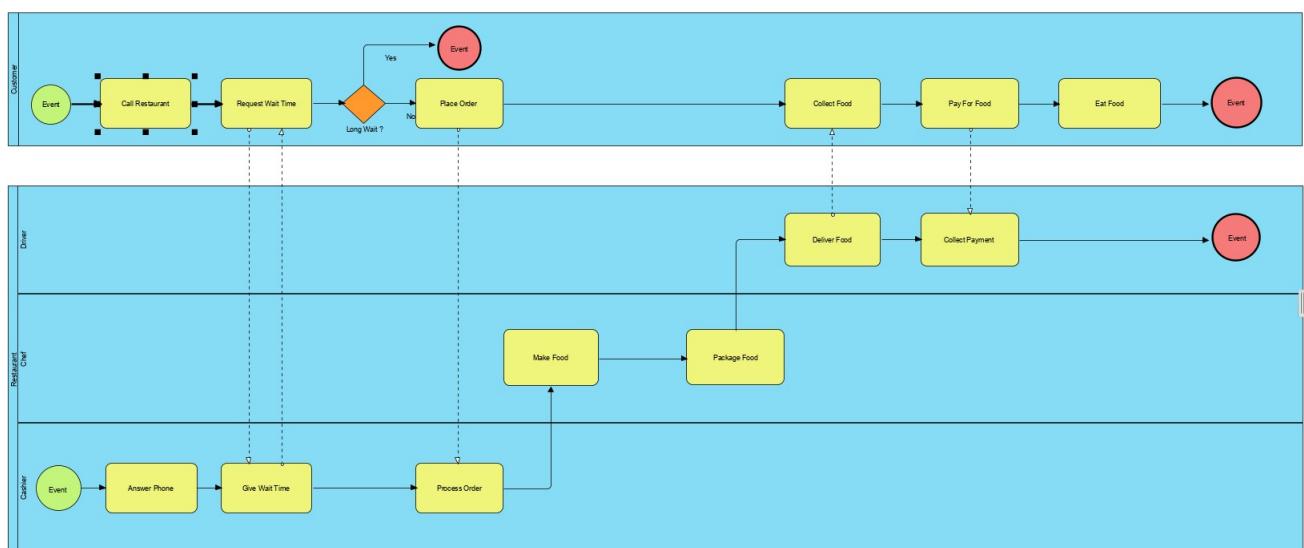
```

<target name="clean">
    <delete dir="classes"/>
</target>
<target depends="init" name="build">
    <javac srcdir=". " destdir="classes" source="1.5" target="1.5" fork="yes" >
        <classpath refid="build.classpath"/>
        <include name="Square.java"/>
        <include name="Rectangle.java"/>
        <include name="LSP.java"/>
    </javac>
</target>
</project>

```

13.3. BPMN

Ebben a feladatban egy tevékenységet kellet modellezünk a BPMN nyelv segítségével. Én egy éttermi rendelést írtam le.



Mint láthatjuk itt egy kétirányú kommunikáció történik az étterem és a vendég közzöt. Először a vevő felhívja az éttermet majd megkérdezi a várakozási időt. Erre a pénztáros megadja a várakozási időt. A kapott információ alapján a vevő eldönti, hogy a várakozási idő hosszú-e. Ha nem akkor leadja a rendelést a pénztárosnak, itt történik még egy feldolgozás majd az információ átadodik a Chef lineba. Itt az étel elkészül majd a Package Food álapotba kerül ami a Driver, Deliver Food továbbítódik. Az étel kiszállításról kerül a vevő átadja a fizetséget a Drivernek aki atveszi azt majd új esemény kezdődik a Restaurant oldaláról. A vevő az étel elfogyasztása után szintén valamilyen új esemény kezdődik. Az étterem és a vevő esetében is előfordulhat, hogy az új esemény helyet a tevékenység vége van.

13.4. TeX UML

Ebben a feladatban ismét egy UML osztálydiagramot fogunk készíteni ezután valamelyen TeX-es csomag felhasználásával. Mi a MetaUML-t fogjuk használni.

MetaUML telepítése Ubuntu rendszerre

Először nyitunk egy terminált, majd fel kel raktunk a texlive-metapostot

```
sudo apt install texlive-metapost
```

ezután lefutatjuk

```
https://github.com/ogheorghies/MetaUML.git
```

parancsot. Miután már megvannak a szükséges fájlok csak annyi maradt hátra hogy a következő három sort(ami a fájljainkat a megfelelő helyekre pakolja) lefuttassuk

```
export METAUML_DIR=/usr/share/texlive/texmf-dist/metapost/metauml  
sudo mv ${METAUML_DIR} ${METAUML_DIR}-backup-$ (date +%s)  
sudo ln -svf ${PWD}/MetaUML/src /usr/share/texlive/texmf-dist/metapost/ ←  
metauml
```

Most, hogy a telepítéssel végeztünk el is készíthetjük az UML diagramunkat az OOCWC projektről. A MetaUML mp fájlokóból generál pdf-et.

Alapvető parancsok

Először is a classok elhelyezésével kell foglalkoznunk ami a Class.Név("ClassNeve") parancsal hozznuk létre. Ez után az első zárojel között a class attributumait a másodikba pedig a metódusok kerülnek.

```
Class.A("MyShmClient")  
  ("# nr_graph: NodeRefGraph*", "# m_teamname: string", "- nr2v: map< ←  
    unsigned_object_id_type, NRGVertex>")  
  ("+MyShmClient()",  
  "+~MyShmClient()",  
  "+start()",  
  "+start10()",  
  "+num_vertices()",  
  "+print_edges()",  
  "+print_vertices()",  
  "+bgl_graph()",  
  "+hasDijkstraPath()",  
  "+hasBellmanFordPath()",  
  "-foo()",  
  "-init()",  
  "-gangsters()",  
  "-initcops()",  
  "-pos()",  
  "-car()",  
  "-route()");
```

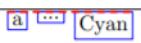
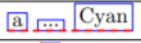
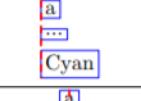
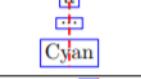
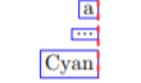
A Classokhoz még hozzátaroznak a stereotzepok létrehozása ezt a classStereotypes.Név("StereotypeNeve").

```
classStereotypes.B("<<Struct>>");  
classStereotypes.C("<<Typedef>>");  
classStereotypes.D("<<Typedef>>");  
classStereotypes.L("<<enumeration>>");  
classStereotypes.P("<<enumeration>>");  
classStereotypes.T("<<Struct>>");
```

Az így létrehozott Classokat (amit én az ABC betűivel neveztem el) packageekbe kellet ragnunk. Packageet a classhoz hasonlóan Package.Neve("PackageNeve") hozunk létre de ebben az esetben csak egy zárojel lesz amibe a Packagebe belerakni kívánt classok nevei lesznek.

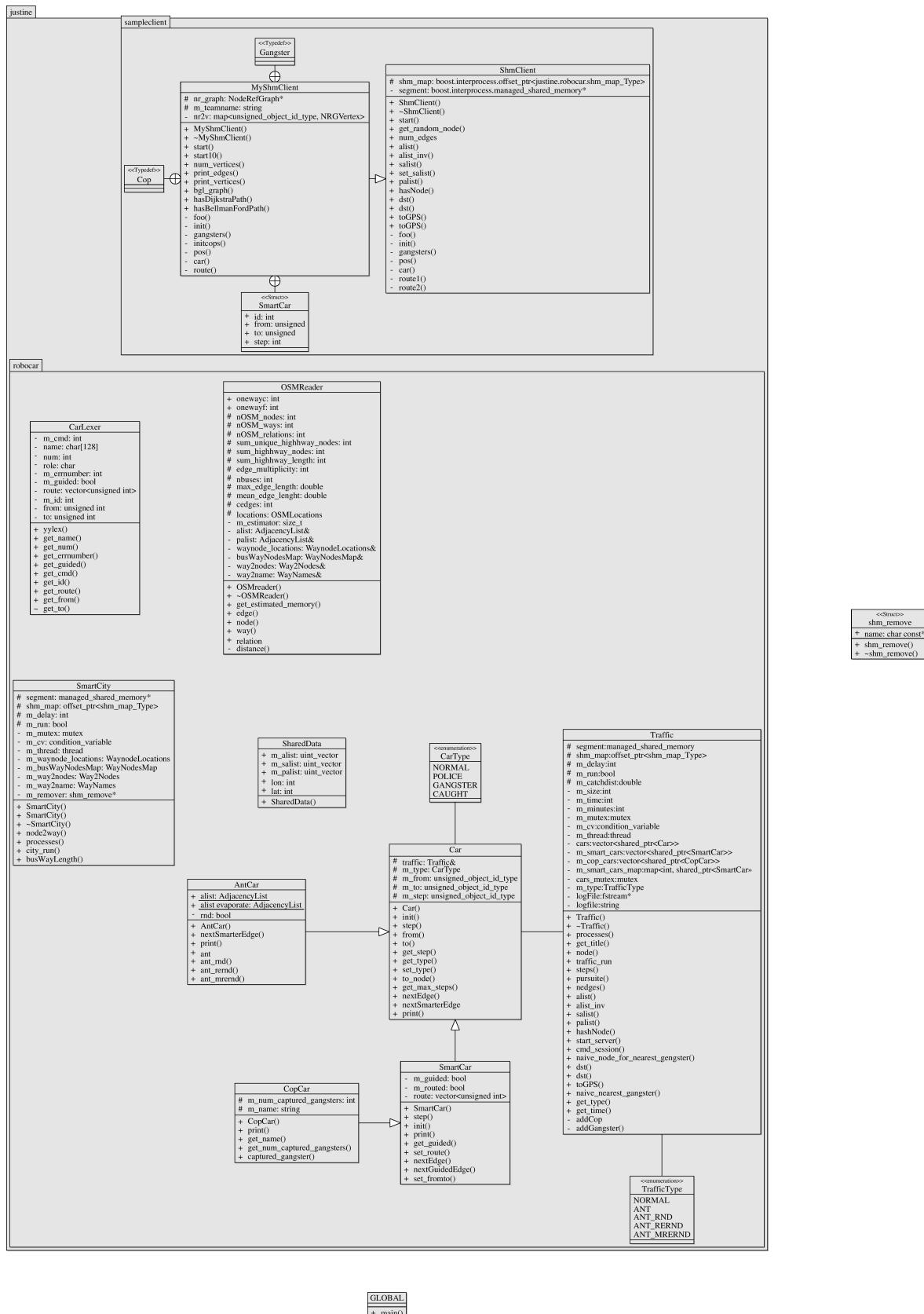
```
Package.Q("sampleclient") (A, B, C, D, E);
```

Ezután már csak a helyére kell ragnunk a létrehozott objektumokat, hogy ne egymás hegyén hátán legyenek. Ezt különböző pozicionáló parancsokkal tehetjük meg:

leftToRight.top(spacing)(X, Y, Z);	
leftToRight.midY(spacing)(X, Y, Z);	
leftToRight.bottom(spacing)(X, Y, Z);	
topToBottom.left(spacing)(X, Y, Z);	
topToBottom.midX(spacing)(X, Y, Z);	
topToBottom.right(spacing)(X, Y, Z);	

Az mp fájl létrehozása után már csak egy feladatunk van a célig mégpedig a pdfünk létrehozása amit az mptopdf fajlnev.mp-vel tehetünk meg. Ezzel létre is hoztuk az első olyan UML diagramunkat amit egy TeX csomaggal hoztunk létre.

Nézük is meg az eredményt:



14. fejezet

Helló, Chomsky!

14.1. Encoding

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

Ebben a feldatban azt kellet bemutatnunk, hogy a javac fordítónak megadhatjuk, hogy milyen karakterkódolást alkalmazon. Ezt a tankönyvtárban talalható mandelbrot nagyítón fogjuk szemléltetni ahol láthatjuk, hogy a Magyar nyelvben használt karakterek találhatóak például a változónevekben. Ha ezt csak simán probálnánk fordítani errorok sorozatát kapjuk. Ez azért van, mert a javac alapból a rendszer alapértelmezett karakterkódolást használja. Ha mi azt akarjuk, hogy mászt használjon azt a -encoding "használni kívánt kódolás" kapcsolót használjuk, szóval a fordítás parancs így fog kinézni javac -encoding "ISO-8859-2" MandelbrotIterációk.java MandelbrotHalmazNagyító.java. Mint láthatjuk nekünk az ISO-8859-2 kódolás a megfelelő mivel ez tartalmaza az Eastern European (Albanian, Croatian, Czech, English, German, Hungarian, Latin, Polish, Romanian, Slovak, Slovenian, Serbian) karaktereket. Ilyen módon használhatunk Magyar ékezetes karaktereket változo, osztály vagy függvény névként, de ennek ellenére nagyon ajánlot elkerülni.

```
andrásr@DESKTOP-3TH3DCG:~/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Chomsky/Encoding$ javac -encoding "ISO-8859-2" MandelbrotIteracio.java
andrásr@DESKTOP-3TH3DCG:~/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Chomsky/Encoding$ java MandelbrotHalmazNagyito
Exception in thread "AWT-EventQueue-0" java.lang.IllegalArgumentException: Width (600) and height (-649) cannot be <= 0
at java.awt.image.DirectColorModel.createCompatibleWritableRaster(DirectColorModel.java:1016)
at java.awt.image BufferedImage.<init>(BufferedImage.java:324)
at MandelbrotHalmaz.<init>(MandelbrotHalmaz.java:55)
at MandelbrotHalmazNagyito.<init>(MandelbrotHalmazNagyito.java:34)
at MandelbrotHalmazNagyito$1.mouseReleased(MandelbrotHalmazNagyito.java:78)
at java.awt.Component.processMouseEvent(Component.java:6539)
at java.awt.Component.dispatchEvent(Component.java:6304)
at java.awt.Container.dispatchEvent(Container.java:2239)
at java.awt.Window.dispatchEvent(Window.java:2025)
at java.awt.Component.dispatchEventImpl(Component.java:4889)
at java.awt.Container.dispatchEventImpl(Container.java:2297)
at java.awt.Window.dispatchEventImpl(Window.java:2746)
at java.awt.Component.dispatchEvent(Component.java:4711)
at java.awt.EventQueue.dispatchEvent(EventQueue.java:760)
at java.awt.EventQueue.access$500(EventQueue.java:97)
at java.awt.EventQueue$3.run(EventQueue.java:709)
at java.awt.EventQueue$3.run(EventQueue.java:703)
at java.security.AccessController.doPrivileged(Native Method)
at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:74)
at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:84)
at java.awt.EventQueue$4.run(EventQueue.java:733)
at java.awt.EventQueue$4.run(EventQueue.java:731)
at java.security.AccessController.doPrivileged(Native Method)
at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:74)
at java.awt.EventQueue.dispatchEvent(EventQueue.java:728)
at java.awt.EventQueue.dispatchOneEventForFilters(EventDispatchThread.java:205)
```

14.2. I334d1c4

Ebben a feladatban írnunk kellet egy olyan OO Java vagy C++ osztályt, amely leet cipherként működik. A leet cipher egyfajt betűhelyettesítést jelent ahol egy adot betüt kicseréljük egy rá nagyon hasonlító másik karakterre például az a betüt a @-ra. Erről bővebben a <https://simple.wikipedia.org/wiki/Leet> oldalon olvashattok.

Ez tulajdonképen egy nagyon egyszerű kis feladat, nem szükséges tulgondolni, bonyolítani a dolgot. A programunk kapni fog egy Angol szót vagy mondatot és a karaktereit át fogja convertálni az adot betü leetes megfelelőjére. Ezt ugy valósítottuk meg, hogy a programban meghatároztuk, hogy melyik betűhöz milyen karakter társul és az így kapott párosok alapján cseréltük ki a betüket. Ha amenyiben a program olyan karaktert kap aminek nincs meghatározva a leetes párja akkor a kapott karaktert adja vissza. Miután a program átfordította a felhasználó által bevitt szöveget lehetőség van egy újabb szöveg leettelésére vagy a programból való kilépésre. Ez úgy lett megoldva, hogy a toLeetCode metódust egy do while-on belül hívtuk meg. A kész kód és a futtatás eredményét lent láthatjuk.

```
class L33tConvertor {

    private String toLeetCode(String str) {
        Pattern pattern = Pattern.compile("[^a-zA-Z]");
        StringBuilder result = new StringBuilder();
        HashMap<Character, String> map = new HashMap<Character, String>();
        map.put('A', "@");
        map.put('B', "B");
        map.put('C', "@");
        map.put('D', "d");
        map.put('E', "€");
        map.put('F', "\u00e1");
        map.put('G', "6");
        map.put('H', "#");
        map.put('I', "!");
        map.put('J', "\u00e1");
        map.put('K', "X");
        map.put('L', "\u00e1");
        map.put('M', "M");
        map.put('N', "r");
        map.put('O', "0");
        map.put('P', "p");
        map.put('Q', "0");
        map.put('R', "@");
        map.put('S', "$");
        map.put('T', "7");
        map.put('U', "\u03bc");
        map.put('V', "v");
        map.put('W', "w");
        map.put('X', "%");
        map.put('Y', "\u00a5");
        map.put('Z', "z");
        result.append(str);
        for (int i = 0; i < str.length(); i++) {
            if (pattern.matcher(str.substring(i, i + 1)).matches()) {
                result.append(map.get(str.substring(i, i + 1).charAt(0)));
            } else {
                result.append(str.substring(i, i + 1));
            }
        }
        return result.toString();
    }

    public void toLeetCode() {
        String str = "Hello World";
        String result = toLeetCode(str);
        System.out.println(result);
    }
}
```

```
for (int i = 0; i < str.length(); i++) {
    char key = Character.toUpperCase(str.charAt(i));
    Matcher matcher = pattern.matcher(Character.toString(key));
    if (matcher.find()) {
        result.append(key);
        result.append(' ');
    } else {
        result.append(map.get(key));
    }
}
return result.toString();
}

public static void main(String[] args) throws IOException {
    L33tConvertor obj = new L33tConvertor();
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String leetWord;
    int cont;
    do {

        System.out.println("\nEnter the English Words :-");
        leetWord = br.readLine();
        String leet = obj.toLeetCode(leetWord);
        System.out.println("The 1337 Code is :- " + leet);

        System.out.println("\n\nDo you want to continue ? [1=Yes and 0=No]");
        cont = Integer.parseInt(br.readLine());
    } while (cont != 0);
}
}
```

```
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Chomsky/Leet$ java L33tConvertor
Enter the English Words :-
Dog
The 1337 Code is :- d0g

Do you want to continue ? [1=Yes and 0=No]
1

Enter the English Words :-
Dogs don't like cats.
The 1337 Code is :- d0g$ d0r' 7 f!X€ @@7$.

Do you want to continue ? [1=Yes and 0=No]
0
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Chomsky/Leet$
```

14.3. Full screen

A feladatunk az, hogy készitsünk egy Java Full screen programot.

Screen.java

```
import java.awt.*;
import javax.swing.JFrame;

public class Screen {
    private GraphicsDevice vc;

    public Screen() {
        GraphicsEnvironment env = GraphicsEnvironment.←
            getLocalGraphicsEnvironment();
        vc = env.getDefaultScreenDevice();
    }

    public void setFullScreen(DisplayMode dm, JFrame window) {
        window.setUndecorated(true);
        window.setResizable(true);
        vc.setFullScreenWindow(window);

        if(dm != null && vc.isDisplayChangeSupported()) {
            try {
                vc.setDisplayMode(dm);
            } catch(Exception ex) {}
        }
    }

    public Window getFullScreenWindow() {
        return vc.getFullScreenWindow();
    }

    public void restoreScreen() {
        Window w = vc.getFullScreenWindow();
        if(w != null) {
            w.dispose();
        }
        vc.setFullScreenWindow(null);
    }
}
```

bucky.java

```
import java.awt.*;
import javax.swing.JFrame;

public class bucky extends JFrame {
    public static void main(String[] args) {

        DisplayMode dm = new DisplayMode(800, 600, 16, ←
            DisplayMode.REFRESH_RATE_UNKNOWN);
        bucky b = new bucky();
        b.run(dm);

    }

    public void run(DisplayMode dm) {
        getContentPane().setBackground(Color.DARK_GRAY);
        setForeground(Color.WHITE);
       setFont(new Font("Arial", Font.PLAIN, 24));

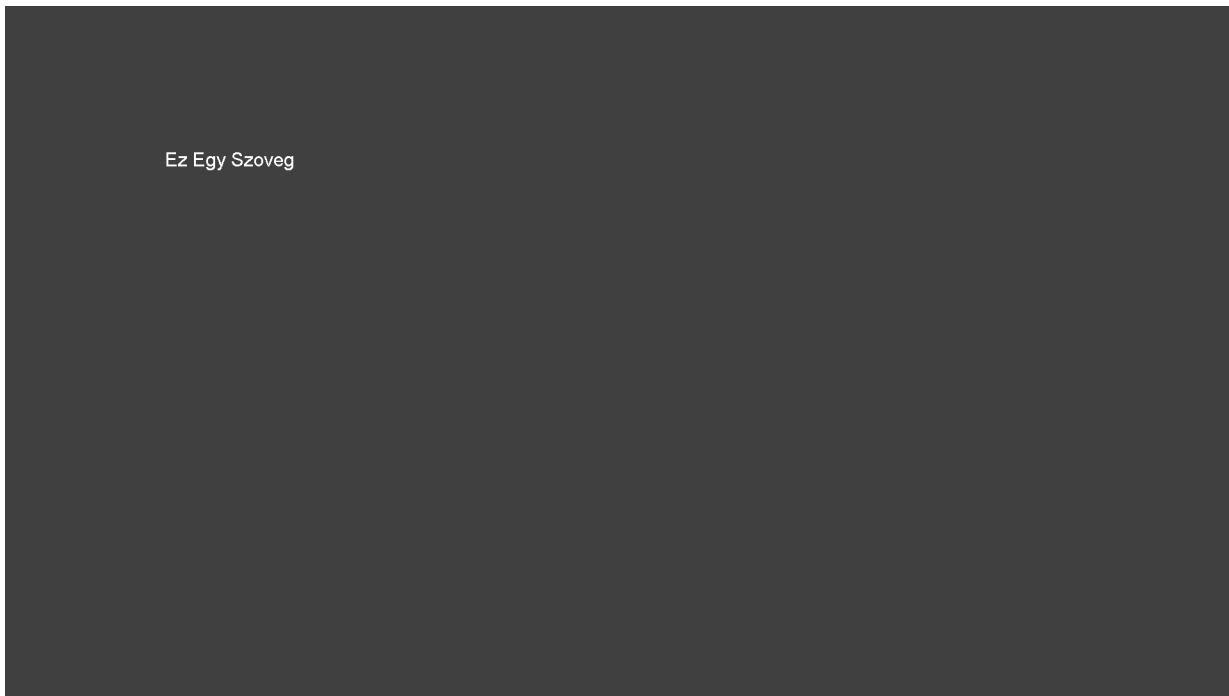
        Screen s = new Screen();
        try {
            s.setFullScreen(dm, this);
            try {
                Thread.sleep(5000);
            }catch(Exception ex) {}
        }finally {
            s.restoreScreen();
        }
    }

    public void paint(Graphics g) {
        super.paint(g);
        g.drawString("Ez Egy Szöveg", 200, 200);
    }
}
```

Láthatjuk, a fullscreen alkalmazásunk két fájlból áll a Screen.java-ban létrehozunk egy ablakot ami a window.setUndecorated(true); sor miatt válik fulscreennek, ugyanis abban az esetben ha false-ra állítjuk akkor egy szabadon átméretezhető ablakot fogunk kapni. És ha már az átméretezésnél tartunk az is true/false-al álítható window.setResizable(); segítségével. Ez nekünk true-ra van állítva, de mivel mi fullscreennel dolgozunk az átméretezés lehetőségét nem fogjuk érezni. A Screen.java álmányban továbbá még nagyon fontos megemlíteni a restoreScreen metódust ami a kijelző vissza állításáért felel, ugyanis nem elegendő a fullscreen ablak létrehozásáról gondoskodnunk mivel annak eltüntetéséjért is mi felelünk.

Most, hogy képesek vagyunk egy teljesképernyös ablak létrehozására kezdenünk is kellen vele valamit, és pontosan erre láthatunk egy egyszerű példát a bucky.java álmányban. Ez minden össze anyit fog tenni hogy kiír egy szöveget a kijelzőre 5 másodpercbe aztán leáll. Azt, hogy az ablakunk, hogy nézzen ki és hogy meddig fusson a run metódusban határoztuk meg. A getContentPane().setBackground(Color.DARK_GRAY); sötétszürkére állítja a háttérét az előttert pedig a setForeground(Color.WHITE);-tal állítjuk fehérré. A setFont-tal állítjuk be a betűtípusát és méretet. A run metódusnak a második fele adja meg, hogy menyi ideig fusson az

alkalmazásunk. Ezt egy try catch-el oldotuk meg ez ha megkapja a displayt akkor altatja a szálat 5 másodpercig utána a restoreScreen-nel eltünteti az ablakunkat. És végül már csak a paint metódusunk maradt ami semmi egyebet nem csinál minthogy kiírja a megadott szöveget.

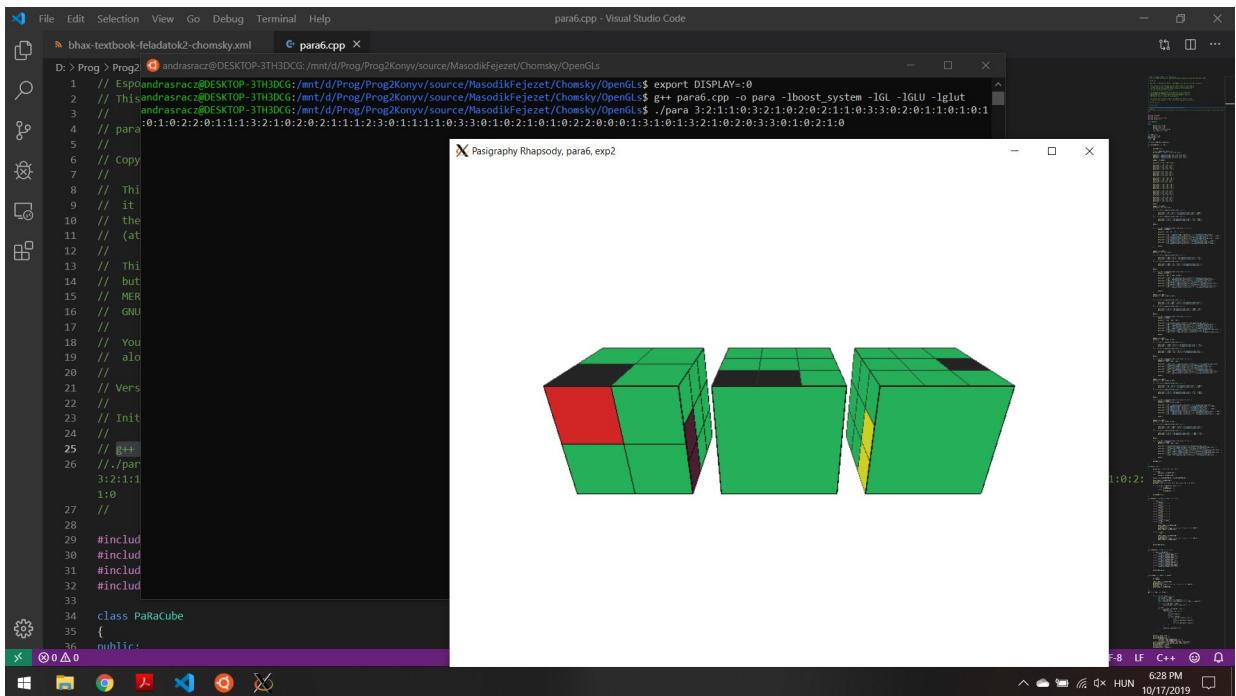


14.4. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Ebben a feladatban a Paszigráfia Rapszódia OpenGL programban kellet apróbb módosításokat eszközölünk. De először ejtsünk néhány szót a Paszigráfia rapszódiáról. Ennek a célja egy univerzális mesterséges nyelv létrehozása és annak széles körű elterjesztése. Ezt úgy valósítja meg, hogy a matematikai logikát használja. Feltehetnénk a kérdést, hogy miért jött ez létre és mire használható. A válasz viszonylag egyszerű és egyben a Paszigráfia Rapszódia létrehozásának motivációját is ez képezi, hogy legyen egy olyan matematikai logikai alapú mesterséges, de természetesen a nemzeti anyanyelveinkre épülő nyelvünk, amely játékok alapjául is szolgálhat, s még egyben a tudomány egyetemes nyelve is lehet.

Most, hogy a Paszigráfia Rapszódiával megismerkedtünk át is térhetünk magára a programra. Mivel windows 10 Linux subsystemmel dolgozunk ezért mindenkelőt szükségünk van az xming alkalmazásra, mivel a subszstem nem képes grafikus alkalmazás futtatására. Miután telepítetük az xminget és elindítottuk megnyitjuk a WSL-t és lefutatjuk a export DISPLAY=:0. Innentől kezdve ebben a terminálban képesek leszünk futtatni grafikus alkalmazásokat. A feladathoz a <https://gitlab.com/nbatfai/pasigraphy-rhapsody/tree/master/para/docs> található para6.cpp-t fogjuk használni.

Ezt a g++ para6.cpp -o para -lboost_system -lGL -lGLU -lglut parancsal fordithatjuk majd a ./para 3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:0:1:0:2:2:0:1:1:3:2:1:0:2:0:2:1:1:1:2:3:0:1:1:1:0:3:3:0:1:0:2:1:0:1:0:2:2:0:0:1:3:1:0:1:3:2:1:0:2:0:3:3:0:1:0:2:1:0 ezzel futtathatjuk. Ha ezeket megtetük akkor már működik is a programunk és láthatjuk a kockákat amik egy logikai kifejezést jelentenek.



Most, hogy ezzel végeztünk és a kódunk működik megváltoztathatjuk a kockák színét amit a glColor3f (.188f, 0.209f, 0.190f) paramétereinek átirásával tehetünk meg. Ez a három szám rgb kódokkal adja meg az adott szint. A második dolog amit megtetünk az irányítás megváltoztatása volt mostmár a forgatások nem inverz módon történik. Ezt a skeyboard függvény módosításaval tudtuk elérni.

Ennek a megvalósítását láthatjuk:

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

    glutPostRedisplay();
}
```

15. fejezet

Helló, Stroustrup!

15.1. JDK osztályok

Ebben a feladatban írnunk kellet egy olyan Boost C++ programot amely kilistáza a JDK összes osztályát. Másszóval végig kell menünk a JDK src folderén és megszámolni, hogy hány .java file található benne.

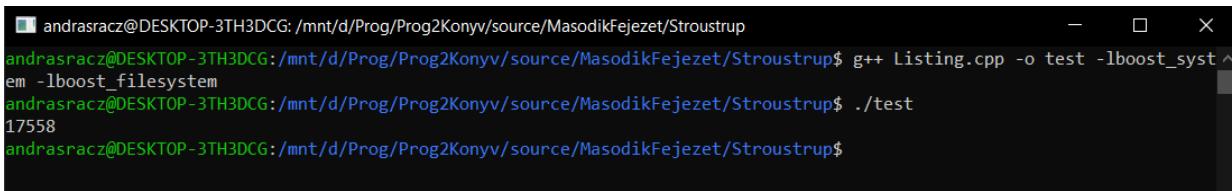
```
#include <boost/filesystem.hpp>
#include <iostream>
using namespace std;
int main(int ac, char** av)
{
    string extension;
    int count = 0;
    boost::filesystem::recursive_directory_iterator iterator(string("/mnt/d/ ←
        Prog/Prog2Konyv/source/MasodikFejezet/Stroustrup/src"));
    while(iterator != boost::filesystem::recursive_directory_iterator())
    {
        string extension = boost::filesystem::extension(iterator->path()). ←
            filename();
        if( boost::filesystem::is_regular_file(iterator->path()) && ←
            extension == ".java") {
            count++;
        }
        ++iterator;
    }
    cout << count << endl;
    return 0;
}
```

A kódöt végigtanulmányozva láthatjuk, hogy az src directory végigjárására a
boost::filesystem::recursive_directory_iterator

használtuk, ami mint neve is mutatja rekurzívan bejárja src könyvtárat. Innen pedig már csak anyi van hátra, hogy meghatározuk, hogy melyik filenak van .java kiterjesztése, amit a

boost::filesystem::extension-el

lehet meghatározni, és ezeket a count változonkba összeszámoljuk. Itt viszont egy hiba csuszott a rendszerbe mivel a boost::filesystem::extension úgy határozza meg egy fájl kiterjesztését, hogy az utolsó ponttól kezdve levágja a file nevének végét és ennek következtében egy olyan mappát is beleszámolt aminek a nevének .java volt a vége. De mint tudjuk a hiba adig nem válik ténylegesen hibává amíg hajlandóak vagyunk kijavitani azt, és ezt meg is tetük a is_regular_file-al ami azt adja meg, hogy a vizsgált elemnek filenak kell lennie. És ezzel imáron pontos eredményt kaptunk(a find \$directory -type f -name "*.java" parancs pontosan ugyan anyi fájlt adot vissza illetve a windows beépített keresője is ezt az eredményt erősítette meg).



```
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Stroustrup$ g++ Listing.cpp -o test -lboost_system -lboost_filesystem
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Stroustrup$ ./test
17558
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Stroustrup$
```

15.2. Hibásan implementált RSA törése és összefoglaló összevonva

A feladatunk az RSA törés hibás implementálása volt, de, hogy ezt meg tudjuk valósítani először az RSA titkosítást kell megismernünk.

Az RSA titkosítás egy olyan rendszer, amely megoldja a kriptográfia egyik legnagyobb problémáját: **Hogyan lehet elküldeni valakinek kódolt üzenetet** anélkül, hogy az üzenetet ismernénk ?

Tegyük fel, hogy valakivel valami titkosat akarunk közölni de nem tartózkodik a közvetlen közelünkbe. Erre megoldást jelenthet az, hogy felhívjuk telefonaikon vagy leírjuk mondandónkat és elküldjük üzenet formájában de ezeket könnyen lehalogathatjuk. Ezt pedig fontos titok esetén nem engedhetjük meg magunknak. Erre a problémára nyújt megoldást a titkosítá. Ez anyit jelent, hogy az üzenethez hozzárendelünk valamilyen kódot ami értelmetlenné alakítja az üzenetet. Ha a kódunk elég bonyolult akkor az üzenethez csak azok féérnek hozzá akik ismerik a kódot. Ezzel csak az az egy probléma van ha a kódot nem tudjuk előzőleg megosztani a másik féllel akkor nem tudja majd dekódolni az üzenetünket. Ezt a kriptográfiai problémát az olyan nyilvános kulcsú titkosítási rendszerek oldoták meg mint például az RSA.

Az RSA titkosítás alatt az üzeneteket egy nyilvános kulcsnak nevezett kóddal titkosítják, amely nyíltan megosztható. Az RSA algoritmus megkülönböztetett matematikai tulajdonságai miatt, miután egy üzenetet titkosítottak a nyilvános kulccsal, csak egy másik kulccsal lehet visszafejteni, az úgynevezett magánkulcsot. minden RSA felhasználó rendelkezik kulcspárral, amely nyilvános és magán kulcsból áll. Ahogy a neve is sugallja a privát kulcsot titokban kell tartani.

James H. Ellis az 1970-es évek elején tette közzé az első nagy fejleményt, amelyet most nyilvános kulcsú kriptográfiának hívunk. Ellis nem találta meg a módját a munkájának végrehajtására, de ezt kollégája, Clifford Cocks továbbfejlesztette, és mostanáig RSA titkosításnak nevezzük. Malcolm J. Williamson, egy másik munkatárs, kitalált egy sémát, amely lehetővé tette két fél számára titkosítási kulcs megosztását, még akkor is, ha a csatornát ellenfelek figyelték meg. Ezt a munkát az Egyesült Királyság hírszerző ügynöksége, a Kormányzati Kommunikációs Központ (GCHQ) végezte, amely a felfedezést minősítette. Részben a technológiai korlátok miatt a GCHQ abban az időben nem látta a nyilvános kulcsú kriptográfia alkalmazását, így a fejlesztés üresen állt a polcon. Csak 1997-ben szüntették meg a munkát, és elismerték az RSA eredeti feltalálóit.

Az RSA működési elve

Az RSA titkosításhoz két kulcs, egy nyílt és egy titkos tartozik. A nyílt kulcs ismert, és ez alapján kódolhtják a nekünk szánt üzeneteket. A nyílt kulccsal kódolt üzenetet csak a titkos kulccsal tudjuk „megfejteni”. Az RSA-eljárásnak a következő módon generáljuk a kulcsokat: Választunk két nagy prím számot, a kettő szorzata lesz az N (ez lesz a modulusa mind a nyilvános, mind a titkos kulcsnak). Számoljuk ki az Euler-féle phi függvény értékét N-re: $\phi(N) = (p-1)(q-1)$. Válasszunk egy olyan egész számot, e-t melyre teljesül $1 < e < \phi(N)$, és e és $\phi(N)$ legnagyobb közös osztója 1. Ezután számítsuk ki d-t, hogy következő kongruencia teljesüljön $d \equiv 1 \pmod{\phi(N)}$. A nyilvános kules az N modulusból és a nyilvános e kifejezésből áll. A titkos kulcs az N modulusból és a titkos d kifejezésből áll, melyeket természetesen nem osztunk meg mással. Üzentküldéskor például A továbbítja a nyilvános kulcsát ($N \& e$ -t) B-nek és a titkos kulcsát titokban tartja. B ezután szeretné elküldeni üzenetét (M) A-nak. Először is M-et számokka alakja darabolja úgy, hogy a kapott m értékre igaz legyen: $m < N$. Ezután kiszámítja c kódszöveget a következő módon: $c = m^d \pmod{N}$. Ez gyorsan végezhető az ismételt négyzetre emeléses hatványozással. B ezután továbbítja „üzenetét” A-nak. Dekódoláskor A saját titkos kulcsát, d-t használva vissza tudja fejteni m-et c-ből úgy, hogy $M = c^e \pmod{N}$. Tudva m-et vissza tudja kapni az eredeti szöveg üzenetet M-et.

Tömören ez a lényege az RSA titkosításnak visszont nekünk egy hibásan működő RSA-t kellet implementálnunk. Ez attól fog hibásan működni mivel a szöveg minden karakterét külön fog kódolni ennek eredménye lesz az, hogy karakterenként lesz egy kulcspárunk. Ilyen módon visszont a szöveget nem tudjuk dekódolni mivel ahhoz az összes kulcsparra szükség lenne. Egy ilyen kódolást fiktív a kulcsok nélküli szinte lehetetlen.

A hibás RSA kódoló

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Comparator;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.*;

public class RSA {
    private final static BigInteger one      = new BigInteger("1");
    private final static SecureRandom random = new SecureRandom();

    private BigInteger privateKey;
    private BigInteger publicKey;
    private BigInteger modulus;

    // generate an N-bit (roughly) public and private key
    RSA(int N) {
        BigInteger p = BigInteger.probablePrime(N/2, random);
        BigInteger q = BigInteger.probablePrime(N/2, random);
        BigInteger phi = (p.subtract(one)).multiply(q.subtract(one));

        modulus      = p.multiply(q);
        publicKey   = new BigInteger("65537");           // common value in practice ←
        = 2^16 + 1
        privateKey  = publicKey.modInverse(phi);
    }
}
```

```
BigInteger encrypt(byte[] bytes) {
    BigInteger swap = new BigInteger(bytes);
    return swap.modPow(publicKey, modulus);
}

BigInteger decrypt(BigInteger encrypted) {
    return encrypted.modPow(privateKey, modulus);
}

public String toString(BigInteger decrypt) {

    byte[] array = decrypt.toByteArray();
    if (array[0] == 0) {
        byte[] tmp = new byte[array.length - 1];
        System.arraycopy(array, 1, tmp, 0, tmp.length);
        array = tmp;
    }
    String str = new String(array, StandardCharsets.UTF_8);
    return str;
}

public static void main(String[] args) {

    int N = Integer.parseInt(args[0]);
    RSA key = new RSA(N);
    System.out.println("key = " + key);

    // create random message, encrypt and decrypt
    // BigInteger message = new BigInteger(N-1, random);

    //// create message by converting string to integer
    String s = "The story of Shakespeare's Hamlet was derived from the ←
        legend of Amleth, preserved by 13th-century chronicler Saxo ←
        Grammaticus in his Gesta Danorum, as subsequently retold by the 16th ←
        -century scholar François de Belleforest. Shakespeare may also have ←
        drawn on an earlier Elizabethan play known today as the Ur-Hamlet, ←
        though some scholars believe Shakespeare wrote the Ur-Hamlet, later ←
        revising it to create the version of Hamlet we now have. He almost ←
        certainly wrote his version of the title role for his fellow actor, ←
        Richard Burbage, the leading tragedian of Shakespeare's time. In the ←
        400 years since its inception, the role has been performed by ←
        numerous highly acclaimed actors in each successive century.";

    byte[] bytes = s.getBytes();
    //BigInteger message = new BigInteger(bytes);
    //BigInteger result = new BigInteger("0");

    List<BigInteger> result = new ArrayList<BigInteger>();
```

```
//list.add(new BigInteger("23456"));

byte[] atmenet = new byte[1];
for(int i = 0; i < bytes.length; i++)
{
    atmenet[0] = bytes[i];
    result.add(key.encrypt(atmenet));
    //result = key.encrypt(atmenet);
}

//BigInteger encrypt = key.encrypt(message);
//BigInteger decrypt = key.decrypt(result);
System.out.println("message = " + s);
System.out.println("encrypted = " + result);
//System.out.println("decrypted = " + key.toString(decrypt));
}
}
```

A feladatunk itt visszont még nem ér véget mivel nekünk meg kellet próbálnunk visszafejteni a hibásan kódolt üzenetünket. Ezt pedig úgy próbáltuk megvalósítani hogy feltételeztük hogy az egyforma betük egyforma kulcsal rendelkeznek. így megeszámláltuk, hogy az egyforma kulcsok egymáshoz képest minden arányba fordulnak elő majd betügyakoriság elve alapján behejtesítetük a betüket. Az így visszakapot szövegen láthatjuk, hogy elégé távol áll a megoldástól ami méginkább megerősíti az RSA titkosítónk hibáját. A dekódolás után csak abban az esetben kaphatunk pontos megoldást ha pontosan sikerül eltalálni a betügyakoriságot. És végül lássuk is a RSA törö megvalósítását:

```
import java.io.*;

class RsaTores {
    public static void main(String[] args) {
        try {
            BufferedReader inputStream = new BufferedReader(new FileReader(←
                "be2.txt"));
            int lines = 0;

            String line[] = new String[10000];

            while((line[lines] = inputStream.readLine()) != null) {
                lines++;
            }

            inputStream.close();

            KulcsPar kp[] = new KulcsPar[100];
```

```
boolean volt = false;
kp[0] = new KulcsPar(line[0]);
int db = 1;

for(int i = 1; i < lines; i++) {
    volt = false;
    for(int j = 0; j < db; j++) {
        if(kp[j].getValue().equals(line[i])) {
            kp[j].incFreq();
            volt = true;
            break;
        }
    }

    if(volt == false) {
        kp[db] = new KulcsPar(line[i]);
        db++;
    }
}

for(int i = 0; i < db; i++) {
    for(int j = i + 1; j < db; j++) {
        if(kp[i].getFreq() < kp[j].getFreq()) {
            KulcsPar temp = kp[i];
            kp[i] = kp[j];
            kp[j] = temp;
        }
    }
}

FileReader f = new FileReader("angol.txt");

char[] key = new char[60];
int kdb=0;
int k;

while((k = f.read()) != -1) {
    if((char)k != '\n') {
        key[kdb] = (char)k;
        //System.out.println(key[kdb]);
        kdb++;
    }
}

f.close();
```

```
        for(int i = 0; i < kdb && kp[i] != null; i++) {
            kp[i].setKey(key[i]);
        }

        for(int i = 0; i < lines; i++) {
            for(int j = 0; j < db; j++) {
                if(line[i].equals(kp[j].getValue())) {
                    System.out.print(kp[j].getKey());
                }
            }
        }
    } catch(IOException e) {
}

}
}
```

15.3. Változó argumentumszámú ctor

Az volt a feladatunk, hogy készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű képet adjon vissza. Ehhez először is fel kellet élesztenünk az prog1-en használt mandelbrot halmazt kirajzóló programot mivel ez fogja majd legenerálni az általunk használt pngt.

Ezzel különösebb dolgunk nem volt a kódot a szokot módon plusz a -lpng kapcsolóval fordítjuk majd futtatjuk csak futatáskor meg kel adni a kimeneti png nevét.

```
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
#define ITER_HAT 32000

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

    clock_t delta = clock ();

    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    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;
// Hány iterációt csináltunk?
int iteracio = 0;
// Végigzongorázzuk a szélesség x magasság rátcsot:
for (int j = 0; j < magassag; ++j)
{
    for (int k = 0; k < szelesseg; ++k)
    {
        reC = a + k * dx;
        imC = d - j * dy;
        reZ = 0;
        imZ = 0;
        iteracio = 0;
        while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
        {
            ujreZ = reZ * reZ - imZ * imZ + reC;
            ujimZ = 2 * reZ * imZ + imC;
            reZ = ujreZ;
            imZ = ujimZ;

            ++iteracio;
        }

        kepadat[j][k] = iteracio;
    }
}

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;
}

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

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

```
int kepadat[MERET][MERET];

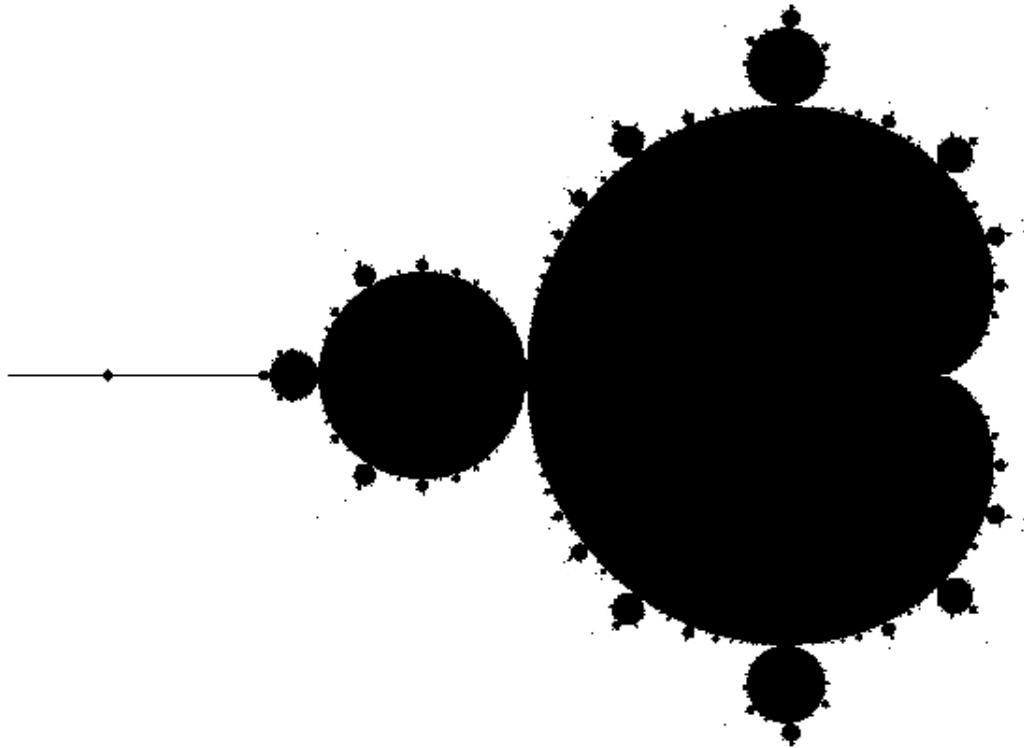
mandel(kepadat);

png::image<png::rgb_pixel> kep(MERET, MERET);

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

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

}
```



Most, hogy a képünk megvan át is térhetünk a perceptronra. A kapott képre azért volt szükség mivel futtatáskor ezt a képet fogja megkapni a perceptron, ami a kép piros részeit a lefoglat tárba másolja és ez alapján új képet állít elő úgy hogy a magasság és a szélesség nem változik.

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

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

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

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

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

double* 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("output.png");

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

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

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

        va_list vap;

        va_start ( vap, nof );

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

        va_end ( vap );

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

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

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

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

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

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

A konstruktorunk, mint láthatjuk változó paraméterszámú ami azért hasznos mert így ugymond több szűrőt tudunk majd használni. Elsőnek a szintek számát kellet meghatároznunk majd az ehez tartozó unitokat és ennek megfelelően meghívtuk a perceptron osztály konstruktorát. Az láthatjuk hogy a () operátor mostmár egy érték helyet egy tömböt fog viszaadni így ennek segítségével már tudunk a képpel dolgozni majd egy új png ként lementeni azt.

```
Perceptron ( std::fstream & file )
{
    file >> n_layers;

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

    for ( int i {0}; i < n_layers; ++i )
    {
        file >> n_units[i];

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

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

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

        for ( int j {0}; j < n_units[i]; ++j )
        {
```

```
weights[i-1][j] = new double [n_units[i-1]];

    for ( int k {0}; k < n_units[i-1]; ++k )
    {
        file >> weights[i-1][j][k];
    }
}

double sigmoid ( double x )
{
    return 1.0/ ( 1.0 + exp ( -x ) );
}

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

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

#ifndef CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );

#else

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

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

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

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

```
    image[i] = units[n_layers - 1][i];

    return image;
}

void learning ( double image [], double q, double prev_q )
{
    double y[1] {q};

    learning ( image, y );
}

void learning ( double image [], double y[] )
{
    //(*this) ( image );

    units[0] = image;

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

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

    int i {n_layers-1};

    for ( int j {0}; j < n_units[i]; ++j )
    {
        backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ( units[i][j] ) ) * ( y[j] - units[i][j] );

        for ( int k {0}; k < n_units[i-1]; ++k )
        {
            weights[i-1][j][k] += ( 0.2* backs[i-1][j] *units[i-1][k] );
        }
    }

    for ( int i {n_layers-2}; i >0 ; --i )
    {

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

            double sum = 0.0;

            for ( int l = 0; l < n_units[i+1]; ++l )

```

```
        {
            sum += 0.19*weights[i][l][j]*backs[i][l];
        }

        backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ( units ←
            [i][j] ) ) * sum;

        for ( int k = 0; k < n_units[i-1]; ++k )
        {
            weights[i-1][j][k] += ( 0.19* backs[i-1][j] *units[i-1][k] ←
                );
        }
    }

    for ( int i {0}; i < n_layers-1; ++i )
    {
        delete [] backs[i];
    }

    delete [] backs;
}

~Perceptron()
{
    for ( int i {1}; i < n_layers; ++i )
    {
        for ( int j {0}; j < n_units[i]; ++j )
        {
            delete [] weights[i-1][j];
        }

        delete [] weights[i-1];
    }

    delete [] weights;

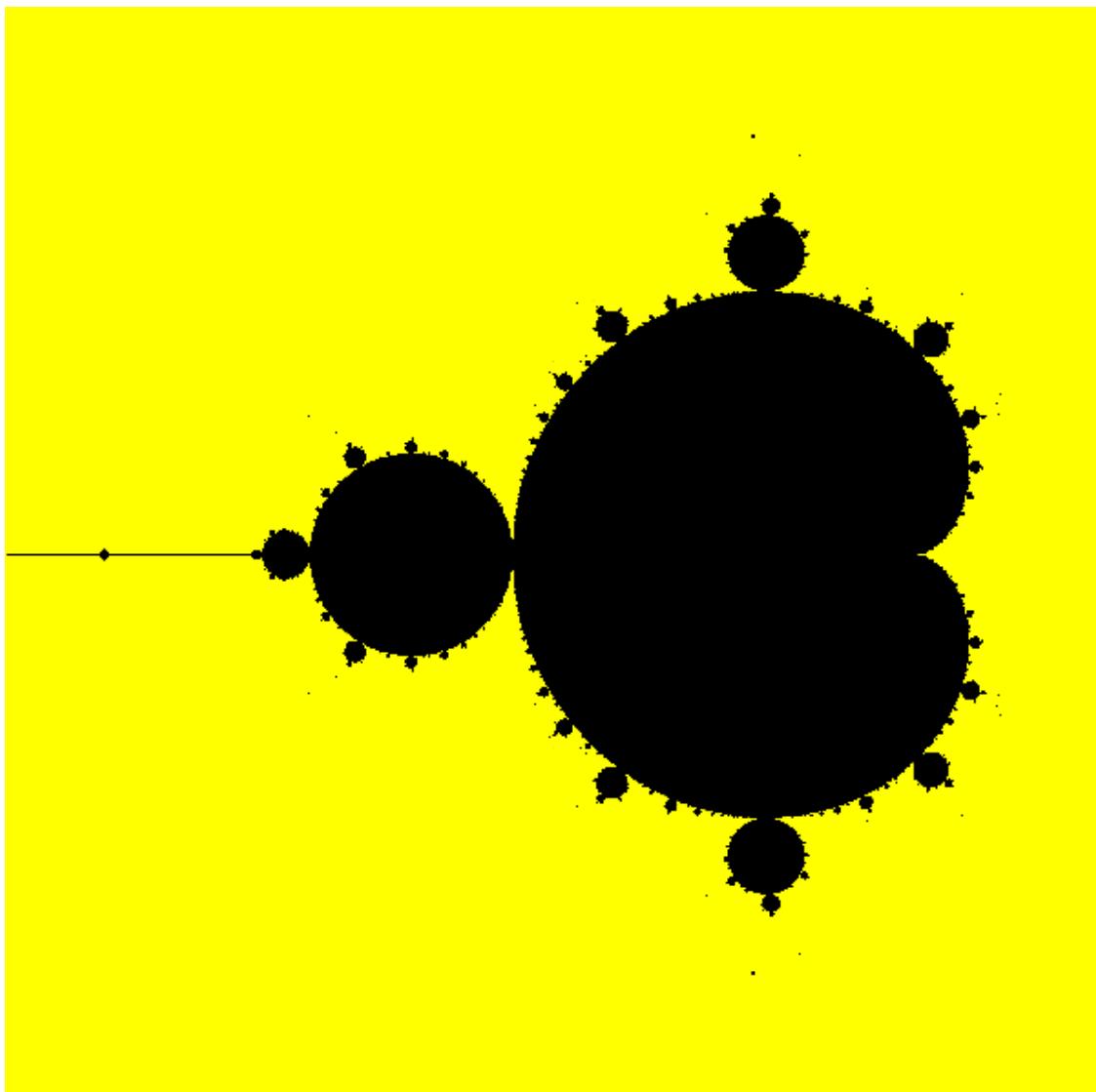
    for ( int i {0}; i < n_layers; ++i )
    {
        if ( i )
            delete [] units[i];
    }

    delete [] units;
    delete [] n_units;
}

void save ( std::fstream & out )
```

```
{  
    out << " "  
    << n_layers;  
  
    for ( int i {0}; i < n_layers; ++i )  
        out << " " << n_units[i];  
  
    for ( int i {1}; i < n_layers; ++i )  
    {  
        for ( int j {0}; j < n_units[i]; ++j )  
        {  
            for ( int k {0}; k < n_units[i-1]; ++k )  
            {  
                out << " "  
                << weights[i-1][j][k];  
  
            }  
        }  
    }  
}  
  
private:  
    Perceptron ( const Perceptron & );  
    Perceptron & operator= ( const Perceptron & );  
  
    int n_layers;  
    int* n_units;  
    double **units;  
    double ***weights;  
};
```

Innen egy double pointert kapunk vissza ami az eredményünket fogja visszaadni amit lejebb meg is tekintetünk.



16. fejezet

Helló, Gödel!

16.1. Gengszterek

Ebben a feladatban a Robotautó világbajnoksággal kellet dolgoznunk még hozzá a gengsztereket lambda segítségével kellet rendeznünk. Ezzen példa alapján most láthatjuk majd a lambda kifejezések előnyeit. A lambda kifejezések inline fügvények írását teszi lehetővé. Ezek olyan rövidke kódsorok amik nem lesznek újra felhasználva így az elnevezésük is fölösleges. Tekintsük is meg az OOCWC kód részletet:

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

Itt az std::sort fügvényt alkalmazzuk a rendezésre. És ez az általunk létrehozott lambda kifejezést felhasználva fog összehasonlítani.

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

Ennek segítségével a gangsters vectort fogjuk rendezni az alapján hogy az adot rendőrhöz melyik gengszer van a legküzelebb. A legközelebi gangster a vektor legelejére kerül.

16.2. STL map érték szerinti rendezése

A feladatunk az volt, hogy érték szerint rendezzük egy map-et. Ehhez egy új gyűjtemény fajtát használunk amivel ebben a feladatban most megismérkedhetünk. Ebben a feladatban ismét a fénykardhoz fogunk hozzájárulni:

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

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

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

return ordered;
}
```

A kódot áttanulmányozva láthatjuk hogy, hogy a map rekordjai egy std::string és egy int típusú kulcs-párpokból áll. Ezekkel a rekordokkal feltöltjük az order vektort, majd ezt fogjuk rendezni érték szerint csökkenő sorrendbe. Ehhez a std::sort() függvényt használva, lambda kifejezés segítségével. Ez akkor térít vissza igaz értéket, ha az első pair értéke nagyobb a második pair értékénél.

16.3. Alternatív Tabella rendezése

Ebben a feladtaban az Alternatív Tabella programban a java.lang Interface Comparable<T> szerepét. Tékinsünk rá a Csapat osztályra ahol a Comparable Interface implementálása megtörténik:

```
class Csapat implements Comparable<Csapat> {
    protected String nev;
    protected double ertek;
    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }
    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        } else {
    }}
```

```
        return 0;
    }
}
}
```

A mi feladatunk az, hogy megmutassuk a Comparable Interface szerepét amire az oracle oldalán megtalálható dokumentációk választ is adnak. Ebben azt olvashatjuk, hogy rendezés csak olyan listákon működik, amelyek minden eleme implementálja a Comparable Interfacet.

16.4. GIMP Scheme hack

Ebben a feladatban a Mandala Scheme-át fogjuk feldolgozni. Ez a GIMP Script egy mandalát fog létrehozni. Anyit kell tennünk, hogy megadunk egy szót, kiválasztunk nekünk tetsző szint és betütípus és a script ezen adatok felhasználásából létrehoza a mandalánkat. Mandalát ugy készítünk, hogy a képünket le-másoljuk elforgatjuk 90 fokkal majd egymásra illesztjük a két réteget és összevonjuk azokat, majd ezeket a lépéseket ujra megismételjük 45 és 30 fokkal és rajzolunk két különböző sugarú kört a képünk köré. Ennek implementálását láthatjuk:

```
(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)))
```

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

(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"
)
```

Először is meghatározzuk a szöveg hoszat a gimp-text-get-extents-fontname-el ami egy listát fog viszaadni aminek az első eleme a szöveg hossza. Erre azéet van szükség, hogy szebb képet hozzunk létre. Ezután elvégezzük a különböző szögekkel való elforgatást. A kód legvégén láthatjuk hogy létrehozunk egy ablakot amiben a felhasználó meg tudaj adni hogy milyen szóból szeretné elkészíteni a mandalát illetve megadhatja a betűtípus, a betűszínt és betűmérettet.

17. fejezet

Helló, !

17.1. FUTURE tevékenység editor

Ebben a feladatban tutorált Szegedi Csaba.

Az volt a feladatunk, hogy javítsunk valamit az ActivityEditor.java JavaFX programon. A program rengeteg hibája közzül mi azt javítotuk ki, hogy egy mappában csatlakoztassuk a tevékenységeket. Tekintsük is meg ennek az okát:

```
java.io.File f = new java.io.File(file.getPath() + System.getProperty("file.separator") + "Új altevénység");

if (f.mkdir()) {
    javafx.scene.control.TreeItem<java.io.File> newAct
        = new FileTreeItem(f, new javafx.scene.image.ImageView(actIcon));
    getTreeItem().getChildren().add(newAct);
```

Ha utánanézünk a java.io.File.mkdir()-nek akkor láthatjuk, hogy a java.io.File a fájl elérési útvonalát kell tartalmaznia és erre meghívva az mkdir() tagfüggvényt, létrehozza a mappát. Ellenben ha egy mappa már létezik az adott elérési utvonalon, akkor az mkdir nem tud létrehozni új mappát ugyanazon névvel. Ez fogja adni a hibánkat, de lásuk is enek a hibának a megoldását:

```
int i = 1;
while (true) {
    java.io.File f = new java.io.File(
        file.getPath() + System.getProperty("file.separator") + "Új altevénység" + i);
    if (f.mkdir()) {
        javafx.scene.control.TreeItem<java.io.File> newAct
        // rr.println("Cannot create " + f.getPath())rr.
        // println("Cannot create " +
        // f.getPath())rr.println("Cannot create " + f. +
        // getPath())rr.println("Cannot
```

```
// create " + f.getPath()) = new javafx.scene. --  
    control.TreeItem<java.io.File>(f,  
    // new javafx.scene.image.ImageView(actIcon));  
    = new FileTreeItem(f, new javafx.scene.image. --  
        ImageView(actIcon));  
    getTreeItem().getChildren().add(newAct);  
    break;  
} else {  
    i++;  
    System.err.println("Cannot create " + f.getPath());  
}  
}  
});
```

Mint láthatjuk a problémát úgy oldottuk meg, hogy a mappa készítést egy ciklusba helyeztük ami az elérési úthoz hozzáadja az i változót. Ez a változó azt számolja, hogy hányszor akkartunk már új mappát létrehozni, ha a mappa létezik simán urjakezdjük eggyel nagyobb értékkel és a mappa nevének végre rakjuk a számot. Abban az esetben egy breakkel kilépünk a ciklusból.

17.2. OOCWC Boost ASIO hálózatkezelése

Ebben a feladatban a scanf szerepére kellet rámutatnunk a carlexer.ll kódban.

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

Az sscanf formázott stringből olvas be, erre azért van szükségünk mivel nem tudjuk, hogy hány gangster adatot tudunk majd beolvasni, ezért adig olvasuk be az adatokat amíg azok validak. Ennek ellenőrzésére pedig a sscanf kívályó mivel ez a beolvasot paraméterek számát jelöli. Ha ez négy lesz akkor egy gangster összes adatát beolvastuk és létrehozunk egy új gangster objektumot és beleteszük a vektorunkba.

17.3. SamuCam

Ebben a feladatban a webcam kezelésére kellet rámutassunk a Samu projectben. A kamera kezeléséhez az openCv könyvtárat használjuk. Mivel nekünk a webcam kezelését kell tanulmányoznunk ezért mi a samuCam forrásfájlhoz kell nyulnunk.

Az alábbi kód részben láthatjuk, hogy nyitunk egy video streamet, a videoCaptur open fügvénye pedig megnyitja a paraméterként kapott képet amit a mi esetünkben a webcamera fogja szolgáltatni. Itt még beállítjuk a video méretét és az FPS-ét is.

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

A kép elemzésére szükségünk lesz egy CascadeClassifier-re amit még a futás kezdete előt betölünk. Majd a load függvényel egy classifert nyitunk meg amibe az arcot tároljuk:

```
cv::CascadeClassifier faceClassifier;
std::string faceXML = "lbpcascade_frontalface.xml"; // https://github. ↵
    com/Itseez/opencv/tree/master/data/lbpcascades
if ( !faceClassifier.load ( faceXML ) )
{
    qDebug() << "error: cannot found" << faceXML.c_str();
    return;
}
```

A következő kódcsípetben 80 milliszekundumonkét, a read függvény segítségével, beolvassunk egy képkockát amit a cv::Mat tömbbe mentjük bele. Ezután átméretezzük a képet és a color spacet átalítjuk grayscalere. Ezután az inputképen különböző méretű objektumokat keresünk a képen amihez a detectMultiScale függvényt használjuk. A találatokat egy listaként téritjük vissza. A talált képből QImage-t készítünk amit majd a SamuBrain fog feldolgozni.

```
while ( videoCapture.read ( frame ) )
{
    if ( !frame.empty() )
    {
        cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0, cv:: ↵
            INTER_CUBIC );

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

```
cv::Mat grayFrame;

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

faceClassifier.detectMultiScale ( grayFrame, faces, 1.1, 4, ←
    0, cv::Size ( 60, 60 ) );

if ( faces.size() > 0 )
{
    cv::Mat onlyFace = frame ( faces[0] ).clone();

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

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

    emit faceChanged ( face );
}

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

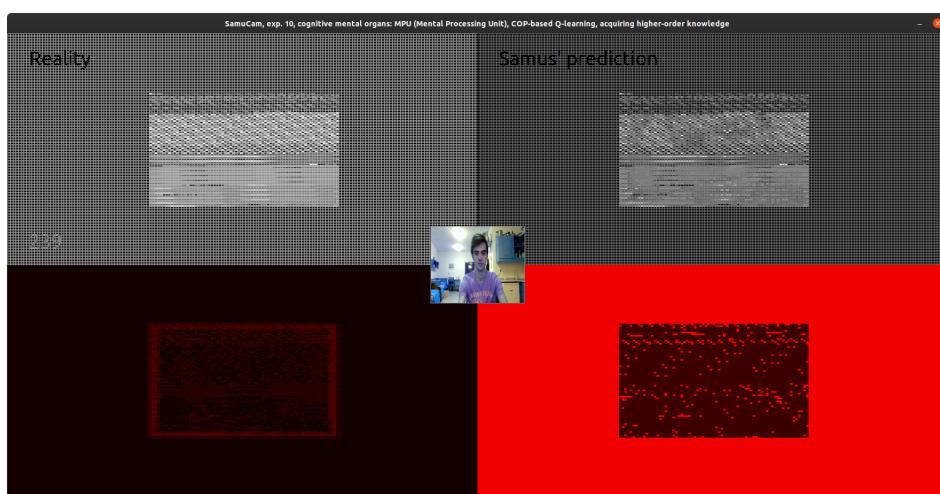
emit webcamChanged ( webcam );

}

QThread::msleep ( 80 );
}
```

A mi esetünkben még egy plusz átirásra is volt szükség mivel mi nem ip camerát használtunk hanem a laptop saját webcam-ját. Ez annyit fog csinálni hogy videoCapture.open nem egy fájl nevét hanem a kamera device ID-ját fogja megkapni.

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



17.4. BrainB

A BrainB már ismerősen hangozhat mivel ezzel már az első fejezetben foglalkoztunk, de ezuttal a Qt slot-signal mechanizmust kell bemutatnunk. Ez a Qt-ban nem másra jó mint objektumok közti kommunikációra. Ez úgy működik, hogy kibocsát egy signált egy adott esemény következtére amit egy slot fog elkapni. A slot egy olyan függvény ami bizonyos signalok esetén meghívodnak. Egy signalt és egy slotot a connect függvényel tudjuk összekapcsolni. A BrainB projectben két ilyen connectet találhatunk a BrainBWin.cpp fájlban:

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

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

Az első connect azt csinálja, hogy ha a brainBThread-ben bekövetkezik a heroesChanged signal akkora a BrainBWin update-Heroes függvénye fut le, endAndStats esetén lejárt a futási idő miatt hatására kiíródnak az eredmények.

```
void BrainBThread::run()
{
    while ( time < endTime ) {
        QThread::msleep ( delay );
        if ( !paused ) {
            ++time;
            devel();
        }
    }
}
```

```
        draw();
    }
    emit endAndStats( endTime );
}
void BrainBWin::endAndStats( const int &t )
{
    qDebug() << "\n\n\n";
    qDebug() << "Thank you for using " + appName;
    qDebug() << "The result can be found in the directory " + statDir;
    qDebug() << "\n\n\n";
    save( t );
    close();
}
```

```
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();
}
```

Illetve végül a felélesztet brainb:



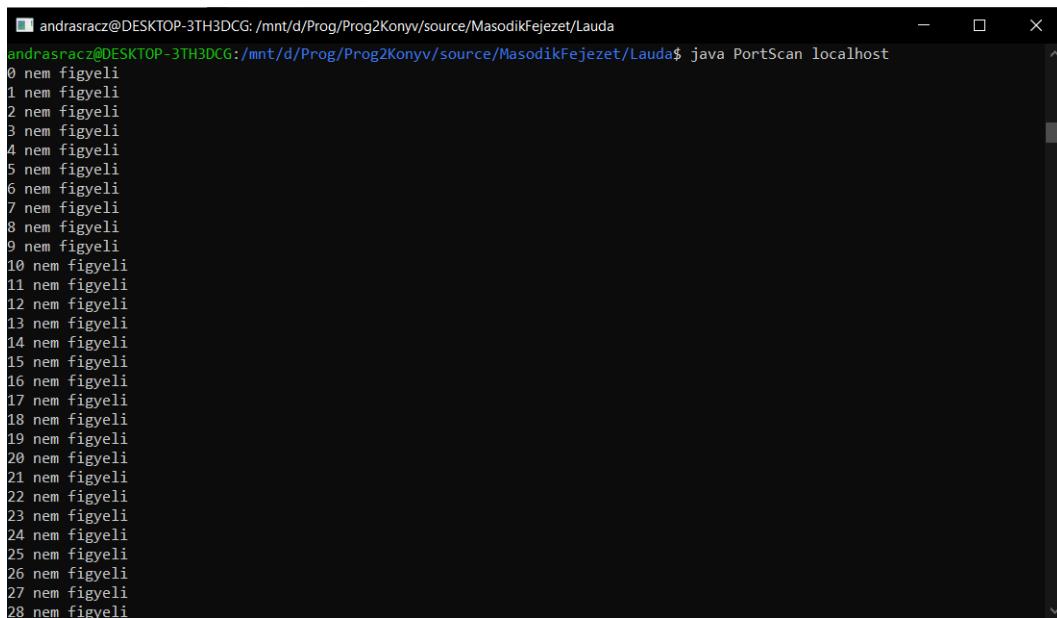
18. fejezet

Helló, Lauda !

18.1. Port Scan

Ebben a feladatban a port szkennelő forrásban a kivétel kezelés szerepére kellet rámutatnunk. Ennek bemutatására írtunk egy egyszerű programot. Ez minden portot 0-tól 1024-ig figyeli, és minden porton kiírja, hogy figyel és bezárja a soketet, ha pedig nem akkor a kivételkezelés catch blokkjában kiírja hogy nem figyel.

```
public class PortScan {
    public static void main(String[] args)
    {
        for(int i=0; i<1024; ++i)
            try {
                java.net.Socket socket = new java.net.Socket(args[0], i);
                System.out.println(i + " figyeli");
                socket.close();
            } catch (Exception e)
            {
                System.out.println(i + " nem figyeli");
            }
    }
}
```



```
andrasracz@DESKTOP-3TH3DCG:/mnt/d/Prog/Prog2Konyv/source/MasodikFejezet/Lauda$ java PortScan localhost
0 nem figyeli
1 nem figyeli
2 nem figyeli
3 nem figyeli
4 nem figyeli
5 nem figyeli
6 nem figyeli
7 nem figyeli
8 nem figyeli
9 nem figyeli
10 nem figyeli
11 nem figyeli
12 nem figyeli
13 nem figyeli
14 nem figyeli
15 nem figyeli
16 nem figyeli
17 nem figyeli
18 nem figyeli
19 nem figyeli
20 nem figyeli
21 nem figyeli
22 nem figyeli
23 nem figyeli
24 nem figyeli
25 nem figyeli
26 nem figyeli
27 nem figyeli
28 nem figyeli
```

18.2. AOP

Ebben a feladatban az LZWBInFába kelet beleszönünk egy átfogó vonatkozást. Itt aspektus orientáltságról van szó. Ezek az aspektusok eltérhetnek majd az eredeti programban használt programozási nyelvtől de akár meg is egyezhetnek azzal. Ezek aspektusokat arra adnak lehetőséget, hogy az alprogramunk működésén tudunk változtatni átfogó vonatkozásokkal. Ahol az alprogram és az aspektus kapcsolódik azt join pointnak nevezük.

Én a BinFa preorder postorder kiírást szöttem bele a kódba.

```
package szoves;
import java.io.FileNotFoundException;
import java.io.PrintWriter;

public aspect szoves {

    int melyseg = 0;
    public pointcut kiir(LZWBInFa.Csomopont elem, java.io.PrintWriter os)
        : call(public void kiir(LZWBInFa.Csomopont, java.io.PrintWriter)) && ↵
            args(elem,os);

    after (LZWBInFa.Csomopont elem, java.io.PrintWriter os) : kiir(elem,os)
    {
        try {
            preOrder(elem,new PrintWriter("pre-order.txt"));
        } catch (FileNotFoundException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        melyseg = 0;
    }
}
```

```
try {
    postOrder(elem, new PrintWriter("post-order.txt"));
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
}

public void preOrder(LZWBInFa.Csomopont elem, java.io.PrintWriter os) {
    if (elem != null) {

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

public void postOrder(LZWBInFa.Csomopont elem, java.io.PrintWriter os) {
    if (elem != null) {
        ++melyseg;
        postOrder(elem.egyesGyermek(), os);
        postOrder(elem.nullasGyermek(), os);
        --melyseg;

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

Láthatjuk, hogy maga az aspektus a kiir függvény meghívásakor kapcsolódik bele a LZWBInFa programba

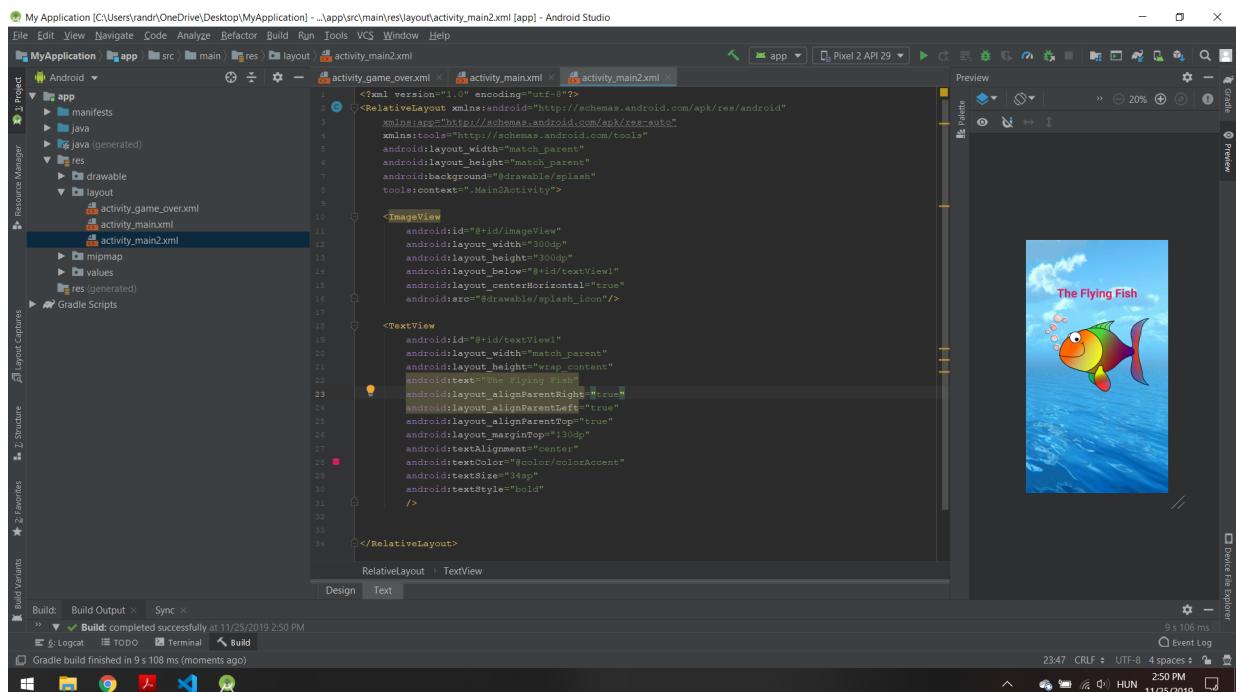
és kiegészít a kiir függvényt a postoder és preorder faépítésel is.

18.3. Android játék

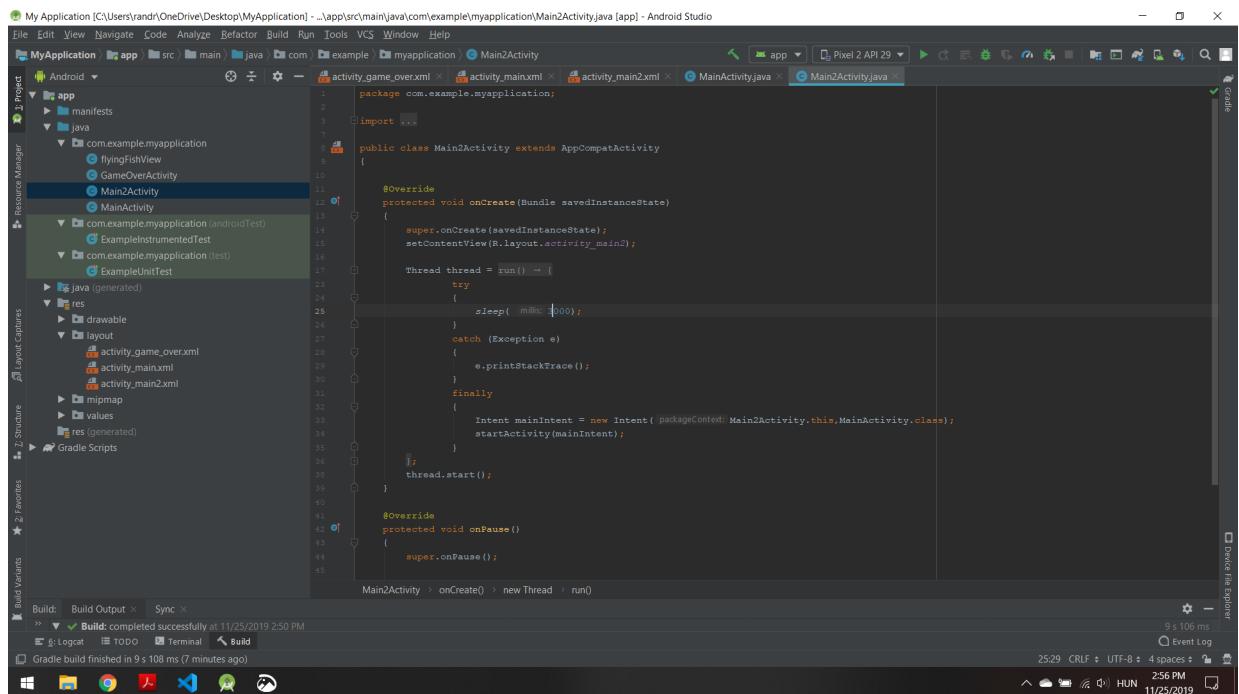
Ebben a feladatban egy android játékot kellet írnunk. Fontos megemlíteni, hogy az általam elkészítet játék egy a kezdő android game developereknek szánt tutoriál alapján készült (ez a tutoriál szolgáltata a játékhoz használt pngket). Ennek fő oka az volt, hogy még sosem mélyedtem bele az android programozásba, nem igazán tudtam, hogy hol kezdjem a feladat megoldását. Mindezek ellenére nagyon hasznosnak véltem ezt a feladatot ugyanis számomra ez rengeteg újdonságot tanított. De nézük is magát a játékot.

A játék maga nem bogyolult úszik egy kis hal amivel különböző bogyokat vehetünk fel. Ezek a bogyok közzül a sárga 10, a zöld pedig 15 pontot ér. Végül pedig vannak a piros golyók amiknek a felvétele életvesztésel jár. A harmadik piros golyó felvétele után a játék véget ér, és láthatjuk, hogy mennyi pontot értünk el.

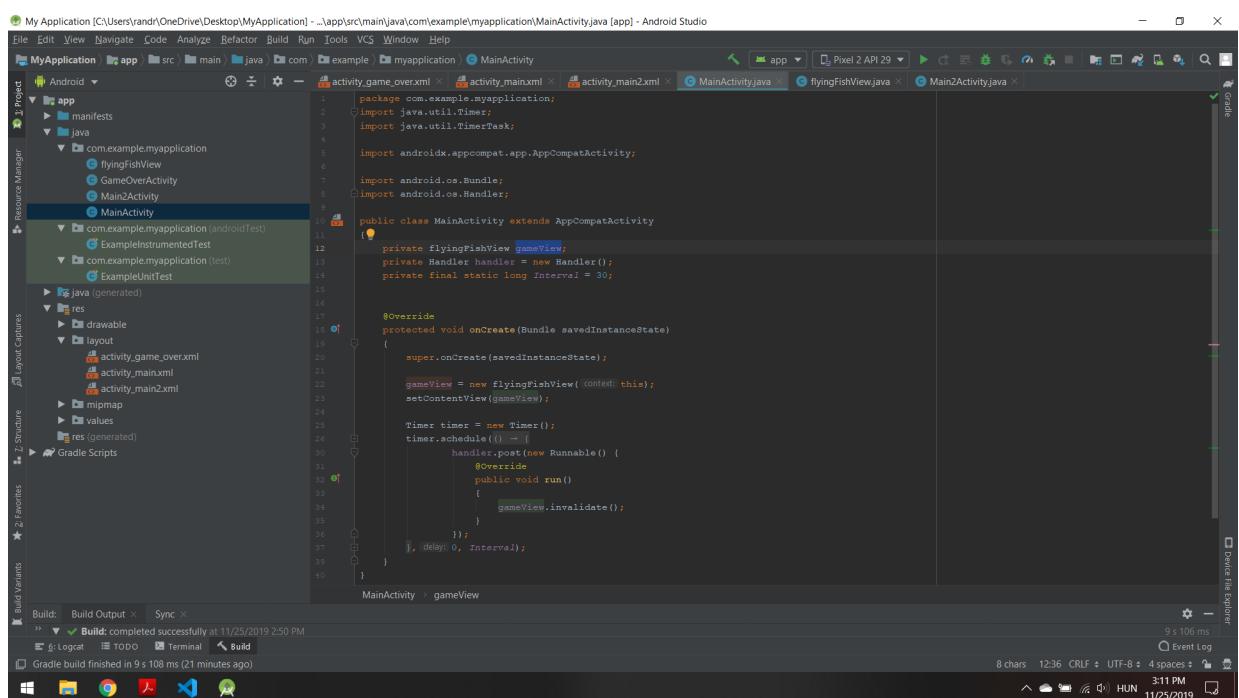
Most, hogy nagyabból tudjuk, hogy mit csinál a játék nézzük meg magát a kódot.



Láthatjuk, hogy először a kis kezdöképernyönk layoutját hozzuk létre, adunk neki egy háttérrel illetve elhelyezünk rajta egy szövegdobozt iletve egy képet.



Az onCreate metódusban láthatjuk, hogy ez a layout mindössze 2000 milliszekundumig lesz látható aztán meghívásra kerül a mainActivity.



A mainActivity osztályban történik maga a játék futása amit ugy valositottuk meg hogy létrehoztunk egy gameView változót ami a játék layoutja és ezen keresztül hívodik meg a flyingFishView osztály ami a játék szabályait illetve a mozgó objetumokat fogja tartalmaznia.



Mint a képen láthatjuk a flyingFishView függvényben a beállítjuk, azokat az objektumokat amiket majd a gameViewban meg szeretnénk jeleníteni például a halacskát, a felvehető golykat, az életet jelző szivet és a háttér képet is.

```
protected void onDraw(Canvas canvas)
{
    super.onDraw(canvas);

    canvasWidth = canvas.getWidth();
    canvasHeight = canvas.getHeight();

    canvas.drawBitmap(backgroundImage, 0, 0, null);

    int minFishY = fish[0].getHeight();
    int maxFishY = canvasHeight - fish[0].getHeight() * 3;
    fishY = fishY + fishSpeed;

    if (fishY < minFishY)
    {
        fishY = minFishY;
    }
    if (fishY > maxFishY)
    {
        fishY = maxFishY;
    }
    fishSpeed = fishSpeed + 2;

    if (touch)
    {
        canvas.drawBitmap(fish[1], fishX, fishY, null);
        touch = false;
    }
    else
    {
```

```
        canvas.drawBitmap(fish[0], fishX,fishY,null);
    }

yellowX = yellowX - yellowSpeed;

if(hitBallChecker(yellowX,yellowY))
{
    score = score + 10;
    yellowX = -100;
}

if(yellowX < 0)
{
    yellowX = canvasWidth + 21;
    yellowY = (int) Math.floor(Math.random() * (maxFishY-minFishY)) ←
              + minFishY;
}
canvas.drawCircle(yellowX,yellowY,25,yellowPaint);

greenX = greenX - greenSpeed;

if(hitBallChecker(greenX,greenY))
{
    score = score + 15;
    greenX = -100;
}

if(greenX < 0)
{
    greenX = canvasWidth + 21;
    greenY = (int) Math.floor(Math.random() * (maxFishY-minFishY)) ←
              + minFishY;
}
canvas.drawCircle(greenX,greenY,10,greenPaint);

redX = redX - redSpeed;

if(hitBallChecker(redX,redY))
{
    redX = -100;
    lifeCounterOffFish--;

    if (lifeCounterOffFish == 0)
    {
        Toast.makeText(getContext(), "Game Over", Toast.←
                      LENGTH_SHORT).show();
    }
}

Intent gameOverIntent = new Intent(getContext(), ←
```

```
        GameOverActivity.class);
    gameOverIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK | ←
        Intent.FLAG_ACTIVITY_CLEAR_TASK);
    gameOverIntent.putExtra("Score", score);
    getContext().startActivity(gameOverIntent);
}
}

if(redX < 0)
{
    redX = canvasWidth + 21;
    redY = (int) Math.floor(Math.random() * (maxFishY-minFishY)) + ←
        minFishY;
}
canvas.drawCircle(redX, redY, 50, redPaint);

canvas.drawText("Score : " + score, 20, 60, scorePaint);

for (int i=0; i<3; i++)
{
    int x = (int) (580 + life[0].getWidth() * 1.5 * i);
    int y = 30;

    if (i < lifeCounterOffish)
    {
        canvas.drawBitmap(life[0], x,y,null);
    }
    else
    {
        canvas.drawBitmap(life[1], x,y,null);
    }
}

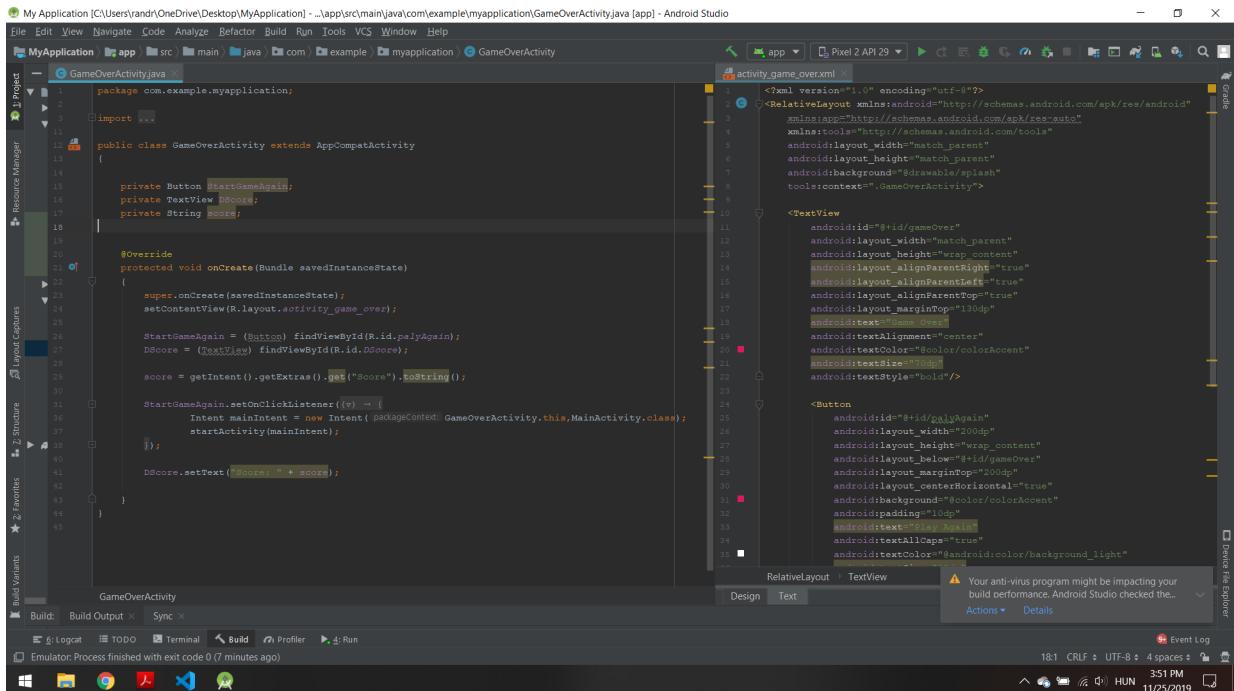
}
```

Az előbb említet objektumokat itt az onDraw-ban fogjuk elhelyezni a vászonon. Láthatjuk, hogy a golyókat egy random szám segítségével helyezük el, erre azért vna szükség, hogy a golyok ne legyenek kiszámithatóak a felhasználó számára. Itt még láthatjuk, hogy megadjuk a golyok méretét, sebeségét, illetve, hogy hány pontot fognak érni. Továbbá if elsek segítségével itt figyeljük, hogy a halacska milyen golyot talált el illetve a játék erre, hogy reagáljon. Azt hogy találtunk e el golyot a hitBallChecker metódus figyeli.

```
public boolean hitBallChecker(int x, int y)
{
    if( fishX < x && x < (fishX + fish[0].getWidth()) && fishY < y && y ←
        < (fishY + fish[0].getHeight()))
    {
        return true;
    }
}
```

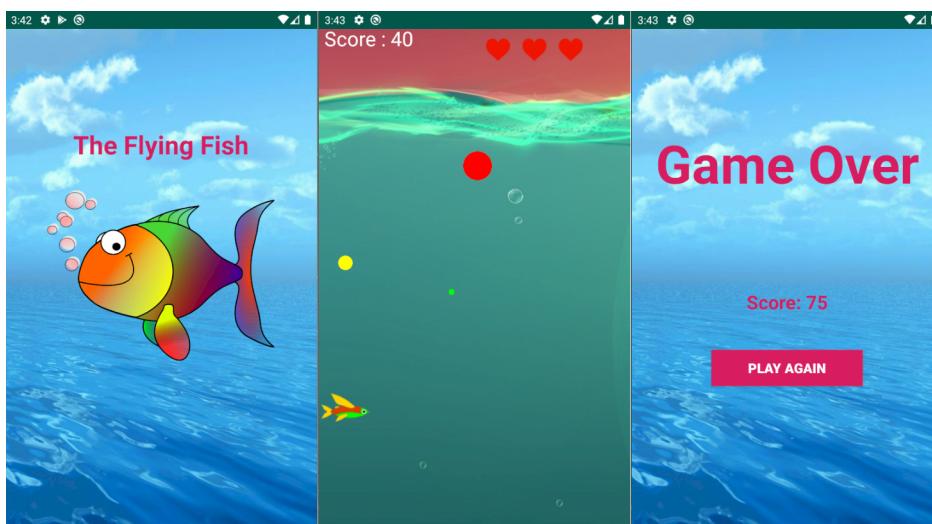
```
        return false;
    }
}
```

A hitBallChecker csak anyit csinál, hogy összeveti a golyok és a halcska poziciját és ha megegyezik akkor true-t ír vissza egyébként pedig falset.



Ha elfogyot az összes életünk a játék végetér és a gameView layoutot leváltja a GameOver layout ami láthatjuk hogy hány pontot értünk el, illetve ha tovább szeretnénk játszani akkor van egy play again gomb is.

És végül lásunk néhány képet a játékról:



18.4. Junit teszt

Ebben a feladatban a [!\[\]\(8eed8cbbb83d6c59a4f97969e09cd314_img.jpg\)

The screenshot shows the Eclipse IDE interface with the following details:

- Title Bar:** Full - BinFaTest/src/BinFa/BinFaTest.java - Eclipse IDE
- Menu Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help
- Toolbars:** Standard, Java, Java Editor, JUnit
- Project Explorer:** Shows a project named LZWBinFajava with a BinFaTest folder containing a BinFaTest.java file.
- Code Editor:** Displays the following Java code:

```
1 package BinFa;
2
3 public class BinFaTest {
4     @org.junit.Test
5     public void tesBitfeldolg\(\) {
6         LZWBinFa binfa = new LZWBinFa\(\);
7         for \(char c : "0111001001000111".toCharArray\(\)\) {
8             binfa.egyBitfeldolg\(c\);
9         }
10        org.junit.Assert.assertEquals\(4, binfa.getMelyseg\(\), 0.0\);
11        org.junit.Assert.assertEquals\(2.75, binfa.getHtleg\(\), 0.001\);
12        org.junit.Assert.assertEquals\(0.997427, binfa.getZoras\(\), 0.0001\);
13    }
14 }
15 }
```

- Outline View:** Shows the class structure: BinFa > BinFaTest > binfa: LZWBinFa > tesBitfeldolg\(\)
- Failure Trace:** Empty
- Bottom Status Bar:** Writable, Smart Insert, 17:1:429, 11:59 AM, HUN, 11/29/2019](https://progpater.blog.hu/2011/03/05/labormeres_oththon_avagy_hogyan_dolgozok_fel_egy_poszt_kézzel_számított_mélységét és szórását kellet egy Junit tesztbe beledolgozni.</p></div><div data-bbox=)

Mint láthatjuk a posztnak található bitsorozatot használtuk. Az egyBitFeldog metódussal fogjuk feldolgozni a karakterenként a megadott sztring char tömb változatát. Végül az assertEquals metódussal ellenőrizzük, hogy a számított értékek megegyeznek-e a mi értékeinkkel. Mint láthatuk a fenti képen az értékeink meggyeztek így, azaz a programunk jól működik.

19. fejezet

Helló, Calvin !

19.1. MNIST

Ebben a feladatban a MNIST python programot kellet felélesztenünk illetve egy saját képet kellet felismertesünk vele. Ez azért volt érdekes mivel ez a program egy tensorflow-s gépi tanulást reprezentál, pontosabban a kézírás felismerését tanítjuk meg neki.

A kód felélesztése nem volt anyira nehéz mivel csak az elavult szintaktikát kellet átírni az újabb verzionak megfelelően. Elöször is a

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

sort át kell irnunk a

```
cross_entropy= tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(←  
    logits = y, labels=y_))
```

sorra.

Ez után már csak egy változtatást kel eszközöljünk a kódba ahhoz, hogy a programot működésre bírjuk. Ezt pedig nem máshol kell megténnünk mit a readimg függvény img változó megadásában.

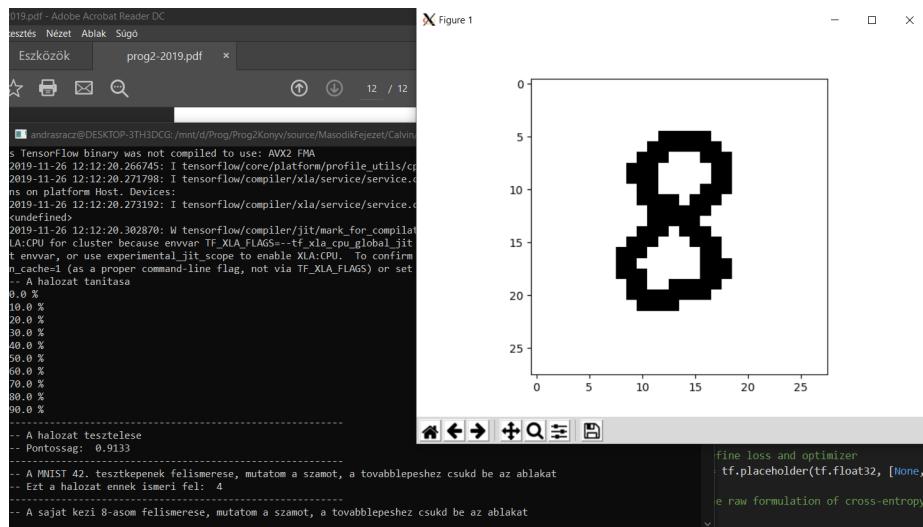
```
img = tf.image.decode_png(file, 1)
```

Mint láthatjuk a file után veszővel elválasztva írtuk egy egyest, erre azért volt szükség mert az új verzióban már meg kel adin a decode_png függvénynek a color channel-ek számát ami a mi esetünkben azért egy mivel a képünk grazscale lesz.

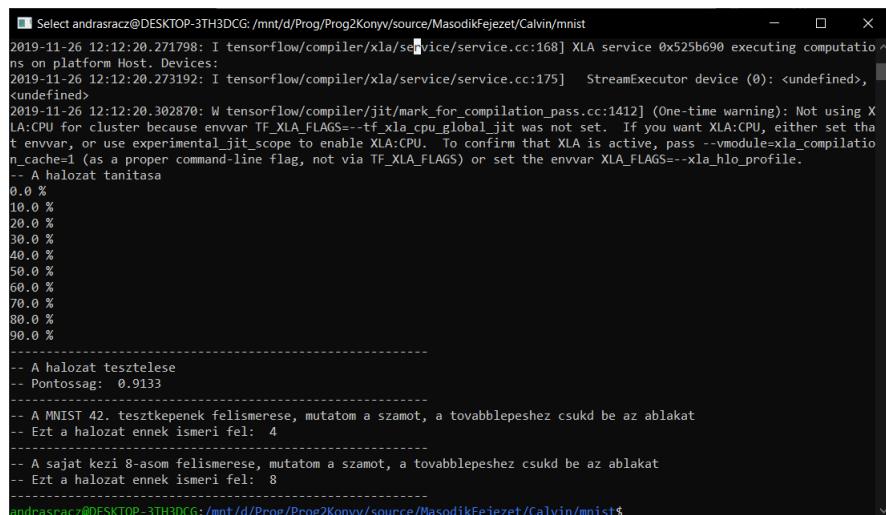
Most, hogy végeztünk a szükséges változtatásokkal és újra működik a program átérhetünk a feladat második részére és beadhatjuk neki a saját képünket.

8

Ez ugy tudjuk megteni, hogy a readimg függvénybe a read file függvénynek megadjuk a képünk nevét. Itt meg kell említeni, hogy ennek a képnak 28*28asnak kell lennie, illetve ha lehet fekete hátréten fehér betűszín legyen. Ha beadtuk a képet akkor a programot futtatva láthatjuk, hogy a program 0.9133 pontosággal fogja megálapítani, az általunk megadot számáról, hogy az melyik szám. Illetve kirajzolja a beadot számunkat grayscalezve.



Ha bezárjuk a megyilt ablakot amin a nyolcasunkat láthatjuk akor a program kiirja, hogy a hálózat ezt minek isméri fel.



Mint láthatjuk a képünket a program 8-nak ismerte fel ami azt jelenti, hogy a programunk jól működik.

19.2. Deep MNIST

Ez a feladat sok aspektusában megegyezik az előző feladattal csupán annyi különbség van, hogy itt az MNIST program mély változatával fogunk foglalkozni. A mély vérzió abban különbozik az előző feladatban használt Soft verziótól, hogy it a képet már több köztes rétegen szürjük át.

Mint már az előbb említetem a feladatunk itt is ugyan az, saját kép beadása a programnak. Itt viszont több módosítást kellet végeznünk mivel a kód sem fájlbeolvasásra sem a beolvasot kép vizsgálatár, illetve anak

eredményének kiírására sem volt képes. De nézük is, hogy mit modosítottunk. Először is be kelett olvasnunk a képünket ami most a mi esetünkbe az előző feladatban használt nyolcas lesz. Ezt ugy oldotuk meg, hogy létrehoztuk a DeepMnist kódban is a readimg fügvényt.

```
def readimg():
    file = tf.io.read_file("gray.png")
    img = tf.image.decode_png(file, 1)
    return img
```

Mit láthatjuk ez hasonló az előző feladatban használtal, a file változó megkapja a képet aminek a grayscalelt változatát átadjuk az img változónak amivel a programunk majd dolgozni fog. Ezután átérhetünk a kép felismerésére.

```
print("-- A sajat kezi 8-asom felismerese, mutatom a szamot, a ←
      továbblepeshet csukd be az ablakat")

image = img.eval(session=sess)
image = image.reshape(28*28)

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.←
                        cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y_conv, 1), feed_dict={x: [image], ←
                keep_prob: 1.0})
saver.save(sess, "model.ckpt")
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
```

Mint láthatjuk ez majdnem teljesen megegyezik a Mnistben lévő vizsgálattal, csupán annyi dolgunk volt vele, hogy bemásoltuk a session-ünk végére illetve megadtuk az img.eval-nak a mi session-ünket.

Ha a fenti lépésekkel elvégeztük akkor már csak a futtatás van hátra. A felismerni kívánt kép az előző feladatban használt kép lesz. A program futása valamennyivel több mint a Soft verziójé de 10 percen belül ez is végzet. De lásuk is mit kaptunk:

The screenshot shows a Visual Studio Code interface. On the left is the code editor with a file named 'DeepMnist.py'. The code is a TensorFlow script for training a neural network on the MNIST dataset. It includes imports, variable definitions, and a main loop for training. On the right is a separate window titled 'Figure 1' showing a 28x28 pixel grayscale image of the digit '8'. The image is centered in a white square. Below the image, there are some status icons and coordinates (x=11.6061, y=26.6515) and [0]. The bottom status bar of the code editor shows 'Python 3.7.0 32-bit' and other system information.

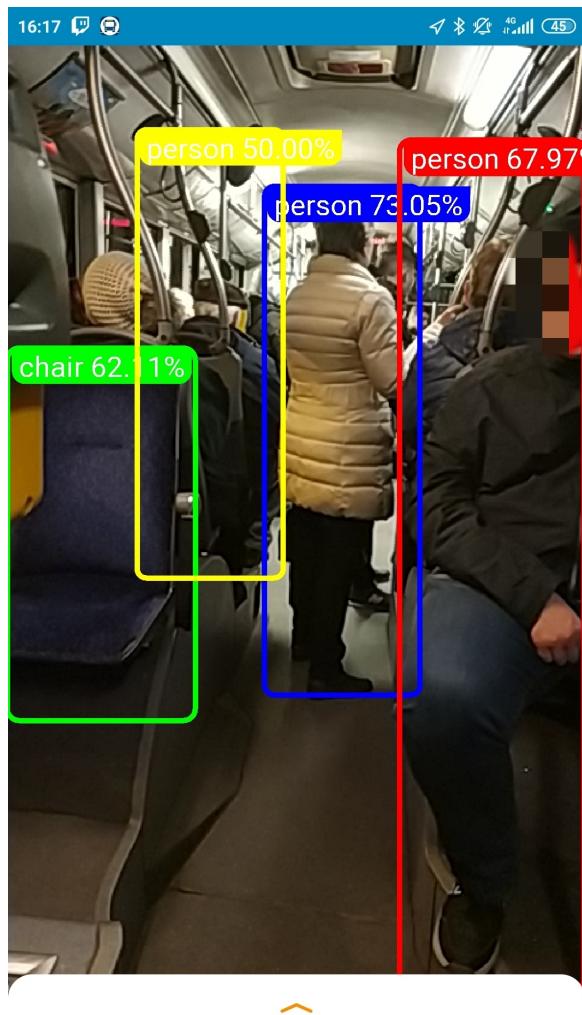
Mint láthatjuk a program megkapta a nyolcasunkat majd a felugró ablakot bezárva láthatjuk, hogy a program a képet nyolcasnak ismerte fel, úgy ahogyan azt el is vártuk tőle.

This screenshot is similar to the previous one, showing the 'DeepMnist.py' script in Visual Studio Code. The terminal output at the bottom of the code editor shows the digit '8' being identified. The text reads: '-- A saját kezi 8-asom felismerte, mutatom a szamot, a továbblepeshez csukd be az ablakat' (-- My handwritten 8 was recognized, show me the number, press the window to continue). The rest of the terminal output is identical to the first screenshot.

19.3. Android telefonra a TF objektum detektálója

Ebben a feladatban a TensorFlow objektum detektálóját kellet letöltenünk és kipróbálnunk. Ezt kétféle képen lehetet megcsinálni, az egyik hogy letültjük az apk-t és egyszerűen telepítjük a telefonra vagy leklónozzuk a tensorflow github repóját és android stúdióba betöltsük magunknak hozzuk létre az apk-t.

Nézzük is meg az appot működése közben:





19.4. Minecraft Malmö

Ebben a feladatban a Microsoft Minecraft Malmö projektjével kellet foglalkoznunk méghozzá az ágens programozásával. Prog 1-en már foglalkoztunk a kérdésel és létre is hoztunk egy ágenst ami a körülötte lévő blokkokat figyelve fedezte fel a világot. Most viszont egy kicsit más irányból közelítetük meg a dolgokat.

Az ágens a világ legenerálása után elkezd hatadni előre a világban, majd amikor akadájba ütközik nem megvizsgálni fogja a maga körül lévő blokkokat hogy merre és hogyna tud majd továbbhaladni hanem random fog választani egy cselekvéssorozatot tartalmazó listból. Ez azért érdekes mivel annak ellenére, hogy az ágens tudná, hogy mi van körülötte a perceken keresztül tartó futtatáson keresztül nem halt meg sőt olyan helyzetbe se került amiből aránylag rövid idő alatt ki ne jött volna. De nézünk bele a kódba egy kicsit.

A könyv elsö fejezetében az ágens programozásának alapjait már kifejtetem szóval ezekre most nem fogok kitérni. Elöször is nézük meg a listánkat:

```

"turn -0.5; wait 1; turn 0; move 1; wait 2",
"move 0; attack 1; wait 5; pitch 0.5; wait 1; pitch 0; attack 1; wait ←
    5; pitch -0.5; wait 1; pitch 0; attack 0; move 1; wait 2",
"move 0; pitch 1; wait 2; pitch 0; use 1; jump 1; wait 6; use 0; jump ←
    0; pitch -1; wait 1; pitch 0; wait 2; move 1; wait 2"
]

```

Mint láthatjuk az ágensünk minden öt lépéssorozatot fog használni, mint láthatjuk az első három sorozat csak mozgásra vonatkozó parancsokat tartalmaz míga az utolsó kető már blokkok kiütését és építést is tartalmaz.

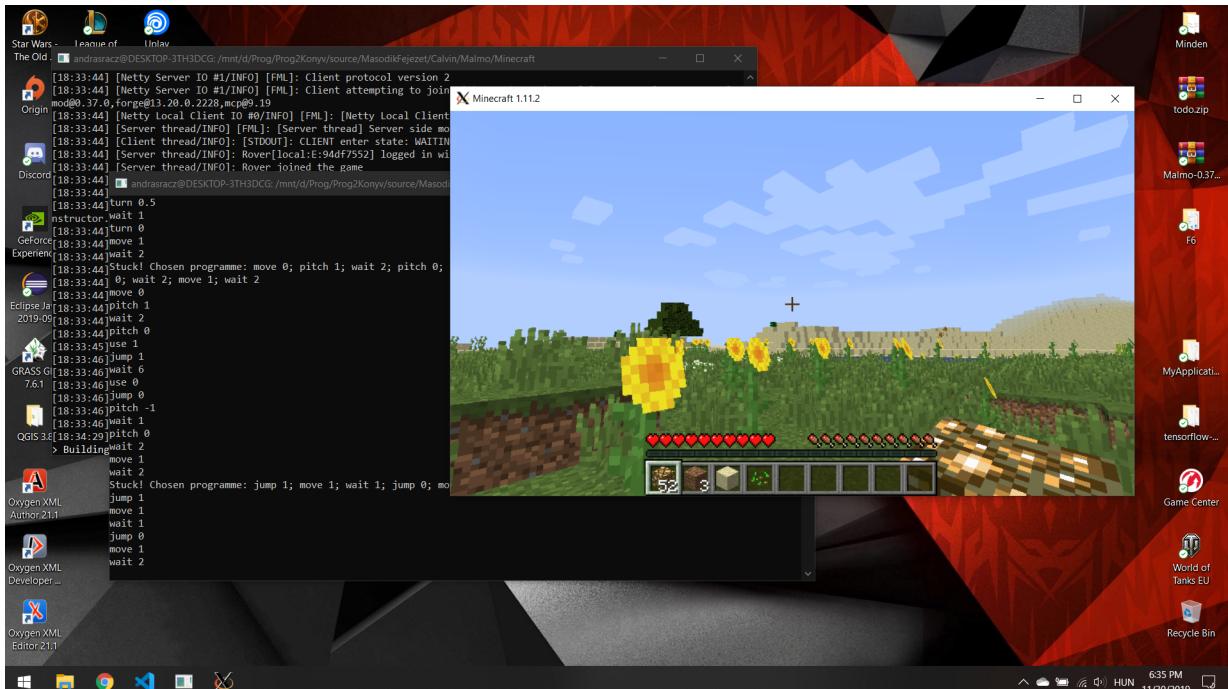
```

if currentSequence == "" and currentSpeed < 50 and waitCycles == 0:
    currentSequence = random.choice(commandSequences)
    print("Stuck! Chosen programme: " + currentSequence)

```

A fent látható rész a program egyik legfontosab része ugyanis ez fog reagálni az ágens beragadására, még hozzá ugy, hogy megnézi azt, hogy van-e vérehajtandó parancsunk. Ezt ugy teszük meg, hogy megvizsgáljuk, hogy a currentSequence üres-e , figyeljük, hogy csökkent-e az ágens sebessége illetve, hogy várakozás ciklusba vagyunk-e. Ha ez a három feltétel teljesül akkor az ágens beragadt így a commandSequences-ből random választunk egy tevékenységsorozatot és odaadjuk a currentSequence-nek.

És nézük is meg élesben:



Mint láthatjuk a terminálon láthatjuk hogy minden egyes beragadás után random választot egy sorozatot majd végrehajtotta azt. Ez nem a legszakszerűbb ágens amit írni lehet ugyanakkor láthatjuk, hogy a random tevékenységek milyen gyakorisággal lesznek hasznosak illetve, hogy medig lehet túlélni a játékot ezen modszer alkalmazásával.

20. fejezet

Helló, Berners-Lee!

20.1. Python élménybeszámoló

A Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba című könyv A python nyelv bemutatása című fejezete bevezet minket a python nyelv világába, elmondja a legfőbb tulajdonságait, szót ejt a python programozási nyelv szinataxisáról illetve alapszinten bemutatja a legfontosabb parancsokat azoknak a használatát.

Mielőt belemélyednénk a python programozásban először ésszerü a szintaxisával megismeredni. A pythonak viszonylag egyszerű, könnyen átlátható, rendezet szintaxissal rendelkezik. Legfőbb tulajdonsága, hogy behuzás alapú, a programban szereplő áltásokat azonos behúzással tudjuk csoportositani. Ugyanakkor fontos jelezője még a nyelvnek, hogy minden utasítás a sor végéig tart így nem kell minden kiírni a jól megszokott ;-ket. A python kód minden egyes sorát úgynevezet tokenekre bonthatunk, amelyk között üres karakterek lehetnek. A tokenek lehetnek: azonosító, kulcsszó, operátor, delimiter, literál. A pythonban a # karaktert használjuk egy sor kommentelésére.

Pythonban minden adatot objektumok reprezentálnak. Azt, hogy egy adaton milyen műveletek végezhetők el az objektum típusa határozza meg. Fontos megemlíteni, hogy pythonban nem szükséges a változó megadásakor típust hozzárendelni, mivel futatási időben ezt a rendszer atomatikusan kitalálja és hozzárendeli. Egy változó lehet szám, azon belül is lehetnek egészek, lebegőpontosak vagy complex számok, logikai változó, szöveg. Fontos még megemlíteni a tuplest, a listát illetve a szótárat. A tuples objektumok gyűjteménye amiben az objektumok típusa lehet eltérő is. A lista különböző típusú elemek rendezet szekvenciája. Ennek jelezője, hogy dinamikus és az elemeit indexekkel azonosítjuk. A szótár pedig kulcsokkal azonosított elemek halmaza.

A fejezet következő részében olvashatunk néhány fontosabbat a listákra vonatkozó parancsok közül mint például count, append, extend, insert, remove, pop, len, del, reverse és a sort. A count(a) visszaadja a-nak előfordulásainak számát. Az append(a) hozzáfűzi a-t a lista végéhez. Az extend(l)-el l-t hozzáfűzhetjük a lista végéhez. Az insert hasonló az appendhez csak az adott elemet akárhova hozzáadhatjuk a listához. Del-el törölni lehet, len-el pedig a lista elemszámát kapjuk meg. A sort rendezzi a listát alapértelmezetten növekvő sorrendbe.

Miután a típusokkal és a hozájuk tartozó fontosabb utasításokkal végrzünk rátérhetünk a nyelv eszközeire is. Elsőnek megemlítiük a print utasítást ami változók vagy stringek kiiratására használhatunk. Majd rátérünk a már más programozási nyelvekből jól ismert elágazásokra és a for illetve while ciklusra

```
# elgazás
if szam :
    print("Kisebb mint 0")
elif szam > 0 :
    print("Nagyobb mint 0")
else :
    print("Pontosan 0")

# for ciklus
for i in range(0,11):
    print("kutya")

# while ciklus
i=0
while(i<3):
    print(i)
    i=i+1
```

Következőnek a függvényekkel foglalkozunk. Pythonban fügvényt a def szóval definiálunk. A függvényekre értékekkel is gondolhatunk mivel azok továbbadhatóak más függvényeknek, illetve objektumkonstruktornak is. A függvények rendelkeznek paraméterekkel amelyek érték szerint adódnak át, illetve beszélhetünk ugyanevezett mutable típuskról pl. listák szótárak. A függvény hívásánál megtudjuk adni a szokot módon úgy, hogy a paramétereket a függvény definícióban meghatározott sorrendben kérjük be. Emellett lehetőségünk van arra hogy konkrét argumentumoknak híváskor adjunk értéket. A függvények általába egy visszatérési értékkel rendelkeznek de visszatérhetnek tuplesel is.

```
# Egy egyszerű függvény egy szám négyzetére
def Negyzet(szam):
    negyzet=0
    negyzet=szam*szam
    return szam
```

A pythonos fejezet végén elérünk a osztályokhoz és objektumokhoz. A python támogatja a klasszikus objektumorientált eljárásokat, definiálhatunk osztályokat, amelynek példányai az objektumok. Az osztályoknak vannak attribútumai objektumok, illetve függvények. Emellett egy osztály örököli más osztálytól is.

```
# Osztály definiálása
class osztalynev(ososztalyok):
    osztalytorzs
```

Az osztály más osztályok opciós listái vesszővel elválasztva. Az osztályoknak lehet egy speciális,

konstruktur tulajdonságú metodusa is az `__init__`. Ennek első paramétere a `self`, vagyis a létrehozandó objektum maga. Ezen kívül további paraméterek is várhatóak.

```
# konstruktorra példa
class teglalap(object):
    def __init__(self, szelesseg, magassag):
        self.szelesseg = szelesseg
        self.magassag = magassag
```

Bár a python osztályai még rengeteg szolgáltatást nyújt a könyv nem tér ki rájuk mivel csak egy alapszintű betekintést kíván nyújtani a Python programozási nyelv világába.

Összeségében a könyv ezen fejezeta a Pythonról jól összeszedet, logikusan felépitve mutatja be a Python nyelv alapjait és szépségeit.

20.2. C++ és Java összehasonlítása

Mint ahogyan a címből is kiderülhetett a Java és a C++ nyelveket fogjuk összehasonlítani a Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven és a Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II. könyvek alapján.

Alapvetőleg a C++ és Java programozási nyelvek egyaránt objektumorientáltak, a programok osztályokból és az ezekből létrehozott objektumokból áll. A két nyelv között talán a legnagyobb különbség az, hogy a C++-al ellentében, ami egy natív nyelv, addig a Java kódot fordítva egy Javabyte kódott kapunk amit majd a Java viruális gép fog értelmezni. Ennek köszönhetően a Java, a C++-al ellentétben, ténylegesen egy cross platform nyelv.

Mint már feljebb is említettem a Java és a C programozási nyelvek nagyon hasonlóak és ez a szintaxisukban is meglátszik. Itt egy egyszerű példa erre a hasonlóságra:

```
//C++
class MyClass
{
    public:
        static doStuff();
};

MyClass::doStuff();
```

```
//Java
class MyClass
{
    public static doStuff()
    {
        ...
    }
}

MyClass.doStuff();
```

A Java nyelvnek egyik nagy előnye a C++-al szemben (többekközt ezért alkalmaza egyre több cég manapság), hogy a Java rengeteg funkciót tartalmaz mint például hálózati és 3D megjelenítésre egyaránt ad eszközöket.

Már a szintaxisoknál felhozott példámból látható volt, hogy a Java osztály koncepciója egyezik a C++-éval visszont ennek ellenére működési különbözők fellelhetők. Egyik ilyen különbség, hogy javában nem áll módunkba osztályon kívül definiálni egyes ostálymetódusokat. A java esetében nem tudjuk meghatározni, hogy mikor fog a memoria felszabadulni mivel a dekonstruktort írni. Ezzel elentében a C++-ban mi felelünk a memóriaért és ebbe beleértendő annak felszabadítása is.

A javában lehetőségünk van arra, hogy programunk egyes részeit csomagokba szervezzük amivel lehetőségeink nyílik a statikus csomagimportálásra java5-től. Ez lehetőséget ad arra, hogy egy statikus változónak nem kell megnevezni az osztályát.

Ebben a hasábban ejtsünk pár szót a stringkről mivel ezek a javában viszonylag jól működnek. A sztringre mint típusra sok megszorítás vonatkozik, hogy szálbiztosá tegye annak használatát. Amikor két sztring megegyezik azok hashCode-ja is megegyezik. Ezeket a hashCode metódussal le tudjuk kérdezni. minden objetumokból sztringet készíthetünk pl a toString metódussal. Javában lehetőségünk van az alap sztring-műveletek mellett például mintailestésre is a matches függvényel.

A komplexebb matematikai műveletekhez a javában a StrictMath és a Math osztályokra van szükség ami a C++-ban a cmath könyvtárnak felel meg.

Míg a C++-ban lehetőségünk van virtuális függvények létrehozására javában csak ezeket használatjuk mivel csak virtuális függvényeket írhatunk. Ebből az következik, hogy egy osztály felülírhatja a szülő összes metodusát. Javában metódust, a C++-al elentében, csak az osztályon belül lehet megadni.

IV. rész

Irodalomjegyzék

20.3. Általános

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

20.4. C

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

20.5. C++

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

20.6. 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/DEAHCackers> (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.