

Univerzális programozás

Így neveld a programozód!

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

Copyright © 2019 Kiss Máté

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

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

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

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

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

COLLABORATORS

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

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2019-02-27	Helló, Turing! csomag kész.	kissmate
0.0.6	2019-03-13	Helló, Chomsky! csomag kész.	kissmate
0.0.7	2019-03-20	Helló, Caesar! csomag kész.	kissmate
0.0.8	2019-03-27	Helló, Mandelbrot! csomag kész.	kissmate

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-04-03	Helló, Welch! csomag kész.	kissmate
0.1.0	2019-04-10	Helló, Conway! csomag kész.	kissmate
0.1.1	2019-04-24	Helló, Schwarzenegger! csomag kész.	kissmate
0.1.2	2019-05-01	Helló, Chaitin! csomag kész.	kissmate

DRAFT

Ajánlás

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

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

DRAFT

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	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	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	13
2.6. Helló, Google!	13
2.7. 100 éves a Brun téTEL	15
2.8. A Monty Hall probléma	16
3. Helló, Chomsky!	19
3.1. Decimálisból unárisba átváltó Turing gép	19
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	20
3.3. Hivatalos nyelv	21
3.4. Saját lexikális elemző	22
3.5. l33t.l	23
3.6. A források olvasása	23
3.7. Logikus	25
3.8. Deklaráció	25

4. Helló, Caesar!	28
4.1. int *** háromszögmátrix	28
4.2. C EXOR titkosító	29
4.3. Java EXOR titkosító	31
4.4. C EXOR törő	32
4.5. Neurális OR, AND és EXOR kapu	34
4.6. Hiba-visszaterjesztéses perceptron	37
5. Helló, Mandelbrot!	39
5.1. A Mandelbrot halmaz	39
5.2. A Mandelbrot halmaz a std::complex osztállyal	41
5.3. Biomorfok	43
5.4. A Mandelbrot halmaz CUDA megvalósítása	46
5.5. Mandelbrot nagyító és utazó C++ nyelven	49
5.6. Mandelbrot nagyító és utazó Java nyelven	51
6. Helló, Welch!	54
6.1. Első osztályom	54
6.2. LZW	56
6.3. Fabejárás	61
6.4. Tag a gyökér	62
6.5. Mutató a gyökér	64
6.6. Mozgató szemantika	65
7. Helló, Conway!	66
7.1. Hangyszimulációk	66
7.2. Java életjáték	69
7.3. Qt C++ életjáték	76
7.4. BrainB Benchmark	76
8. Helló, Schwarzenegger!	80
8.1. Szoftmax Py MNIST	80
8.2. Mély MNIST	83
8.3. Minecraft-MALMÖ	83

9. Helló, Chaitin!	84
9.1. Iteratív és rekurzív faktoriális Lisp-ben	84
9.2. Gimp Scheme Script-fu: króm effekt	85
9.3. Gimp Scheme Script-fu: név mandala	85
10. Helló, Gutenberg!	86
10.1. Programozási alapfogalmak	86
10.2. Programozás bevezetés	88
10.3. Programozás	89
III. Második felvonás	91
11. Helló, Berners-Lee!	93
11.1. Bevezetés a mobilprogramozásba.	93
11.2. Java és C++	95
12. Helló, Arroway!	99
12.1. OO szemlélet	99
12.2. Homokozó	101
12.3. Gagyí	102
12.4. Yoda	104
12.5. Kódolás from scratch	105
13. Helló, Liskov!	108
13.1. Liskov helyettesítés sértése	108
13.2. Szülő-gyerek	110
13.3. Anti OO	112
13.4. Hello, Android!	113
13.5. Ciklomatikus komplexitás	115
14. Helló, Mandelbrot!	116
14.1. Reverse engineering UML osztálydiagram	116
14.2. Forward engineering UML osztálydiagram	117
14.3. Egy esettan	118
14.4. BPMN	122
14.5. TeX UML	123

15. Helló, Chomsky!	124
15.1. Encoding	124
15.2. Full screen	125
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	128
15.4. Paszigráfia Rapszódia LuaLaTeX vizualizáció	128
15.5. Perceptron osztály	129
16. Helló, Stroustrup!	131
16.1. JDK osztályok	131
16.2. Változó argumentumszámú ctor	133
16.3. Hibásan implementált RSA törése és összefoglaló	134
17. Helló, Gödel!	138
17.1. STL map érték szerinti rendezése	138
17.2. GIMP Scheme hack	139
17.3. Alternatív Tabella rendezése	143
17.4. Gengszterek	147
18. Helló,!	149
18.1. SamuCam	149
18.2. BrainB	152
18.3. FOOCWC Boost ASIO hálózatkezelése	153
18.4. OSM térképre rajzolása	154
19. Helló, Lauda!	157
19.1. Android Játék	157
19.2. Port scan	161
19.3. Junit teszt	162
19.4. AOP	163
20. Helló, Calvin!	165
20.1. MNIST	165
20.2. Android telefonra a TF objektum detektálója	167
20.3. Minecraft MALMO-s példa	168
20.4. Deep MNIST	171

IV. Irodalomjegyzék**172**

20.5. Általános	173
20.6. C	173
20.7. C++	173
20.8. Lisp	173

DRAFT

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- C kapcsán a [**KERNIGHANRITCHIE**] könyv adott részei.
- C++ kapcsán a [**BMECPP**] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a [The GNU C Reference Manual](#), mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipete! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [**BMECPP**] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

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

- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

A könyvben szereplő összes forráskód megtalálható csomagonként összegyűjtve az alábbi linken: <https://github.com/kissmate3/Prog1>

A végtelen ciklus egy olyan ciklus, amelyben a feltétel állandóan adott(igaz), ezért a ciklus nem lép ki, hanem újra és újra lefut. Például `for(;;)`

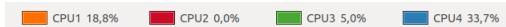
Vannak ciklusok amik a szálakat 100%-ban vagy 0%-ban dolgoztatják.

0%-ban dolgoztat:

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

int main(){
    while(1) {
        sleep(100);
    }
}
```

Magyarázat: Az unistd header tartalmazza a `sleep()` függvényt. Ezért kell include-olni az `stdio.h` header (standart input/output) mellett. Az `int main()` a fő függvényünk. A `while()` pedig a ciklus. A `()` belülre kell írnunk a feltételt. Amíg ez igaz, a ciklus újra és újra lefut. A példánkban a ciklusban az 1 szám szerepel. Ez az érték mindenkorán igaz, tehát a ciklus állandóan újraindul amíg ki nem lőjjük. A `sleep(100)` függvény pedig azért kell, mivel ez altatja a processzor folyamat szálát. A függvényben megadott érték jelenti azt, hogy hány másodpercig altatja a processzort.



100%-ban dolgoztat egy szálat:

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

Magyarázat: Az előző példától nem sokban tér el. Az include-k és a ciklus magyarázata megegyezik, az előző példáéval. Itt annyi a különbség, hogy nincs benne a sleep() függvény, azaz a szál nincs altatva. Így a végletes ciklus 100%-ban dolgoztat 1 szálat.



100%-ban dolgoztat minden szálat:

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

Magyarázat: A programunk, az előzőhez egy openmp-vel bővült. Ezzel az include-val belépünk, a párhuzamos programozás küszöbérre. #pragma omp parallel sor adja azt az utasítást a gépnek, hogy a feladat az összes szálon fusson. (A fordításnál -fopenmp kapcsolóval kell bővítenünk a parancsot.)



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

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

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

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

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

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

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

akár önmagára

```
T100(T100)  
false
```

ezt a kimenetet adja.

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

```
Program T1000  
{  
  
    boolean Lefagy(Program P)  
    {  
        if(P-ben van végtelen ciklus)  
            return true;  
        else  
            return false;  
    }  
  
    boolean Lefagy2(Program P)  
    {  
        if(Lefagy(P))  
            return true;  
        else  
            for(;;);  
    }  
  
    main(Input Q)  
    {  
        Lefagy2(Q)  
    }  
  
}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok: Ha úgy vesszük, hogy a T100 és T1000 létező program és T1000 ben meghívjuk saját magát. A T100 alapján ha a programunkba van végtelen ciklus, akkor igaz értéket ad a Lefagy program a Lefagy2 programnak, így tehát az is igaz értéket fog adni, viszont ha a Lefagy false értéket ad vissza akkor a Lefagy2 belém egy végtelen ciklusban, tehát a program le fog fagyni. Tehát olyan program mint a T100 nem működik mivel, ha egy olyan program érkezik bele amiben van végtelen ciklus, akkor a program beáll mert a ciklus nem áll meg.

2.3. Változók értékének felcserélése

A feladat két változó értékének felcserélése. Például $a=1$, $b=2$, ebből lesz a megoldás, hogy $a=2$, $b=1$. Napjainkba a számítógép fejlettsége és gyorsasága miatt, már egyszerűen megcsinálhatjuk egy segédváltóval vagy exort-tal, de régen nagyon sokat számított az erőforrások jó felhasználása, elosztása. Ezért ezek a megoldásoknál sokkal könnyebb volt a számítógépeknek számolni, ha különbséggel vagy szorzással cserélük fel a változókat. Az utóbbi kettőt nézzük most meg:

Változócsere különbséggel:

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

int main()
{
    int a=1;
    int b=2;

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

    a=a-b;
    b=a+b;
    a=b-a;
    printf("%d %d\n", a, b);
}
```

Magyarázat: A fejlécet már ismerjük az előző feladatból. A printf() függvény a kiíratáshoz kell majd nekünk. az első argumentum a kíratás formátuma, a többi pedig a változók kiíratása. A „%d” azt jelenti, hogy egy egész típusú változót fogunk kiíratni, még a „\n” a sortörést jelenti. Maga a feladat egyszerű matematika. Legegyszerűbben a példával lehet megérteni.

$a=1, b=2$

$a=1-2=-1$, „a” értéke -1 lesz.

$b=-1+2=1$, „b” értéke 1 lesz, ami az „a” értéke volt.

$a=1-(-1)=2$ az „a” értéke 2, ami a „b” értéke volt

Kész is a cserénk.

Változócsere szorzattal:

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

int main()
{
    int a=1;
    int b=2;

    printf("%s\n%d %d\n", "szorzassal:", a, b);
```

```
a=a*b;  
b=a/b;  
a=a/b;  
  
    printf ("%d %d\n", a, b);  
}
```

Magyarázat: A megoldás itt annyiban különbözik, hogy nem „+” és „–”, -t használunk hanem „*” és „/” -t.
Példa:

a=1, b=2

a=1*2=2 „a” értéke 2 lesz.

b=-2/2=1 „b” értéke 1 lesz, ami az „a” értéke volt.

a=2/1=2 az „a” értéke 2, ami a „b” értéke volt.

Kész is a cserénk.

2.4. Labdapattogás

```
#include <stdio.h>  
#include <curses.h>  
#include <unistd.h>  
  
int  
main ( void )  
{  
    WINDOW *ablak;  
    ablak = initscr ();  
  
    int x = 0;  
    int y = 0;  
  
    int xnov = 1;  
    int ynov = 1;  
  
    int mx;  
    int my;  
  
    for ( ; ; ) {  
  
        getmaxyx ( ablak, my , mx );  
  
        mvprintw ( y, x, "O" );  
  
        refresh ();  
        usleep ( 100000 );  
  
        x = x + xnov;
```

```
    y = y + ynov;

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

}

return 0;
}
```

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

Magyarázat: Az új dolog ami a fejlécnél feltűnik az a curses.h header. Ez képernyő kezelési függvényeket tartalmaz, és a program megjelenítéséhez szükségünk van rá.

A következő részlet:

```
WINDOW *ablak;
ablak = initscr();
```

Így formázzuk meg a kimenetet. Az initscr() függvény curses módba lépteti a terminált.

A deklarált x és y -on lesz a kezdő értékünk. Az xnov és ynov pedig a lépésközöt mutatja. (lépésenként a koordináta rendszeren xnov, ynov-al való elmozdulást). Az mx és my lesznek a határértékek, hogy a program csak az ablakon belül mozogjon.

A végtelen ciklus következetében, a labda addif pattog, amíg ki nem lőjük a programot. A ciklusban az első függvény a getmaxyx(). Ez határozza meg, hogy mekkora az ablakunk mérete. refresh() függvénytel frissítjük az ablakot. Közöttük a mvprintw() függvény az x és y tengelyen megrazolja a „ „ között lévő szöveget, számot vagy karaktert, esetünkben az O-t. Az usleep függvény azt szabályozza mennyi ideig altassa a ciklust még újra indul, azaz milyen gyorsan pattogjon a labda.

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

Megnöveljük az értékeket, minden ciklus lefutásnál (mozog a "labda").

A következő négy if-el pedig azt vizsgáljuk, hogy a labda az ablak szélén van-e, ha igen akkor -1 -el szorozzuk, ezáltal a labda irányt változtat. A fordításnál -lncourses kapcsolót kell használnunk.

Nézzük ugyan ezt a feladatot "if" nélkül:

Forrás:https://progpater.blog.hu/2011/02/13/megtalaltam_neo_t

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

int
main (void)
{
    int xj = 0, xk = 0, yj = 0, yk = 0;
    int mx = 80 * 2, my = 24 * 2;

    WINDOW *ablak;
    ablak = initscr ();
    noecho ();
    cbreak ();
    nodelay (ablak, true);

    for (;;)
    {
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;

        yj = (yj - 1) % my;
        yk = (yk + 1) % my;

        clear ();

        mvprintw (0, 0,
                  " " ←
                  -----);
        mvprintw (24, 0,
                  " " ←
                  -----);
        mvprintw (abs ((yj + (my - yk)) / 2),
                  abs ((xj + (mx - xk)) / 2), "X");

        refresh ();
        usleep (150000);
    }
    return 0;
}
```

Magyarázat: A prgoramunk ugyan azt csinálja mint az "if"-es változata. Csak ugye most logikai kifejezés, utasítás nélkül. A megoldáshoz szükségünk van matematikai számításokra, ehez deklarálunk egész tipusú változókat. A számításokat egy végtelen ciklusban számoljuk és mvprinw-vel íratjuk ki a képernyőre. A clear()-el minden egyes számítás előtt letisztítjuk az ablakot. az eslő kettő mvprintw-vel a felső és alsó

határokat rajzoljuk ki. A 3 al pedig a "Labdát". Az Usleep függvény itt is a pattogás sebeségét határozza meg.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Szóhossz:

```
#include <stdio.h>

int main()
{
    int a=1;
    int bit=0;
    do
        bit++;
    while(a<<=1);
    printf("%d %s\n",bit,"bites a szohossz a gepon");
}
```

Ez a program a géünk szó hosszát fogja kiírni, azaz az int méretét. A feladatot a BogoMIPS ben használt while ciklus feltétellel írjuk meg (A BogoMIPS a processzorunk sebeségét lemérő program amit Linus Torvalds írt meg).

A main függvényben az első sor az int a=1. Itt deklaráljuk a változót, amivel vizsgáljuk meg a géünk szóhosszát(Az int méretét). A "bit" változó fogja a lépéseket számlálni. A programot dowhile ciklus-sal(házteljesítő) futtatjuk, mivel a sima while nem számítaná bele az első lépést, tehát ha a géünk 32 bites, a program 31 bitet írna. A ciklus addig fut amíg az "a" nem lesz egyenlő nullával. És akkor mi is az a bitshift operátor. Ugye vesszük az 1-et, a=1. ennek a Bináris kódja a 0001, a bitshift operátor egy 0-val eltolja, azaz 0010 kapjuk, ez a 2 szám, a count növekedik tehát az értéke 1 lesz. A ciklus újra lefut és eltolja még egyszer a számot egy 0-val, így 0100 kapunk ami a négy. Ez addig fut, még a géünk szó hosszán (az int méretén) kívül nem tolja az 1-est. Ekkor az a értkében csak 0 fog szerepelni, azaz az "a" értéke 0 lesz, a while ciklus befejeződik, és kiíratjuk hányat lépett a ciklus, és ez a szám adja meg, hogy hány bites a szóhossz.

```
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/2.csomag$ gcc szo.c -o szoh
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/2.csomag$ ./szoh
32 bites a szohossz a gepon
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/2.csomag$
```

2.6. Helló, Google!

A PageRank egy keresőmotor amit a Google használ. A programot két fiatal írta meg 1998-ban. Nevét az egyik kitalálója után kapta.

A következőben, egy 4 lapból álló PageRank-at fogunk megnézni. A lapok PageRank-ét az alapján nézzük, hogy hány oldal osztotta meg a saját honlapján az oldal hiperlinkjét.

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

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

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

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

    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

    return sqrt(osszeg);
}

int
main (void)
{
    double L[4][4] = {
        {0.0, 0.0, 1.0 / 3.0, 0.0},
        {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
        {0.0, 1.0 / 2.0, 0.0, 0.0},
        {0.0, 0.0, 1.0 / 3.0, 0.0}
    };

    double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
    double PRv[4] = { 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0 };

    int i, j;

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

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

}

kiir (PR, 4);

return 0;
}
```

Forrás:https://progpater.blog.hu/2011/02/13/bearazzuk_a_masodik_labot

Kezdjük az új headerrel, ez a math.h. Ez tartalmazza a matematikai számításokhoz szükséges függvényeket. A main() fügvénnyben először is létrehozunk egy mátrixot, ami a lapok összeköttetését adja meg. Ha az érték 0 akkor a lap nincs összekötve az adott lappal és persze önmagával sincs. Ahol 1/2 vagy 1/3 az érték az azt jelzi, hogy az oldal hányszorosan van összekötve, például az 1/2: Az oldal 2 oldallal van összekötve és abban az egyik kapcsolatot jelzi (az 1).

A PR tömb fogja a PageRank értéket tárolni. A PRv tömb pedig a mátrixal való számításokhoz kell. A következő lépés egy végtelen ciklus. Ez majd a számítások végén a break parancsal lép ki, ha a megadott feltétel teljesül. A forciklusban van maga a PageRank számítása ami majd a tavolság függvényt is meghívja, ami egy részszámolást tartalmaz. A végtelen cikluson belül lévő ciklusok azért 4-ig mennek mert 4 oldalt nézünk. A ciklusbol való kilépés a "break" parancsal történik majd ha a tavolság függvényben kapott eredmény kisebb mint 0.00000001. A végén a kiir függvény megkapja a PR értékeit és az oldalak számát és kiíratja azokat.

2.7. 100 éves a Brun téTEL

A tételet Viggo Brun bizonyította 1919-ben. Ezért is neveztek el róla. A téTEL kimondja hogy az ikerprímek reciprokösszege a Brun konstanthoz konvergál, ami egy véges érték.

Brun téTEL R szimulációban:

```
library(matlab)

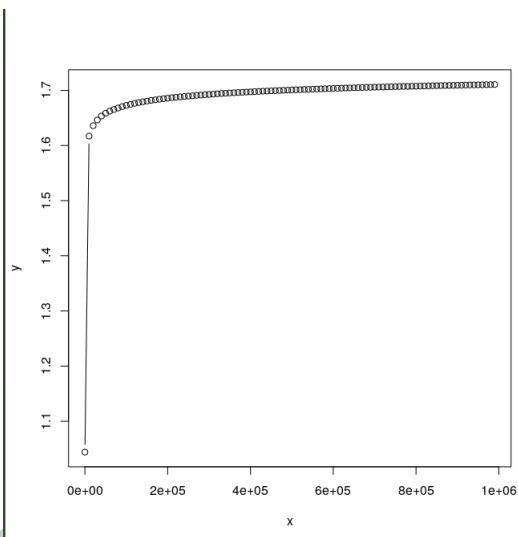
stp <- function(x){

  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")
```

A számoláshoz először is kell egy matlab könyvtár. A program fő része az stp függvény. egy a függvény megkapja x-et. X egy szam lesz ami megmondja meddig kell a prímeket számolni. Ehez a primes függvényt használjuk. primes(x) kiírja x-ig a prímeket. A diff vektorban eltároljuk a primes vektorban tárolt egymás melletti prímek különbségét. A számítást úgy végezzük, hogy a 2 prímtől indulva kivonjuk a prímből az előtte lévő prímet. Az idx el vizsgaljuk meg, hogy mely prímek különbsége 2 és ezek hol vannak (a helyük a which függvény adja meg). A t1primes vektorban elhelyezzük ezeket a prímeket. A t2primes vektorba pedig ami ezeknél a primknél kettővel nagyobb (azaz ikerprímek). rt1plus2 vektorban végezzük a reciklopépzést és a pár reciprokát összeadjuk. A returnban pedig a sum függvényel vissza adjuk ezek summázott összegét. Végezetül a plot függvénnnyel lerajzoljuk grafikusan.

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



2.8. A Monty Hall probléma

Tutorált: Földesi Zoltán

Ez egy valószínűségi paradoxon. A kérdés egy vetélkedő játékból indul. Van 3 ajtó és az egyik mögött egy értékes nyeremény van, a másik kettő mögött semmi. A versenyzőnek a 3 ajtó közül választania kell egyet. Miután választott, a műsorvezető kinyit egy ajtót, ami mögött nincs a nyeremény. Felteszi a kérdést, hogy akarunk-e változtatni a választásunkon. Itt jön a felvetés, hogy megéri-e változtatni, vagy nem.

Megoldás: Első ránézésre mi is, és szinte mindenki azt mondaná, hogy nem számít, hogy vált-e mert 50-50% az esélye, hogy melyik ajtó mögött van a nyeremény. Mivel már nem 3 hanem 2 ajtó közül lehet választani, így már figyelembe se veszik azt a harmadik ajtót. De a megoldás az, hogy igen, nagyobb az esélyünk akkor ha az előző döntésünket megváltoztatjuk és a másik ajtót választjuk.

Magyarázat: Kezdetben 3 ajtóból 1 ajtót kell választanunk, azaz 1/3 az esélye, hogy eltaláljuk a jó megoldást és 2/3 hogy nem. Ezek után a műsorvezető kinyit egy ajtót ami mögött nincs a nyeremény. Ez a kezdeti valószínűségen nem változtat, úgyanúgy 1/3 eséllyel választottuk azt az ajtót ami mögött a nyeremény van. Viszont azok az ajtók közül ami mögött nincs semmi, azokból már csak az egyik van csukva. Biztosra tudjuk, hogy a nyeremény a maradék két ajtó közül valamelyik mögött van. Tehát 2/3 az esélye annak, hogy a másik ajtó mögött van a nyeremény.

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MontyHall_R/mh.r

Szimuláció:

```
kiserletek_szama=10000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

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

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

  }else{

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


  }

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


}

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

for (i in 1:kiserletek_szama) {

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


}

valtoztatesnyer = which(kiserlet==valtoztat)

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

Most a kísérletet 10000x fogjuk szimulálni. A kísérlet vektorban 1 és 3 "ajtó" közül választunk 10000x. A replace=T-vel tesszük lehetővé, hogy egy eredmény többször is kijöhessen. A játékos valasztásait a játékos vektornál ugyan így meghatározzuk. A sample() függvényel végezzük aa kiválasztást. A musorvezeto vektort a length függvényel a kísérletek számával tesszük egyenlővé. Következik a for ciklus ami i=1 től a kísérletek számáig fut (10000). A ciklusban egy feltétel vizsgálat következik. az if-fel megvizsgáljuk, hogy a játékos álltal választott ajtó megegyezik e a kísérletben szereplő ajtóval. Ha a feltétel igaz egy mibol vektorba beletesszük azokat az ajtokat amiket a játékos nem választott, az else ágon pedig ha a feltétel nem igaz ,akkor azt az ajtót eltároljuk amit nem a választott és a nyereményt rejtvő ajtót. A musorvezeto vektor-

ban pedig azt az ajtót amit ki fog nyitni. A nem változtat és nyer vektorban azok az esetek vannak amikor a játékos azt az ajtót választotta elsőre ami mögött az ajtó van és nem változtat a döntésén. A változtat vektorban pedig azt mikor megváltozatja a döntését és így nyer ezt egy forciklussal vizsgáljuk. A legvégén kiíratjuk az eredményeket, hogy melyik esetben hányszor nyert.

DRAFT

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

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

Magát a fél fogalmát 1936 -ban Alan Turing alkotta meg. A gép decimális számrendszerből unáris számrendszerbe írja át a számot. Az unáris számrendszer másnéven egyes számrendszer. A lényege, hogy 1 eseket írnunk csak, ha az 1 számot akarjuk unárisba átváltani, az értéke egy, ha a 2-öt akkor az értéke 11, a tíz pedig 1111111111. Az a program c++ ban a következő:

```
#include <iostream>

using namespace std;

int main()
{
    int a;
    int tiz=0, szaz=0;
    cout<<"Decimalis szam:\n";
    cin>>a;
    cout<<"A szam unarisban:\n";
    for (int i=0; i<a; i++) {
        cout<<"1";
        ++tiz;
        if (tiz==10) {cout<<" "; tiz=0;}
        if (szaz==100){cout<<"\n";szaz=0;}
    }
    return 0;
}
```

A kód egyszerű. Bekérünk egy decimális számot "a"-ba, és egy forciklus segítségével addig irunk minden egy 1-est amíg i(ami kezdetben 0 és minden egyel növeljük) kisebb mint a. Én hogy a kimenet szébb legyen 2 változót használtam, 10 db 1 es után egy szóközt teszünk, míg 100 db után egy sortörést.

Magyarázat az Állapotmenet grafikájának:

A gép beolvassa a memorászalag számjegyeit, (Az ábrán a szám a 10) ha elér az "=" ig, az előtte lévő számmal kezd el dolgozni, még az 0 nem lesz. Az első elem egy 0, de mivel a következő nem nulla, hanem

1 ezért ebből kivon 1-et, azaz hátulról a második elemet 0-ra állítja. A kezdő elem ami 0 volt, az pedig 9 lesz, és ebből minden kivonásnál kiírat sorban 1 est, így annyi 1 lesz, mint a decimális szám értéke. (Ha 100 lenne a szám akkor az 100 után 099 lenne aztán 098,097...,089,088 és így megy 000-ig, és a kimeneten 100db 1-es lesz.)

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

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

A generatív nyelvek kidolgozását Noam Chomsky nevéhez fűzzük. A nyelveket osztályba rendezzük. Van-nak erősebb és gyengébb osztályok. Az erősebb osztály képes létrehozni gyengébb osztályt.

Négy darab alapon fekszik a generatív nyelvtan:

- 1.Terminális szimbólumok. Azaz a konstansok.
- 2.Nem terminális jelek. Ezek a változók.
- 3.Kezdőszimbólum. Egy kijelölt szimbólum.
- 4.Helyettesítési szabályok. Ezzel szavakat értelmezzük majd.

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

1.nyelv

S, X, Y
a, b, c

Az S, X, Y lesznek a változóink. Az a,b,c pedig a konstansok

S \rightarrow abc, S \rightarrow aXbc, Xb \rightarrow bX, Xc \rightarrow Ybcc, bY \rightarrow Yb, aY \rightarrow aaX, aY \rightarrow aa

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

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

Azt láthatjuk, hogy egészen addig alkalmazzuk a helyettesítési szabályokat még csak konstansaink lesznek. Azaz mindenkoralsóbb osztályt hozunk létre.

2. Itt a változók az A.B.C és a konstansok a,b,c.

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

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

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

de lehet így is:

A (A->aAB)
aAB (A->aAB)
aaABB (A->aAB)
aaaABBB (A->aC)
aaaaACBBB (CB->bCc)
aaaabCcBB (cB->Bc)
aaaabCBcB (cB->Bc)
aaaabCBBc (CB->bCc)
aaaabbCcBc (cB->Bc)
aaaabbCBcc (CB->bCc)
aaaabbbCccc (C->bc)
aaaabbbbcccc

3.3. Hivatkozási nyelv

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

Ahogy a beszéltyelv, úgy a programozási nyelv is fejlődik. Ennek a bemutatására az alábbi programot fogjuk használni:

```
#include <stdio.h>

int main(){
    for(int i=0;i<1;i++){
        printf("Lefut");
    }
}
```

Itt ami lényeges, nem a kódban lesz, hanem a fordításnál. Megvizsgáljuk, hogy a C89 es nyelvtan és a C99-es szerint hogyan fordítja le a programot a fordító. Ha a C89 es nyelvtannal fordítom: "gcc -std=gnu89 fajlnev.c -o fajlnev". A program hibát fog írni a for ciklusnál. Most ha a fordításnál átírjuk "gcc -std=gnu99 fajlnev.c -o fajlnev"-re (Azaz a fordító a 99 nyelvtan lesz) ,akkor láthatjuk, hogy lemegy a fordítás és a program működik. A kódon belül, a for ciklusban deklaráltuk az int i-t.

magyarázat: Az okot a kódon belül, a for ciklusban kell keresni, ugyanis az "i" -t a forcikluson belül deklaráltuk. A C89 nyelvtanban ez még nem volt megengedett, így a fordító hibát írt, de a C99-ben már igen, ezért nem jelez hibát.

3.4. Saját lexikális elemző

A program a bemeneten megjelenő valós számokat összeszámolja.

A lexikális elemző kódja:

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

A szamok változóval számoljuk hányszor fordul elő szám a bemenetben. A programot a % -jelekkel osztjuk fel részekre. a

```
[0-9]+ {++szamok; }
```

Ez a sor adja azt, hogy 0-9 vagy nagyobb számot talál akkor növelje a "szamok" változót. A printf el pedig csak kiíratjuk hogy hány szám volt a bemenetben (ez az elemzés). A yylex() a lexikális elemző a fordítás a következő:

```
flex program.l
```

ez készít egy "lex.cc.y" fájlt. ezt az alábbi módon futtatjuk.

```
cc lex.yy.c -o program_neve -lfl
```

A futtatáshoz pedig hozzá kell csatolni a vizsgált szöveget.

3.5. I33t.I

Tutor: Földesi Zoltán

Lexelj össze egy I33t ciphert!

```
% {
#include <string.h>
int szamok=0;
%
%"0" {printf("o");}
%"1" {printf("i");}
%"3" {printf("e");}
%"4" {printf("a");}
%"5" {printf("s");}
%"7" {printf("t");}

%"o" {printf("0");}
%"i" {printf("1");}
%"e" {printf("3");}
%"a" {printf("4");}
%"s" {printf("5");}
%"t" {printf("7");}

%%
int
main()
{
    yylex();
    printf("%d szam", szamok);
    return 0;
}
```

Ez a program lefordítja a I33t nyelven írt titkos szöveget vagy a rendes szöveget írja át a I33t nyelvre.

A program működése az előzővel majdnem megegyezik, csak annyiban tér el, hogy valós számok helyett, itt most a megadott számokat keresi a bemenetben és azok a számok helyett a I33t nyelvben való megfelelő betűket írja a helyére. Ha pedig a I33t nyelvre akarjuk fordítani, akkor a betűket vizsgálja és a megfelelő számot írja be.

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, jelkezelo);
```

A példában szereplő kód részlet ellentetje, azaz ha a SIGNT jel kezelése nem lett figyelmen kívül hagyva akkor, a jelkezelő függvény kezelje.

ii.

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

Ez egy forciklus, benne az i értéket 0 állítjuk, és amíg az i értéke kisebb mint 5 addig a ciklus újra és újra lefut. A 3 argumentumban növeljük az i értékét.

iii.

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

A ciklus majdnem ugyan az mint az előző. az eltérés az i érték növelésében van, nem tünik nagy eltérésnek, de fontos. Az előzőbe ++i még itt i++. A különbség az, hogy a ++i nél először növeli az i értékét, aztán az i értékét átadja, még az i++ először átadja az i értékét és aztán növeli az i értékét. Ez a forciklusban úgy van ++i -nél hogy megnöveli egyel az i-t és utánna hajtja végre újra a lefutást. Az i++ nál pedig először végre hajtja aztán növeli az i értékét.

iv.

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

Ez a forciklus egy tombot feltölt az i értékével. a tomb ezek után úgy fog kinézni, hogy tomb[5]={0,1,2,3,4}, mivel i++, ezért előbb átadja az értéket és utánna növeli az i értékét. Bug: A programba máshogys viselkedik , mivel az előző érték mindenkor minden memóriaszemét lesz. A megoldás, hogy a forcikluson belül adjuk hozzá a tombhoz az értéket for(){ ezen belül }.

v.

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

Itt a forciklusunk második argumentumába az i kisebb mint n feltétel mellett van egy másik feltétel. A forciklus csak akkor fut le ha mind a 2 feltétel teljesül. A második feltétel az, hogy az s és a d mutató egyenlő (minden ciklusnál növeljük az értékeket). A feltételt az és operátorral kötjük össze. Bug: A hiba, hogy a második feltétel nem logikai feltétel. Ezt a feltétel is egy if el a forcikluson belül kéne vizsgálnunk.

vi.

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

A printf fügvénnyel kiíratunk valamit. Ebben az esetben két egész tipusú változót. A printf-en belül az f fügvénnyel határozzuk meg a számot. Bug: Rossz a sorrend, ezért hibát kapunk.

vii.

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

A printf fügvénnyel kiíratunk két egész számot, az első számot az f függvény adja (az f függvény az "a"-t kapja meg), míg a másik az a változó értéke.

viii.

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

A printf fügvénnyel kiíratunk két egész számot. Az előzőnél annyival másabb, hogy a függvény az a memória címét kapja meg.

3.7. Logikus

Tutorált: Földesi Zoltán

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

Minden x esetén létezik olyan y ahol x < y és y prím. Ez azt jelenti, hogy a prímek száma végtelen.

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

Minden x esetén létezik olyan y ahol x < y és y prím és az SSy is prím, lefordítva azt jelenti, hogy az ikerprímek száma végtelen.

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

Létezik olyan y, minden x számra, hogy ha x prím akkor x < y, lefordítva a prímek száma véges.

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

Létezik, olyan y ami minden x számra y < x akkor x nem prím, lefordítva ugyan azt jelenti mint az előző, csak tagadással megfogalmazva.

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

A megoldáshoz tudnunk kell mi mit jelent, vannak a logikai összekötőjelek, mint az és=\wedge, \neg=nem, \vee=vagy, \supset=implikáció A kiíratást a \text el végezzük. Vannak kvantorok a "létezik"=\exists és a "minden"=\forall. Az "S" értéknövelés.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató

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

A program:

```
#include <iostream>

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

Mit vezetnek be a programba a következő nevek?

- `int a;`
Egy egész tipusú változót deklarál.
- `int *b = &a;`
Egy int tipusú mutatót deklarál, ami képes egy változó memóriacímét tárolni. "b" mutató "a" ra mutat.
- `int &r = a;`
Egy egész tipusú referenciaját deklarál, ami hasonló a mutatóhoz, de nem ugyan az, a referencia úgymond egy állnév, pontosabban egy már létező változóhoz egy másik név.

Ez egy egész tipusú 5 elemű tömb.

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

Ez egy referenciája a "c" 5 elemű tömbnek (Az összes elemnek).

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

A d tömbben minden egyes tag egy mutató.

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

Az int tipusú változó visszatérési típusát tartalmazó függvény.

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

Egy egész típusra mutató mutatót visszaadó függvény.

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

Egy egész típusú afó és két egész típusú kapó függvényre mutató mutatót visszaadó, egész típust kapó függvény

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

Egy egész típust visszaadó és két egész típust kapó függvényre mutató mutatót visszaadó, egész típust kapó függvényre

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

A következő programban egy alsó háromszögmátrixot hozunk létre.

Forrás:https://gitlab.com/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/-Caesar/tm.c

a kód:

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

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

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

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

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

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

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

Magyarázat: Szokás szerint includoljuk a szükséges include-kat. A fő függvényben az első sora az "int nr=5" ítt adjuk meg, hogy 5 sorunk legyen a kimeneten. A "double **tm", sorral foglalunk le tárhelyet a memoriában. Az első ifben megtaláljuk a malloc függvényt ami dinamikus memória foglaló, ezzel nr számú double ** mutatót foglalunk le, ha null értéket ad vissza az azt jelzi ,hogy nincs elég hely a foglaláshoz. A következő if lefoglalja a mátrix sorait, az első sornak egy double * mutatót foglal le, a másodiknak 2 , a harmadiknak 3 , nr ig. A 3. for ciklussal megadjuk a mátrix elemeit. Az "i" a matrix sorai, a "j" pedig a benne lévő mutatók. a "tm[i][j]=i*(i+1)/2+j; érjük el azt, hogy az elemek minden egyel nőjenek. A 4. for ciklus pedig a kírás. Ezek után már csak annyit csinálunk, hogy a 3 sort megváltoztatjuk, mert így is ki lehet íratni. A legvégén pedig a free()-vel felszabadítjuk a lefoglalt memóriát, ezzel megelőzve a memória folyást.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

A feladat lényege , hogy egy szöveget titkosítsunk Exor-ral(XOR). Az XOR a "kizáró vagy". A szöveget az alábbi módon titkosítjuk: Az eredeti szöveg bájtjaihoz rendelünk titkosító kulcs bájtjokat. Aztán X.cOR-t műveletet végzünk rajta. Az XOR-t művelet úgy működik, hogy ha a bitek azonosak (1,1;0,0) akkor 0 ad vissza értéknek, ha pedig különbözök (1,0;0,1) akkor 1 et ad vissza, és így minden bitpáron elvégezve ezt megkapunk egy titkosított szöveget.

Forrás:https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_parhuzamos_prog_linux/ch05s02.htm

Kód:

```
#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);
    }
}
```

A kód magyarázata: Először is beinclude-oljuk szükséges include-kat. Aztán két állandó változót definiálunk a #define parancsal. Ezeknek az értéke nem változik. Az első állandó a MAX_KULCS az értéke 100. A második pedig a BUFFER_MERET 256, ez nekünk a beolvasásnál fog kelleni. A fő függvényben egy-egy char típusú tömb méreteivé tesszük a 2 állandót. Ezek után 2 változót hozunk be, a kulcs_index, ami a kulcsunk aktuális elemét tárolja, és az olvasott_bajtok ami a beolvasott bajtok összegét tárolja. A kulcs_merete változóban a kulcs méretét adjuk meg a "strlen()" függvény segítségével, amit mi adunk meg egyik argumentumként. Az strncpy függvény pedig a kulcs kezeléséhez kell. Ezután a while ciklusban beolvassuk a buffer tömbe a bemenetet, a while ciklus addig fut, ameddig van mit beolvasni. A read függvényel lépünk ki a ciklusból. A while cikluson belül a forciklusban végig megyünk az összes bajton és végre hajtjuk a titkosítást.

A futtatás a következő: A fordítás: gcc fajlnev.c -o fajlnev miután lefut, utána futtatjuk: ./fajlnev 56789012 (ez a kulcs) titkosítando.txt (ide írjuk a titkosítandó txt fajl nevét, relíciós jelek között) > titkos.szoveg (titkosított fajlneve). A titkos szöveget a more titkos.szoveg parancsal nézhetjük meg.

4.3. Java EXOR titkosító

Tutor: Mózes Nőra.

A forrást a tutortol származik. A feladatban az előző feladatot fogjuk megírni Java-ban. A könyvben most találkozunk először a Java nyelvel. A Java egy objektumorientált programozási nyelv, azaz a nyelv objektumokból, osztályokból áll. A Sun Microsystems informatikai cég alkotta meg. Maga a nyelv a C és a C++ nyelvekhez hasonló, azonban sokkal egyszerűbb (az említett objektumorientáltság miatt). A kezdéshez beszéljünk kicsit az osztályokról, azaz a "Class"-okról. A Classok egy függvények csoportja. Van public és private része, a publikus függvényeket a programból bármi meghívhatja, míg a private függvényeket vagy változókat csak az osztályon belüli vagy barát függvények hívhatják meg.

```
public class ExorTitkosito {

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

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

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

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

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

    public static void main(String[] args) {

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

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

Az ExorTitkosito() függvény, kapja meg a bekért argumentumokat. Ha rosszul kapja meg, a throw() hibát ad vissza. A függvény beldejében történik a titkosítás XOR-al. Ez ugyan úgy működik, mint a fenti C kódban. Ami érdekes lehet számunkra az a byte típus, ez 8-bit. Byte tipusú lessz a kulcs és a buffer tömb is,

ezek tárolják a kulcsot és a beolvasott szöveget.

Vizsgáljuk meg a "main". A Java nyelvben a main az osztály egyik függvénye (eltére a C++ -tol, ahol a main egy különálló fő függvény az osztálytól.) Az alábbi sor "public static void main(String[] args)" a main függvény fejléce. A "public" mutatja, hogy publikus, azaz elérhető. A "static"-al jelöljük, hogy része az osztálynak. A void típust meg már ismerjük az előzőkből. A main-be képesek vagyunk argumetnumokat bekérni a terminálból. A main-en belül láthatjuk a try() és a catch() függvényt, ezekkel a függvényekkel C++ -ban, A try() a hiba üzenetet küldi még a catch() ezt elkapja és kiírja nekünk.

A fordításhoz java fordító kell. Ehez most a "javac"-t fogjuk használni. Ha ez nincs fent a számítógépünkön, akkor a gép jelezni fogja, hogyan kell telepítenünk. Fordítani és futtatni az alábbi módon fogjuk:

```
//Fordítás:  
javac ExorTitkosító.java  
//Futtatás:  
java ExorTitkosító titkosítandó.szöveg > titkosított.szöveg
```

4.4. C EXOR törő

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

Az alábbi feladatban a 3.2 feladatban lévő titkosítóhoz írunk egy programot ami feltöri a titkosított szöveget. A program alapműködése ugyan azon az elven alapszik, mint a 3.2 mivel ugyan így XOR- al alakítjuk vissza a szöveget. A lényeg, hogy a kulcsot amivel titkosítottunk azt ismerjük, mert ezzel a kulcsal tudjuk feltörni. Úgy működik, hogy a titkosított bájtokat össze exortáljuk a kulcsal, és így újra az eredeti bájtokat kapjuk. A feladatban a 3.2 ben titkosított azöveget és a kulcsot fogjuk használni, ugyanis erre épül a program.

Kód:

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256  
#define KULCS_MERET 8  
#define _GNU_SOURCE  
  
#include<stdio.h>  
#include<unistd.h>  
#include<string.h>  
  
int tiszta_lehet(const char titkos[], int titkos_meret)  
{  
    return 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;

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

    for(int i=0; i<MAX_TITKOS-(p-titkos);++i)
        titkos[p-titkos+i]='\0';

    for(int ii='0';ii<='9';++ii)
        for(int ji='0';ji<='9';++ji)
            for(int ki='0';ki<='9';++ki)
                for(int li='0';li<='9';++li)
                    for(int mi='0';mi<='9';++mi)
                        for(int ni='0';ni<='9';++ni)
                            for(int oi='0';oi<='9';++oi)
                                for(int pi='0';pi<='9';++pi)
    {
        kulcs[0]=ii;
        kulcs[1]=ji;
        kulcs[2]=ki;
        kulcs[3]=li;
        kulcs[4]=mi;
        kulcs[5]=ni;
        kulcs[6]=oi;
        kulcs[7]=pi;
```

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

Forrás:https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_parhuzamos_prog_linux/ch05s02.htm

Elsőnek is definiáljuk az állandókat és az include-kat. Az állandók közül most is a buffer a beolvasáshoz szükséges, a kulcs mérete mégint a kulcsot tartalmazó tömbhöz kell ami az előzőleg használt kód miatt 8. A fő függvény előtt találunk függvényeket. Az átlagos szóhossz és a tiszta lehet függvény a törés gyorsaságát segítik elő. Az átlagos szóhossz megadja az szóhossz atlagat még a tiszta lehet pedig a gyakori magyar szavak figyeli. A void exor () függvény megkap egy kulcsot, a méretét, a titkos szövegetnek a tömbjét és annak a méretét. És itt a forciklusban a kulcsot össze exortálja a titkos szöveggel. Az exor_tores függvény meghívja az exor függvényt is vissza adja a tiszta szöveget. A fő függvényben láthatjuk deklarációk után a titkos szöveg beolvasását. Utána a program megnézi az összes lehetséges permutációt és a megoldást kírja a kimenetre, ezzel a kódval a 3.2 programot használva fel tudjuk törni a szöveget.

4.5. Neurális OR, AND és EXOR kapu

R

A feladatban egy Neurális hálózatot fogunk írni R nyelvben. A nevét a neuron-ról kapta, ami egy idegsejt, Ezekből épül fel az idegrendszer. Ez egy ingerlékeny sejt, ami ingerület fel és leadásával továbbít információt, amit fel is dolgoz.

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

```
library(neuralnet)

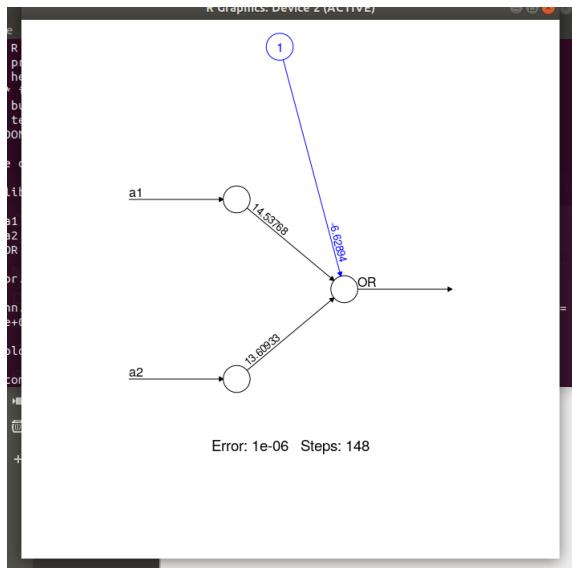
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR      <- c(0,1,1,1)

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])
```



A program elején meghyvjuk a neuralnet könyvtárat ami tartalmazza a nekünk szükséges függvényeket. A bemenet az a1 és az a2 lesz, A gép most a lofikai vagyot, azaz az OR -t fogja megtanulni. Ha a1 és a2 bemenet 0 ad, Az OR értéke is 0 lesz, minden más esetben az OR értéke 1. Ezeket az or.data-ban tárolja el a program, úgy mond "megtanulja". Az nn.or értékét pedig a neuralnet() függvényel határozzuk meg. A függvény első argumentumában a megtanuladnó érték van, aza hogy az OR értéke 0 legyen vagy 1. A második argumentumban adjuk meg az or.data ami alapján tanulja meg a program. A harmadik argumentumban rejtett neutronok száma van. A stepmax a lépésszámot adja. A plot függvényel kirajzolunk (lásd a képen) a tanulás folyamatának egyik esetét.

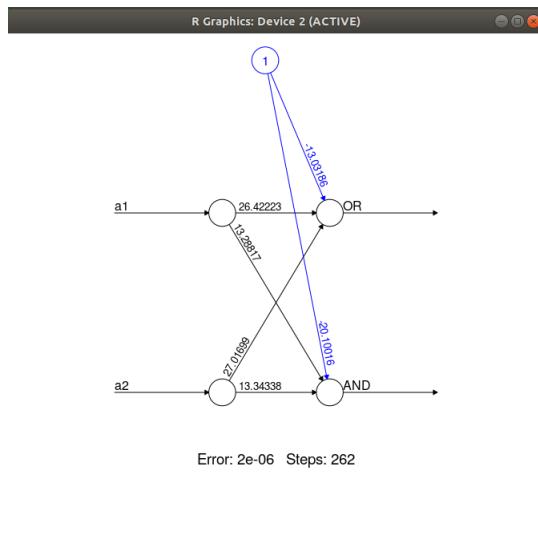
```
library(neuralnet)
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR      <- c(0,1,1,1)
AND     <- c(0,0,0,1)

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])
```



A programunk azzal bővül, hogy megtanítjuk a programnak az OR és az AND -et fogja megtanulni a program. A különbség az előzőtől annyi, hogy az AND csak akkor kap 1 értéket, ha a1 és a2 értéke is 1, különben az AND értéke 0. A tanulás folyamat ugyan olyan mint az előző. A tanulás módját az orand.data-ba mentjük.

```

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

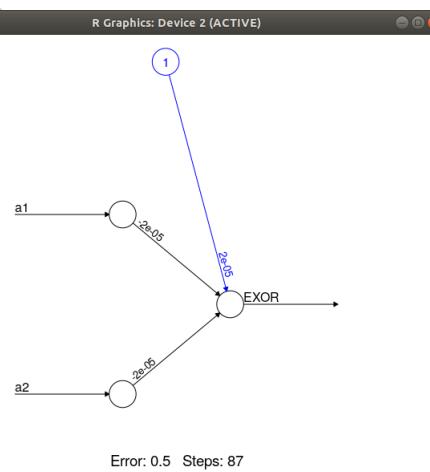
exor.data <- data.frame(a1, a2, EXOR)

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])

```



Itt pedig az EXORT tanítjuk meg a programmal. Az EXOR-nál az EXOR értéke akkor 1, ha az a1 és a2

értéke 1,0 vagy 0,1 . Ha mind akét érték 0,0 vagy 1,1 akkor az EXOR értéke 0 lesz. Ezt a tanulási mintát az exor.data-ban mentjük el. És a tanulás pont úgyanúgy van mint a fentiekben. A képen láthatjuk, hogy a program nem tanulta meg amit kell, ugyanis az eredmények hibásak. A kulcs abban van, hogy a rejtett neutronok értéke 0. A következőben nézzük meg a megoldását.

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

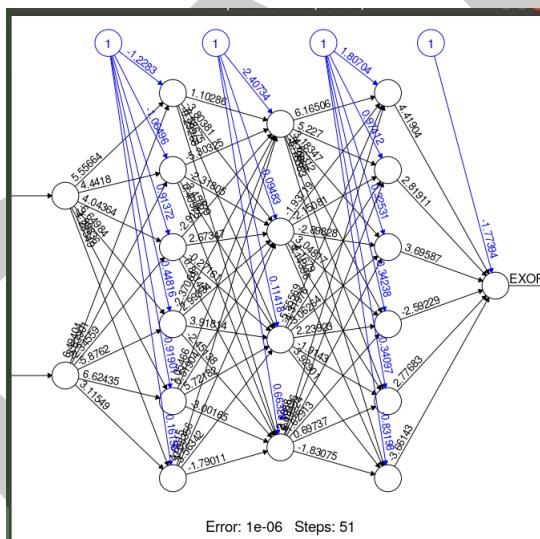
exor.data <- data.frame(a1, a2, EXOR)

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])
```

Itt anyiban változtattunk, hogy a rejtett neutronoknak létrehoztunk 3 réteget , a rétegek értékei 6,4,6. Ahogy a képen is látszik, az eredmény így jó.



4.6. Hiba-visszaterjesztéses perceptron

C++

A perceptron a mesterséges intelligenciának olyan, mint az agynak a neuron. A program képes feldolgozni és megtanulni a bemenetet, ami 0,1 ből áll.

Forrás:<https://youtu.be/XpBnR31BRJY>

Kód:

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

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

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

    double* image = new double[size];

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

    double value = (*p) (image);

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

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

A kód magyarázata: két headere van szükségünk az "mlp.hpp" és a "png++/png.hpp" -re, ezek a megjeleníté miatt kellenek nekünk és ebbe van a perceptron elve is. A fő fügvényünk elején lefoglaljuk a tárhelyet a képnak és megadjuk a méreteit. Következik a perceptron létrehozása és a megfelelő értékek hozzá adása. A "double* image = new double[size];" sorral a végén létrehozunk egy size méretű képet és utána feltölthjük a megadott képpel. A delete parancsokkal töröljük a perceptronról és a képet.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Mandelbrot halmaz egy halmaz a komplex számsíkon. Nevét Benoit Mandelbrottól kapta, aki megfogalmazta a fraktálok fogalmát (A fraktálok komplex alakzatok).

Forrás:<https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsocpp/mandelbrot/mandelbrot.cpp>

Kód:

```
include "png++-0.2.9/png.hpp"

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

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

struct Komplex
{
    double re, im;
```

```
};

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

    int i, j, k;

    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;

    struct Komplex C, Z, Zuj;

    int iteracio;

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

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

            while(Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < 255)
            {
                Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
                Zuj.im = 2 * Z.re * Z.im + C.im;
                Z.re = Zuj.re;
                Z.im = Zuj.im;
            }

            tomb[i][j] = 256 - iteracio;
        }
    }

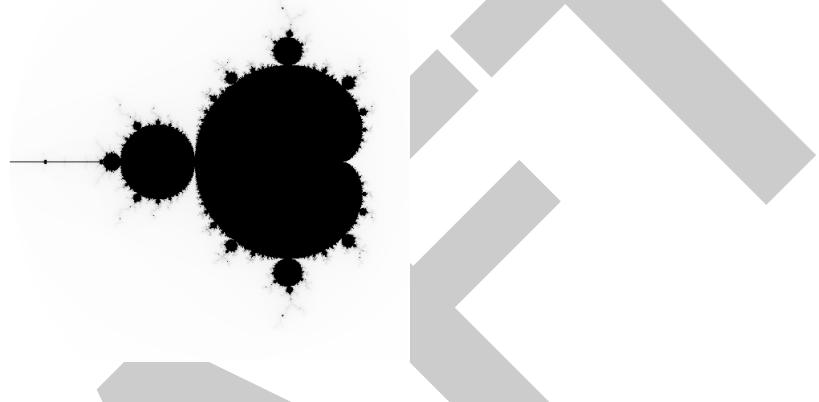
    GeneratePNG(tomb);

    return 0;
}
```

A kód magyarázata: Az include-ról kicsit lejebb, bővebben kifejtve beszélek majd. A kódot állandók definiálásával kezdjük, ilyen lesz a kép maximum szélessége, magassága. Az első függvény fogja nekünk generálni a képet. A "png" csomagot használjuk ehez. Létrehozunk egy üres pngt ami 500x500 pixel ((500X500 as mátrix)). A forcikluson belül rgb színkóddal határozzuk meg a színes pixeleket. és a "image.write" a képet kiküldjük a kimenetrre egy adott névvel. ez a függvény a fő függvény legalján lesz meghívví. A következő egy struktúra amiben 2 double típusú változót deklarálunk , ez a komplex szá-

moknak a struktúrája. Ezután a fő függvényben létrehozunk egy tömböt ami 500x500 elemű. Ezekhez az állandókat használjuk. 3 egész tipusú deklarálása után 2 double változót deklarálunk a "dx" és "dy" amivel a pixeleket fogunk meghatározni. A következő sorban lefoglaljuk a helyet c, z, zuj változóknak, utánna elvégezzük a számításokat és beletesszük azokat a tömbe és meghívjuk a függvényt amivel generáljuk.

Most nézzük meg a headert. A png++ headerre van szükségünk ahoz hogy png-t tudjunk kezelni. Ez alapból nincs meg a gépen, ezért először is le kell töltenünk az internetről egy fájlt ami tartalmazza a headert. Miután ezt letöltöttük, még telepíteni kell a libpng könyvtárat az alábbi módon: "sudo apt-get install libpng++-dev".



5.2. A Mandelbrot halmaz a std::complex osztályval

Itt a feladat ugyan az mint az előző. A különbség az, hogy itt most használhatjuk a complex headert. Ez a header már alapból tartalmaz komplex számokat, így az előző feladatban létrehozott struktúra itt már nem fog kelleni.

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

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

int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double 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 )
double dy = ( d - c )
double reC, imC, rez, imZ;
int iteracio = 0;

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

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

        reC = a + k * dx;
        imC = d - j * dy;
        std::complex<double> c ( reC, imC );

        std::complex<double> z_n ( 0, 0 );
        iteracio = 0;

        while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
        {
            z_n = z_n * z_n + c;

            ++iteracio;
        }

        kep.set_pixel ( k, j,
                        png::rgb_pixel ( iteracio%255, (iteracio*iteracio ←
                            )%255, 0 ) );
    }
}

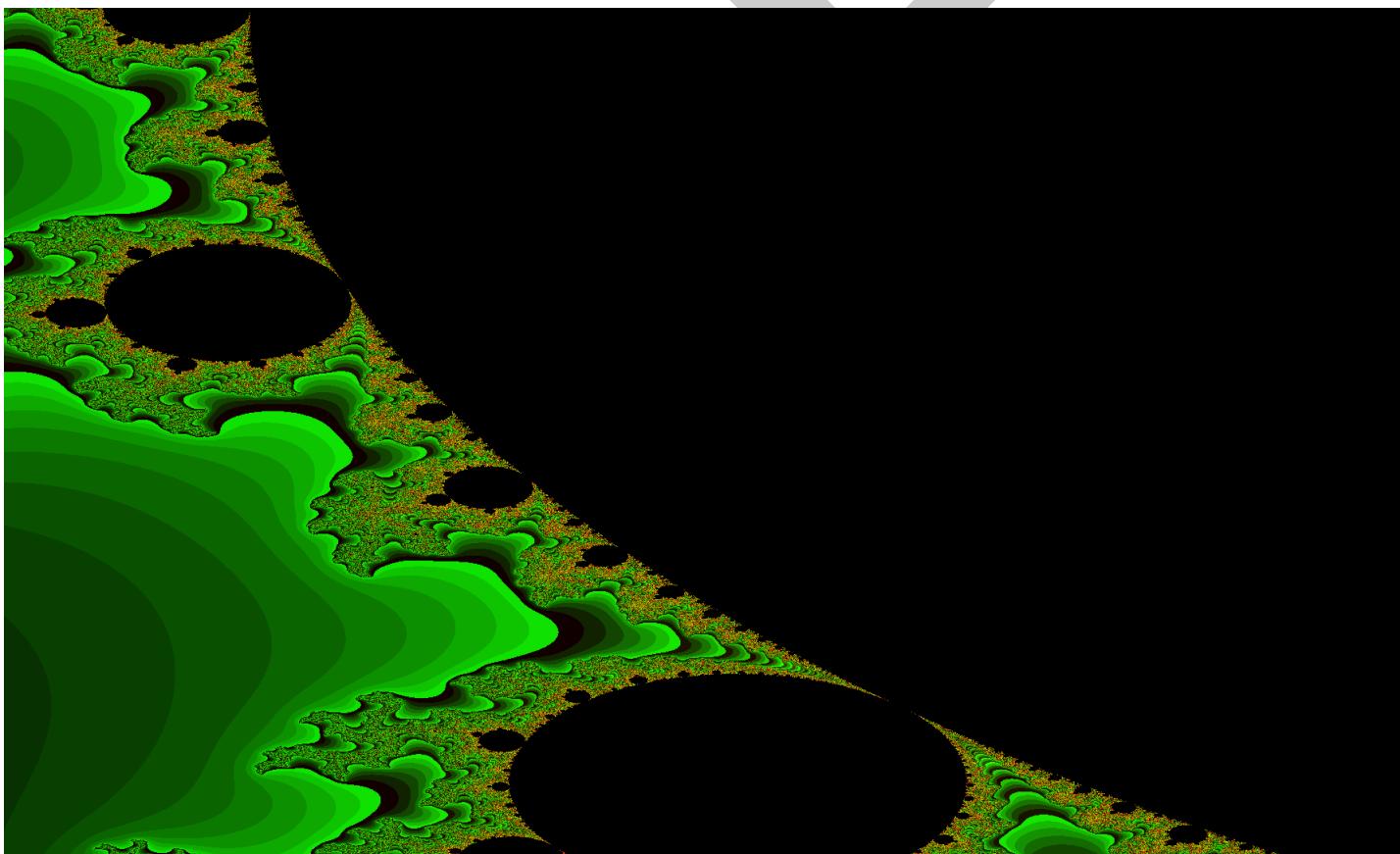
int szazalek = ( double ) j / ( double ) magassag * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
}
```

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

}
```

Kód magyarázata: A headereket már megbeszéltük. a fő függvényben dekralálunk 2 változót, ha argumentumként jól adjuk meg ezeket, akkor ezeket átadja a változóknak, ha nem jól adjuk meg, akkor kiírjuk, hogy kell helyesen használni. Ezek után megadjuk a szélességet és a magasságot, ami ebbe az esetbe FullHD. és az iterácoós határt. továbbá deklarálunk változókat amik a kép elkészítéséhez kellenek majd. Az if fügvény vizsgálja meg, hogy jól adtuk e meg az argumentumot. és itt adja át az előbb említett értékeket. Az else ág a rossz esetén, a segítséget írja ki. Ezek után lefoglaljuk a helyet a képnek. A dx, dy-hez hozzá rendeljük a megfelelő változókat. A forciklusban végig megyünk minden elemen és megadjuk a c változó értékét. Ekkor használjuk a complex-et, while ciklusban végezzük a számításokat, utánna rgb kóddal a pixeleket kiszinezzük.

A futtatáshoz szükségünk lesz a -lpng kapcsolóra.



5.3. Biomorfok

A biomorf program a mandelbrot programkódját vesszük alapul. A mandelbrot halmaz tarttarlmazza az összes ilyen halmazt. A program ugyanúgy bekéri a megfelelő bemeneteket, ha nem jó akkor kiírja. Ha jó, akkor a megfelelő változók megkapják a megfelelő értékeket. Ezután történik a kép létrehozása. Ugyan úgy megkapja a dx és dy az értéket. Aztán pedig a komplex számokat gozzuk létre. Megint végig megy a

program minden ponton és ahol kell használjuk az rgb kódos színezést. A legvégén pedig kiküldjük a képet a kimenetre.

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

Kód:

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

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

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

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

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

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

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

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

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

for ( int y = 0; y < magassag; ++y )
{

    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)
        {

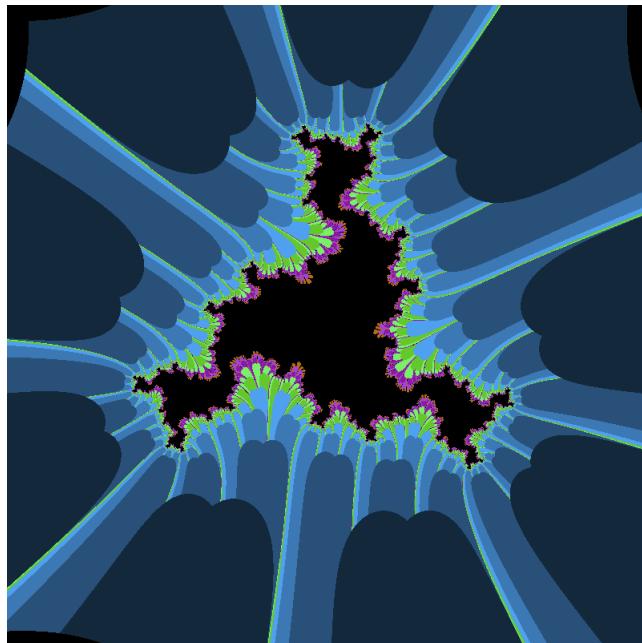
            z_n = std::pow(z_n, 3) + cc;

            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

Továbbra is a mandelbrot halmazzal foglalkozunk, de mi is az a cuda? A CUDA az Nvidia videókártyáknak egy párhuzamos számításokat segítő technológia. Használható C és C++ nyelveknél is. Ezen technika segítségével fogjuk felgyorsítani a kép létrehozását. Ehet szükségünk lesz egy Nvidida videókártyára ami rendelkezik CUDA-val. Továbbá telepítenünk kell. A kód kiterjesztése ".cu"

Megoldás forrása:https://progpater.blog.hu/2011/03/27/a_parhuzamossag_gyonyorkodtet

Kód:

```
#include <pngpp/image.hpp>
#include <pngpp/rgb_pixel.hpp>

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

#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{

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

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

```
int iteracio = 0;

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

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

while (reZ * reZ + imZ * imZ < 4 && iteracio < 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])
{
    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

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

    cudaMemcpy (kepadat, device_kepadat,
               MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
    cudaFree (device_kepadat);

}

int
main (int argc, char *argv[])
{
    clock_t delta = clock ();

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

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

    int kepadat [MERET] [MERET];

    cudamandel (kepadat);

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

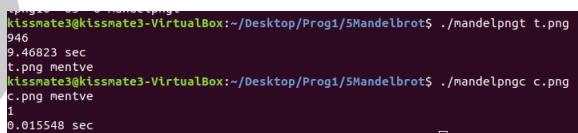
    for (int j = 0; j < MERET; ++j)
    {
        for (int k = 0; k < MERET; ++k)
        {
            kep.set_pixel (k, j,
                           png::rgb_pixel (255 -
                                           (255 * kepadat [j] [k]) / ITER_HAT,
                                           255 -
  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

```

```
    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;  
}
```

Nézzük a kódot. Az include-k alatt két állandót definiálunk, a kép méretét és az iterációs határt. A következő lépés a Mandelbrot halmaz létrehozása. ezt egy függvénytel hozzuk létre. A függvény előtt jelezzük, hogy a számításokat Cudával végezzük majd a fordításnál. A függvényen belül deklarálunk float tipusú változókat a számításokhoz. A matematikai számítás ugyan az mint az 5.1 feladatban, szóval ezt nem fejtem ki most. A következő függvény előtt nem "device" jelzés van hanem "global". Ezzel szintén azt jelezzük, hogy a Cuda fogja végezni a számítást. A "threadIdx" jelzi az aktuális szálat és a "blockIdx" hogy, melyik blokban folyik a számítás. A kép értékeit a j és a k változókban tároljuk el. Ezt a két értéket fogja kapni az előző függvény. A következő függvény a cudamandel(). Ez egy Méret x Méret azaz 600x600-as tömböt kap. Deklarálunk egy mutatót és a Malloc segítségével lefoglaljuk a megfelelő tárhelyet és a mutató ide fog mutatni. Itt hozzuk létre a megfelelő blokkokat. A végén a tárhelyet felszabadítjuk. A fő függvényünkben sem történik nagy változás. Egyből egy idő méréssel kezdünk. Lemérjük mennyi időbe telik a gépnek, hogy megalkossa a képet. Utánna deklaráljuk a tömböt, meghívjuk a cudamandel() függvényt és már az ismert módon létrehozzuk a képet.

A kódot az "nvcc" fordítóval fordítjuk, le kell tölteni, ehez a gép ad segítséget. A következőléppen fordítjuk: "nvcc mandelpngc_60x60_100.cu -lpng16 -O3 -o mandelpngc ". Miután fordítottuk utánna futtatjuk. Ha egymás mellé tesszük a Cudas és a nem kudás képalkotást, láthatjuk, hogy a kép elkészítési ideje a cudásnál sokkal gyorsabb.



```
kissmate@kissmate3-VirtualBox:~/Desktop/Prog1/5Mandelbrot$ ./mandelpngt t.png  
946  
9.46823 sec  
t.png mentve  
kissmate@kissmate3-VirtualBox:~/Desktop/Prog1/5Mandelbrot$ ./mandelpngc c.png  
c.png mentve  
1  
0.015548 sec
```

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

A program azt fogja csinálni, hogy létrejön nekünk egy mandelbrot halmaz és az egérrel képesek vagyunk belenagyítani akár a végtelenségig a halmazba.

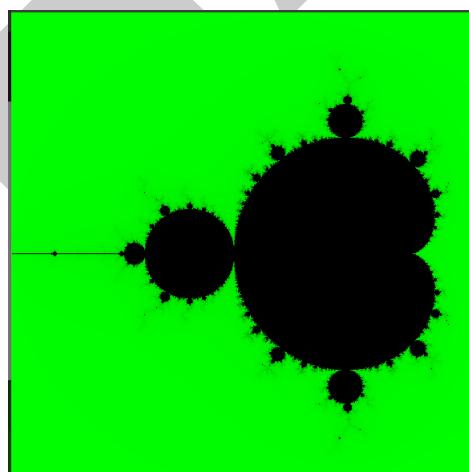
Kód:

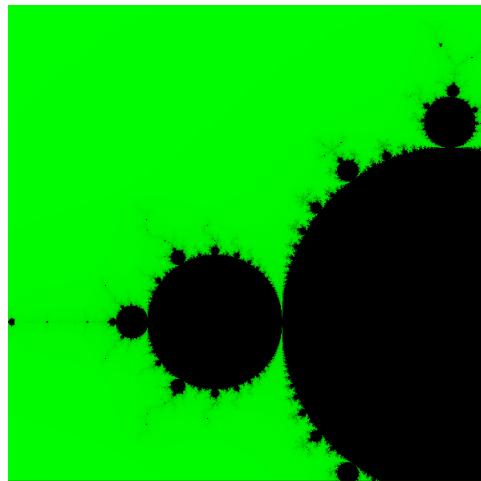
```
#include<QApplication>
#include "frakablak.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Frakablak w1,
    w2(-.08292191725019529, -.082921917244591272,
        -.9662079988595939, -.9662079988551172, 1200, 3000),
    w3(-.08292191724880625, -.0829219172470933,
        -.9662079988581493, -.9662079988563615, 1200, 4000),
    w4(.14388310361318304, .14388310362702217,
        .6523089200729396, .6523089200854384, 1200, 38655);
    w1.show();
    w2.show();
    w3.show();
    w4.show();
    return a.exec();
}
```

Ez a program nem elég önmagában, több forrásra van szükséünk. Ilyen például a frakablak.h header. Egy mappába össze kell szednünk az összes forrást. és telepítenünk kell ezt: "sudo apt-get install libqt4-dev". A qmake -project parancsal létrehozunk egy .pro fájlt. ebbe meg kell adnunk a QT+=Widgets parancsot a megfelelő helyre. Ez létrehoz egy fájlokat .o kiterjesztéssel és egy makefilet, ezek után make parancsal létrehozzuk a nagyítót. Ezek után kész is a programunk. A fraksal.cpp-ben készül el az ábránk amit majd nagyítani fogunk. Az rgb pixel színezést azonban már a frakablak végzi.

Forrás:https://progpater.blog.hu/2011/03/26/kepes_egypercesek





5.6. Mandelbrot nagyító és utazó Java nyelven

Ebben a feladatban az 5.5 feladatot fogjuk megírni Java-ban. A program lényege itt is az, hogy a mandelbrothalmazba belenagyítunk.

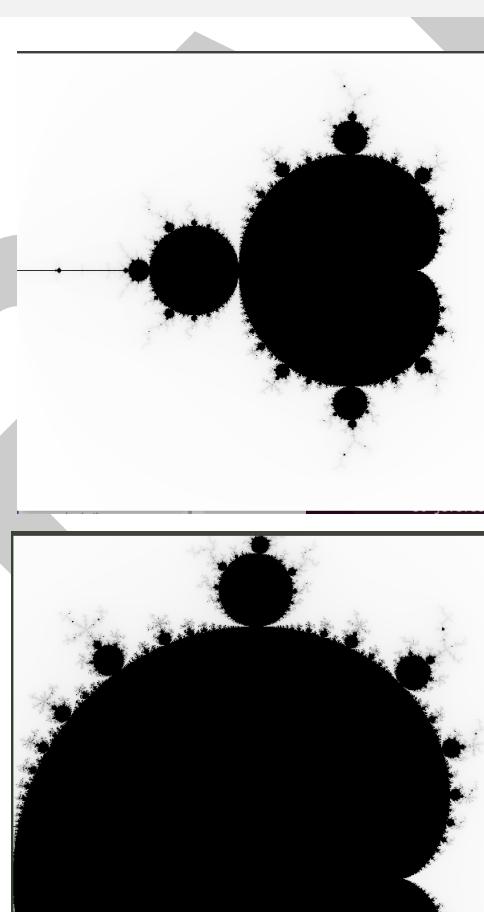
A program elején létrehozzuk a Mandelbrot halmazt. Ehez az extends szóval hozzá kapcsoljuk a Mandelbrothalmazt építő java kódunkat. A mousePressed() függvényel megadjuk a programnak az egér által kijelölt kordinátákat. Ezután A kijelölt területen újraszámoljuk a halmazt. Majd feldolgozza a létre jött kép szélét és magasságát. A pillanatfelvétel() függvénytel egy pillanatfelvételt készítünk. A függvényen belül elnevezzük a tartomány szerint és egy png formátumú képet készítünk a pillanatfelvételből. A nagyítás során láthatunk egy segítő négyzetet, ezt a paint() függvényel hozzuk létre.

A kód forrása:<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html>

```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {  
    private int x, y;  
    private int mx, my;  
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,  
        int szélesség, int iterációsHatár) {  
        super(a, b, c, d, szélesség, iterációsHatár);  
        setTitle("A Mandelbrot halmaz nagyításai");  
        addMouseListener(new java.awt.event.MouseAdapter() {  
            public void mousePressed(java.awt.event.MouseEvent m) {  
                x = m.getX();  
                y = m.getY();  
                mx = 0;  
                my = 0;  
                repaint();  
            }  
            public void mouseReleased(java.awt.event.MouseEvent m) {  
                double dx = (MandelbrotHalmazNagyító.this.b  
                    - MandelbrotHalmazNagyító.this.a)  
                    /MandelbrotHalmazNagyító.this.szélesség;  
                double dy = (MandelbrotHalmazNagyító.this.d  
                    - MandelbrotHalmazNagyító.this.c)  
                    /MandelbrotHalmazNagyító.this.magasság;
```

```
new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+ ←
    x*dx,
    MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
    MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
    MandelbrotHalmazNagyító.this.d-y*dy,
    600,
    MandelbrotHalmazNagyító.this.iterációsHatár);
}
});
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent m) {
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
}
public void pillanatfelvétel() {
    java.awt.image.BufferedImage mentKép =
        new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKép.getGraphics();
    g.drawImage(kép, 0, 0, this);
    g.setColor(java.awt.Color.BLUE);
    g.drawString("a=" + a, 10, 15);
    g.drawString("b=" + b, 10, 30);
    g.drawString("c=" + c, 10, 45);
    g.drawString("d=" + d, 10, 60);
    g.drawString("n=" + iterációsHatár, 10, 75);
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
    g.dispose();
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrotHalmazNagyitas_");
    sb.append(++pillanatfelvételSzámláló);
    sb.append("_");
    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
    sb.append("_");
    sb.append(d);
    sb.append(".png");
    try {
```

```
        javax.imageio.ImageIO.write(mentKép, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
public void paint(java.awt.Graphics g) {
    g.drawImage(kép, 0, 0, this);
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
}
public static void main(String[] args) {
    new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
```



6. fejezet

Helló, Welch!

6.1. Első osztályom

Tutor: Ignéczi Tíbor, a kód tőle származik.

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

C++:

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

double PolarGen::kovetkezo ()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1= std::rand() / (RAND_MAX +1.0);
            u2= std::rand() / (RAND_MAX +1.0);
            v1= std::rand() / (RAND_MAX +1.0);
            v2= std::rand() / (RAND_MAX +1.0);
            w= std::rand() / (RAND_MAX +1.0);
        } while ( u1== u2 || v1== v2 || w== 0.0 );
        nincsTarolt = false;
        tarolt= w;
    }
    else
        tarolt= u1;
    return tarolt;
}
```

```
v1=2*u1-1;
v2=2*u1-1;
w=v1*v1+v2*v2;
}
while (w>1);
double r =std::sqrt ((-2 * std::log (w)) /w);
tarolt=r*v2;
nincsTarolt =!nincsTarolt;
return r* v1;
}
else
{
    nincsTarolt =!nincsTarolt;
    return tarolt;
}
}
int main (int argc, char **argv)
{
    PolarGen pg;
    for (int i= 0; i<10;++i)
        std::cout<<pg.kovetkezo ()<< std::endl;
    return 0;
```

Ez a program véletlenszerűen fog számokat generálni nekünk. Ezt egy osztályon belül fogjuk kivitelezni. Az osztályunk neve a PolarGen-t kapta. Két részre tudjuk most bontani. Van egy nyilvános és egy privát rész. A nyilvánoshoz hozzá tudunk férnim viszont a privátban, csak az osztályon belül tudjuk meghívni a benne lévő változókat és függvényeket. Az osztály elején egyből ott van a koonstruktor ezt onnan tudjuk felismerni, hogy ugyan úgy hívjuk ahogyan az osztályt is. ebben kezdő értékeket tudunk adni és egy objektum létrehozásával egyből lefut. Esetünkben most a "nincsTarolt" privát változó értékét fogja "True"-ra állítani és az strand is itt lesz ami a véletlen szám generálásához kell. Utána van a destruktur ami ugyan úgy néz ki mint a konstruktor csak előtte van '~' jel. Ez a program végén fog lefutni. Ebbe szoktok felszabadítjuk a memóriát. A privát részben létrehozunk egy logikai és egy double tipusú változót. A kovetkezo() függvény az amiben a random számokat fogjuk létrehozni. Azt hogy ezt hogyan végezzük matematikailag, azt most figyelmenkívül hagyjuk. A main függvényben meghívunk egy osztálytípusú változót. Ez fogja beindítani a konstruktort. Utána pedig egy forciklusban tíz véletlen száot íratunk ki.

Java:

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

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
```

```
if(nincsTarolt)
{
    double u1, u2, v1, v2, w;
    do{
        u1 = Math.random();
        u2 = Math.random();
        v1 = 2* u1 -1;
        v2 = 2* u2 -1;
        w = v1*v1 + v2*v2;
    } while (w>1);

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

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

Ez az előző program csak javaban megírva. A program felépítése sokkal átláthatóbb és egyszerűbb lett. Alapjaiban ugyan úgy működik mint a C++ megfelelője

6.2. LZW

A program a bemeneti adatokból egy bináris fát épít. Bináris fa, ha mindegyik csomópontnak maximum 2 gyermekje van. (2 elágazása). A fa 0 és 1 számokból épül fel. A kitüntetett elem a gyökér. Innen minden elemet el tudunk érni. A következőben megnézzük, hogy is működik ez. Az eltérés, hog itt nem fő függvény van, hanem minden egy osztály része.

A kód forrása:https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat/

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

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

typedef struct binfa
{
    int ertek;
    struct binfa *bal nulla;
    struct binfa *jobb egy;
} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

    if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
        perror ("memoria");
        exit (EXIT_FAILURE);
    }
    return p;
}

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

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

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

    while (read (0, (void *) &b, 1))
    {
        if (b == '0')
        {
            if (fa->bal nulla == NULL)
            {
                fa->bal nulla = uj_elem ();
                fa->bal nulla->ertek = 0;
                fa->bal nulla->bal nulla = fa->bal nulla->jobb egy = NULL;
                fa = gyoker;
            }
        }
    }
}
```

```
    else
    {
        fa = fa->bal_nulla;
    }
}
else
{
    if (fa->jobb_egy == NULL)
    {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
    }
    else
    {
        fa = fa->jobb_egy;
    }
}
}

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

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

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

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

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

rszoras (gyoker);

double szoras = 0.0;

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

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

```
    szabadit (gyoker);
}

int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

void
ratlag (BINFA_PTR fa)
{

if (fa != NULL)
{
    ++melyseg;
    ratlag (fa->jobb_egy);
    ratlag (fa->bal_nulla);
    --melyseg;

    if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
    {

        ++atlagdb;
        atlagosszeg += melyseg;
    }
}
}

double szorasosszeg = 0.0, atlag = 0.0;

void
rszoras (BINFA_PTR fa)
{

if (fa != NULL)
{
    ++melyseg;
    rszoras (fa->jobb_egy);
    rszoras (fa->bal_nulla);
    --melyseg;

    if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
    {
        ++atlagdb;
        szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
    }
}
}

int max_melyseg = 0;

void
```

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

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

Magyarázat: Elsőnek is a szükséges headereket inculdeáljuk. Ezek után deklaráljuk a Binfánk struktúráját. A struktúra egy egészet tartalmaz aminek a neve "ertek" és 2 mutatóval fog rendelkezni, amikben bal és jobb gyermeket tároljuk majd. Az 1 érték jobbra fog kerülni, még a 0 balra. A "typedef" el adunk neki egy nevet, amivel a programon belül fogjuk hívni. Ezután az uj_elem függvény következik. ez fogja nekünk lefoglalni a tárhelyet a memóriában amit "NULL" kezdőértékkel fog rendelkezni. Ha nincs memória, akkor hibát dob ki. A végén vissza adj a lefoglalt mutatót. Utána függvény prototípusokat kapunk, ezek közül a feladatnak megfelelően csak a kiir() és a szabadit() függvényeket fogjuk megvizsgálni. Ugorjunk a main fő függvényre. AZ első egy char típusú változó, ebben fogjuk tárolni ideiglenesen a beolvasott karaktert. Aztán létrehozzuk a gyökérelemet és értékül adunk neki egy karaktert, jelen esetben '/'. A while ciklusban fog zajlani a faépítés. Először is megvizsgálja a beolvasott karakter. Mindig a gyökér elemtől indul. Ha a beolvasott karakter értéke 0, akkor először megvizsgálja, hogy a gyökérnek vagy az adott csomópontnak van-e bal_nullas gyermeke, ha van, akkor rálép a csomópontra, ha viszont nincs akkor a gyökérnek vagy az adott csomópontnak létrehoz egy bal_nullas gyermeket. Ha a beolvasott karakter értéke 1, akkor a program ugyan ezen az elven mint a 0 -ás értéknél végig vizsgálja csak a jobb_egyes gyermekkel. Most következik a kiír és a szabadit függvény. A szabadit() függvény egy rekurzív függvény. Törli a memóriából az eltárolt elemeket. A kiir() függvény is rekúrzív függvény. Bejárja a fa elemeit.

6.3. Fabejárás

Tutor:Földesi Zoltán

A fabejárásnak 3 tipusa van, preorder, inorder és postorder. Azt hogy hogyan járja be a fát már a neve is rejtő. Kezdjük a preorder fabejárással. Itt mindenkor a gyökérrel kezdi a program a vizsgálatot ó, aztán a bal oldalt járja be és legvégezetül pedig a jobb oldalt fogja bejárni. Az

Kezdjük a preorder fabejárással. Itt mindenkor a gyökérrel kezdi a program a vizsgálatot ó, aztán a bal oldalt járja be és legvégezetül pedig a jobb oldalt fogja bejárni.

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

Inorder fabejárás: Az inorder fabejárásnál először a bal oldalt vizsgáljuk meg, utánna jön a gyökér és legvégül pedig a jobb oldalt nézzük. Erre példa az előző programban lévő kiir függvény, ahol inorder fabejárás van.

A postorder fabejárás: Itt először a fa bal oldalát fogja átvizsgálni aztán a jobb oldalt és legutoljára pedig a gyökeret vizsgáljuk.

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

}

6.4. Tag a gyökér

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

Megoldás forrása:https://progpater.blog.hu/2011/03/31/imadni_fogjatok_a_c_t_egy_emberkent_tiszta_szivbol

Az alábbi program a fenti C változatnak lesz úgymond a C++ változata. Az első lépés, hogy ami C-ben struktúra volt, azt átírjuk C++-ban egy osztályba, mivel a C++ -ban megtehetjük. Ez az alábbi módon fog kinézni:

```
class LZWBInFa
{
public:
    LZWBInFa (char b = '/') : betu (b), balNulla (NULL), jobbEgy (NULL) ←
    {};
    ~LZWBInFa () {};
    void operator<<(char b)
    {
        if (b == '0')
        {
            // van '0'-s gyermeke az aktuális csomópontnak?
            if (!fa->nullasGyermek ()) // ha nincs, csinálunk
            {
                Csomopont *uj = new Csomopont ('0');
                fa->ujNullasGyermek (uj);
                fa = &gyoker;
            }
            else // ha van, arra lépünk
            {
                fa = fa->nullasGyermek ();
            }
        }
        else
        {
            if (!fa->egyesGyermek ())
            {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }
}
```

Ezen belül fogjuk a gyökeret létrehozni és értékül adni neki a '/'-t. A mutatóinak értékét 0-ra állítjuk. Ezek után jön a faépítés a már fentiekben elmagyarázott módon. A program megnézi, hogy 1-est vagy 0 érkezik a bemenetről. itt azt látjuk, hogy a betétel a << operátorral történik, ez annyiban különbözik a C-ben írt programtól, hogy ez egyből beleteszi a fába a beérkezett karaktert. Egy új csomópontot a "new" szóval tudunk létrehozni, ha szükséges. Ez azért lehetséges mert van egy Csomópont osztályunk (Lásd a lenti programban). A Class csomóponton belül az egyesGyermekek() és a nullasGyermekek() függvények a gyermekükre mutató pointereket fogják tartalmazni. Az ujNullasGyermekek és au ujEgyesGyermekek-nek pedig adunk egy gyermeket ás arra fogja állítani a mutatót. A private részben fogjuk ezeket deklarálni, ez azt jelenti, hogy csak az osztályon belül használhatóak ezek a változók. A legvégén jön a main főfüggvény. Itt deklaráljuk a char típusú változót amibe beolvassunk és innen kerül az osztályokhoz. Végül meghívjuk a kiir és a szabadit függvényeket amire példát az előző programokban találunk. Ugye a kiir()-al kiíratjuk az eredmény és a szabadit()-al pedig felszabadítjuk a lefoglalt merőriát.

```
class Csomopont
{
public:
    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0) {};
    ~Csomopont () {};
    Csomopont *nullasGyermekek () {
        return balNulla;
    }
    Csomopont *egyesGyermekek ()
    {
        return jobbEgy;
    }
    void ujNullasGyermekek (Csomopont * gy)
    {
        balNulla = gy;
    }
    void ujEgyesGyermekek (Csomopont * gy)
    {
        jobbEgy = gy;
    }
private:
    friend class LZWBInFa;
    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &);
    Csomopont & operator=(const Csomopont &);
};

int main ()
{
    char b;
    LZWBInFa binFa;
    while (std::cin >> b)
    {
```

```
    binFa << b;
}
binFa.kiir ();
binFa.szabadit ();
return 0;
}
```

6.5. Mutató a gyökér

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

Vegyük alapul a C++ ban megírt LZWBina-t. első dolgunk, hogy a fában a gyökér elemet átalakítjuk egy mutatóvá. Azt az alábbi módon fogjuk megcsinálni: A gyökér elem a protected részén van az osztálynak. Itt az eredeti "Csomopont gyoker;" az alábbi módon átírunk:

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

Ugye C++ ban a mutatót egy '*'-al jelüljük. Ha most futtatnánk a programot, akkor számtalan hibába ütközénk. Ezeket ki kell javítanunk. A programban így nem a gyökér memóriacímét kell átadnunk (Töröljük az összes referenciajelet a gyokerek előtt) és mivel mutató lett a gyökér így nem '.'-al hiavatkozunk hanem '->'-al. Itt láthatunk példát arra, hogy hogyan:

```
//előtte:
fa=&gyoker;
//utánna:
fa=gyoker;

//előtte:
szabadit (gyoker.egyesGyermek ());
szabadit (gyoker.nullasGyermek ());
//utánna:
szabadit (gyoker->egyesGyermek ());
szabadit (gyoker->>nullasGyermek ());

}
```

Ha mindezek után lefuttatjuk a programunkat az lefordul, azonban futtatáskor szegmentális hibába ütközünk. Ez azért van, ugyanis a gyökér memóriacíme nincs lefoglalva. Ennek a megoldását a konstruktorban és a destrukturban fogjuk megalkotni. A konstruktorban foglaljuk le és a destrukturba fogjuk törölni a lefoglalt memóriát. Lásd:

```
LZWBinFa ()  
{  
    gyoker= new Csomopont ('/');  
    fa = gyoker;  
}  
~LZWBinFa ()  
{  
    szabadit (gyoker->egyesGyermek());  
    szabadit (gyoker->nullasGyermek());  
    delete(gyoker);  
}
```

6.6. Mozgató szemantika

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

DRAFT

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

Ebben a feladatban egy QT programban fogjuk szimulálni a hangyák mozgását. A való életben a hangyák sose zavarodnak össze, sose torlódnak fel, sőt minél többen vannak annál jobban és gyorsabban mozognak. Továbbá tudjuk, hogy a látásuk nem a legjobb. Tehát a kulcs az egymás közötti kommunikáció, ezt feromonokkal érik el. Ebben a szimulációban mi is úgymond feromonokkal fogjuk a hangyák közötti kommunikációt elérni. A szabály, hogy mindenkor a legerősebb feromonú hanyga felé lépünk, a programban látszik, hogy a hanygák feromon csíkokat hagynak maguk után, ami idő elteltével egyre gyengül míg el nem tűnik.

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

Kód forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Myrmecologist

Kód:(csak a main.cpp van itt a többi kód a forrásnál található)

```
#include <QApplication>
#include <QDesktopWidget>
#include <QDebug>
#include <QDateTime>
#include <QCommandLineOption>
#include <QCommandLineParser>

#include "antwin.h"

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

    QCommandLineOption szeles_opt ( { "w", "szelesseg" }, "Oszlopok (cellákban ←
        ) száma.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( { "m", "magasság" }, "Sorok (cellákban) ←
        száma.", "magasság", "150" );
```

```
QCommandLineOption hangyszam_opt ( {"n","hangyszam"}, "Hangyak szama. ←
    ", "hangyszam", "100" );
QCommandLineOption sebesseg_opt ( {"t","sebesseg"}, "2 lepes kozotti ←
    ido (milliseconben).", "sebesseg", "100" );
QCommandLineOption parolgas_opt ( {"p","parolgas"}, "A parolgas erteke. ←
    ", "parolgas", "8" );
QCommandLineOption feromon_opt ( {"f","feromon"}, "A hagyott nyom ←
    erteke.", "feromon", "11" );
QCommandLineOption szomszed_opt ( {"s","szomszed"}, "A hagyott nyom ←
    erteke a szomszedokban.", "szomszed", "3" );
QCommandLineOption alapertek_opt ( {"d","alapertek"}, "Indulo ertek a ←
    cellakban.", "alapertek", "1" );
QCommandLineOption maxcella_opt ( {"a","maxcella"}, "Cella max erteke." ←
    , "maxcella", "50" );
QCommandLineOption mincella_opt ( {"i","mincella"}, "Cella min erteke." ←
    , "mincella", "2" );
QCommandLineOption cellamerete_opt ( {"c","cellameret"}, "Hany hangya ←
    fer egy cellaba.", "cellameret", "4" );
QCommandLineParser parser;

parser.addHelpOption();
parser.addVersionOption();
parser.addOption ( szeles_opt );
parser.addOption ( magas_opt );
parser.addOption ( hangyszam_opt );
parser.addOption ( sebesseg_opt );
parser.addOption ( parolgas_opt );
parser.addOption ( feromon_opt );
parser.addOption ( szomszed_opt );
parser.addOption ( alapertek_opt );
parser.addOption ( maxcella_opt );
parser.addOption ( mincella_opt );
parser.addOption ( cellamerete_opt );

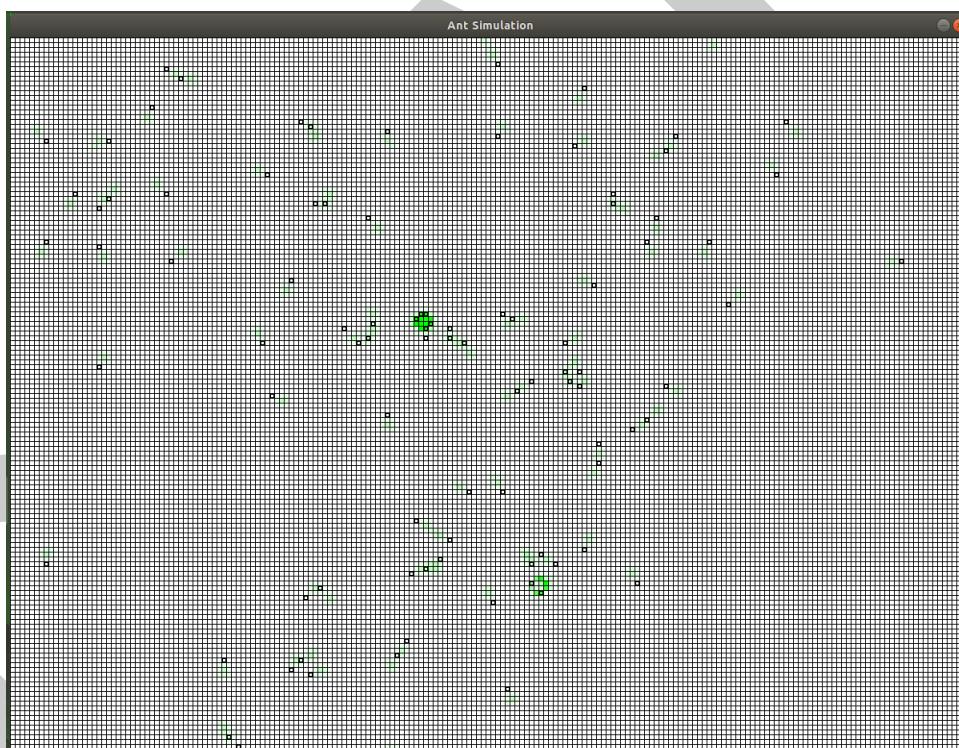
parser.process ( a );

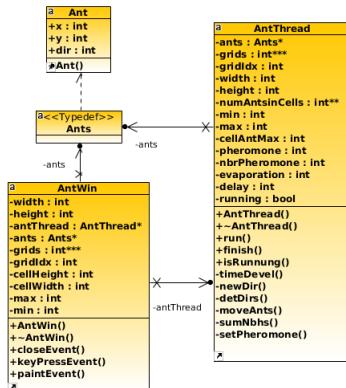
QString szeles = parser.value ( szeles_opt );
QString magas = parser.value ( magas_opt );
QString n = parser.value ( hangyszam_opt );
QString t = parser.value ( sebesseg_opt );
QString parolgas = parser.value ( parolgas_opt );
QString feromon = parser.value ( feromon_opt );
QString szomszed = parser.value ( szomszed_opt );
QString alapertek = parser.value ( alapertek_opt );
QString maxcella = parser.value ( maxcella_opt );
QString mincella = parser.value ( mincella_opt );
QString cellameret = parser.value ( cellamerete_opt );

qsrand ( QDateTime::currentMSecsSinceEpoch() );
```

```
AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), feromon ←  
    .toInt(), szomszed.toInt(), parolgas.toInt(),  
    alapertek.toInt(), mincella.toInt(), maxcella.toInt(),  
    cellameret.toInt() );  
  
w.show();  
  
return a.exec();  
}  
}
```

Kezdjük az ant.h tartalmazza a hangya tulajdonságait. Hol van az x és y tengelyen és hogy merre mutat az iránya, merre megy. Az antwin-ban van pedig a hangyaboly(ants). Továbbá ezen belül adjuk meg, hogy egy cella hány pixelből álljon és hogy az ablak szélessége és magassága hány. Forciklus-okkal felépítjük cellákóból az ablakot és elhelyezzük benne a hangyákat(azt ant-ből). Új és új hangyák jelennek meg. Ezek megváltoztatják a régebbi hangyák irányát. Az antwin.h tartalmazza a billentyűzet parancsait, például a p vel megállítjuk a folyamatot. Az antheard.cpp tartalmazza a mozgáshoz, törléshez, az új irány megadásához, a hangyák számának eltárolásához szükséges függvényeket. Itt vizsgálja azt is hogy a hangyák száma nő e vagy csökken az idő mulásával.





A kép, a megoldás videóból származik.

7.2. Java életjáték

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

Az életjátékpt azaz sejtautómatákat először Naumen János vetette fel. A felvetés a gép önreprodukciójának matematikai modellalkotást tartalmazta. A legismertebb modell a John Horton Conway-féle életjáték. Maga a "játék" egy négyzetrácsos mezön zajlik amin mozognak a sejtek. A sejtek "élete" szabályokhoz van kötve. Megvan adva hogy mi a feltétele hogy egy sejt létrejöjjön, életbenmadardjon vagy elpusztuljon. Conway erre 3 feltételt szabott meg:

1.szabály: Egy sejt csak úgy éli túl, ha kettő vagy három szomszédja van.

2.szabály: Egy sejt akkor pusztul el, ha kettőnél kevesebb szomszédja van. Ezt elszigetelődésnek hívjuk. A másik eset hogy akkor pusztul el ha háromnál több szomszédja van. Ezt túlnépesedésnek hívjuk.

3.szabály: A harmadik szabály a születésre vonatkozik, és akkor történik meg ha egy cellának a körzetében 3 sejt található.

Ezen a 3 szabály meghatározásával kapunk egy önműködő sejtautómát. Beleszolásunk csak kezdetben van, utánna a szabyályok szerint önállóan működik a program. Mi most külön a sikló-kilövőt fogjuk vizsgálni. Hogy ezt elérjük, rögzítenünk kell adott cellákban sejteket, így létre jön egy "sikló ágyú", ez időközönként "siklókat" fog lőni.

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apb.html?fbclid=IwAR0Gc-Q7v353I_qDrqAd4LfIhWrzTwYnnsTZ5wTpBAhQjwZ63pl2moebOpY

Kód:

```

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() {
        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() {  
    // Vonszolással jelöljük ki a négyzetet:  
    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) { // végig lépked a sorokon  
        for(int j=0; j<rács[0].length; ++j) { // s az oszlopok  
            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) { // sorok
        for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok

            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;

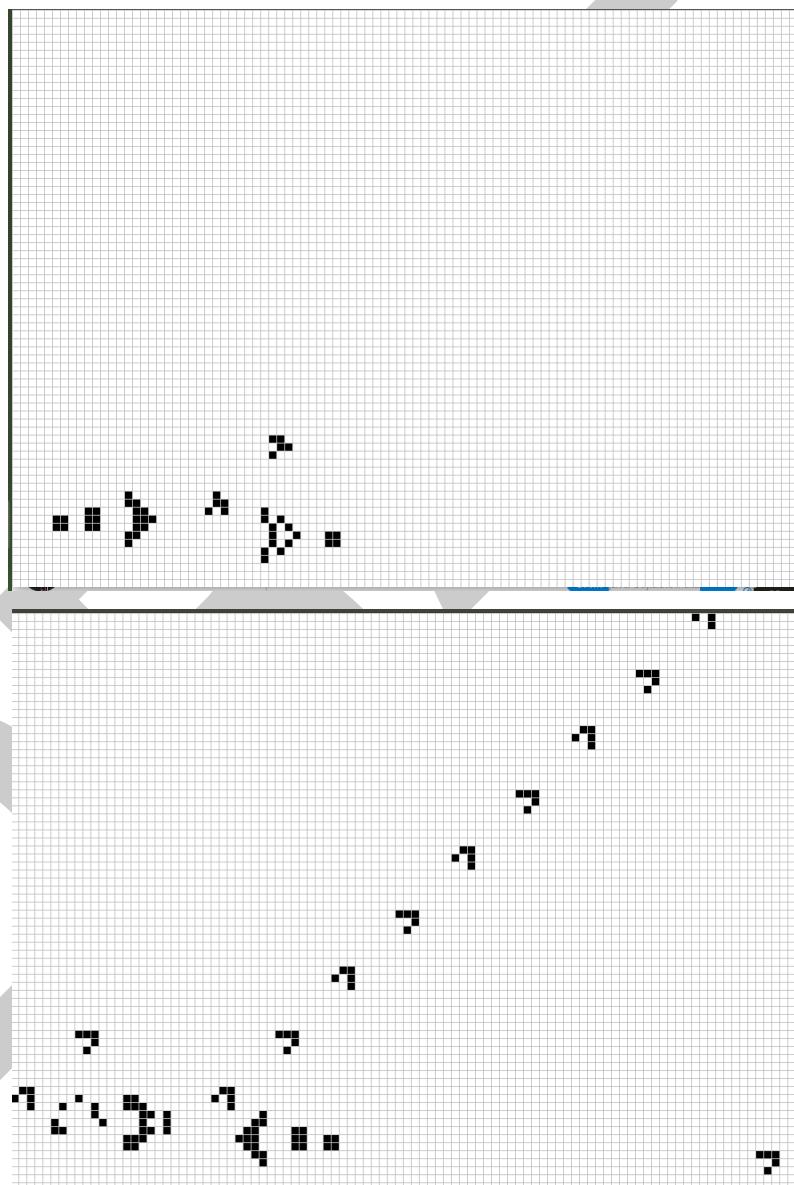
rács[y+ 10][x+ 22] = ÉLŐ;
rács[y+ 10][x+ 23] = ÉLŐ;
rács[y+ 10][x+ 24] = ÉLŐ;
rács[y+ 10][x+ 25] = ÉLŐ;

rács[y+ 11][x+ 25] = ÉLŐ;

}

public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    // A pillanatfelvétel kép fájlneve
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételszámláló);
    sb.append(".png");
    // png formátumú képet mentünk
    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);
}
```

A program elején megadjuk, hogy egy sejt lehet élő vagy halott. A feladatban 2 rácsfélét használunk, az egyik rács a sejt állapotát fogja tárolni míg a második az egy másdopercel későbbi tulajdonságait. Meghatározzuk az aktuális rácsot a rácsIndex-el. Utánna pedig egy cella magasságát és szélességét, ezt követően pedig, hogy hány cellából álljon a "játék". A következő hogy a az állapotok között mennyi idő teljen el. A függvények közül az első függvény megkapja a méreteket és létrehozza az ablakot. Itt készíti el a 2 rácsot is és az indexet is elindítja. Kezdetben minden rács HALOTT. Ezen belül lesz meghívva a siklólövő aminek a kód végén minden kordinátája megvan adva. Vannak billentyűről beérkezőparancsaink is, különböző feladatokkal ellátva pl a "g" betűvel, a két állapot közötti időt csökkentjük. Ugyan így vannak az egérrel történő infomrációk feldolgozására szolgáló függvények. Külön a kattintásra és a mozgatásra. Külön tudunk készíteni pillanatfelvételt az aktuális állapotról az "s" gomb segítségével. A programban a sejtér rajzolását a paint() függvénytel végezzük. A szomszédokSzáma() függvéyenben vizsgáljuk a szabályokat és aszerint történik a sejtek viselkedése.



7.3. Qt C++ életjáték

Most alkossuk meg az életjátékot Qt C++-ban is. A program lényege itt is ugyan az mint a java-ban, ugyan azok a szabályok. Ha kettő vagy górom szomszédja van, akkor életben marad, ha 2 nél kevesebb kipustul, ha 3 nál több, akkor tulnápesedés miatt pusztul ki. Itt is a siklóágú lesz a fő célunk.

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

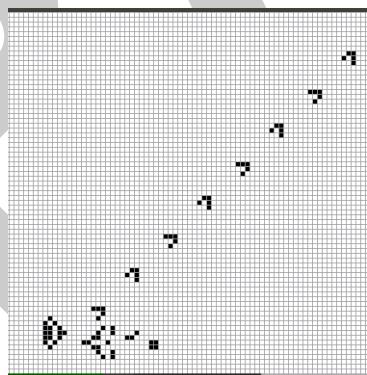
Kód:(ez csak a main.cpp a többi szükséges fájl a forrásnál található.)

```
#include <QApplication>
#include "sejtablak.h"
#include <QDesktopWidget>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    SejtAblak w(100, 75);
    w.show();

    return a.exec();
}
```

A forrásfájljaink a Sejtablak.cpp , sejtszal.h, sejtszal.cpp és a sejtablak.h . Ezek az átláthatóság miatt vannak külön. A sejtablak.h és .cpp tartalmazza a függvényeket amivel majd a kirajzolás fog történni és ebben van a sikló lövés is,úgy mint a java-s pájjánál, külön minden egyes cellát megadunk amiben sejt van. A szejtszal.h és .c pedig az életjátékhoz szükséges szabályokat . Ezen belül vannak a függvények melyek az adott állapotokat vizsgálják és a szabályok szerint alakítják a programot.



7.4. BrainB Benchmark

Tutorált: Földesi Zoltán

Elsőnek is a Benchmark jelentését nézzük meg. A benchmark egy elemzés, tesztfeladat. Egy bizonyos tesztet végez el és azt az elért pontszám alapján összehasonlítja a tesztet elvégzők között. Ilyen például telefonok teljesítményét végző benchmark , vagy az esetünkben az agy teljesítményét vizsgáló Benchmark. Ennek segítségével tudunk egy vizsgálati alapot venni egy adott feladatban. Például a telefonoknál, hogy minél több pontot ér el annál jobb a teljesítménye és össze tudjuk hasonlítani más telefonok teljesítményével. Ugyan így a BraniBenchmarkban, a tesztet megoldó emberek közül az adott pontszám megadja

hogy ki teljesített a legjobban és egymáshoz is tudjuk vizsgálni őket. Az adott programunk az egyének figyelemképességét és koncentrációját fogja vizsgálni. Adott egy karakter, ami a mi karakterünk és azon kell tartanunk az egér kurzort. A program azt vizsgálja, hogy mennyi ideig vagyunk képesek a kurzort a mi karakterünkön tartani, azaz meddig nem veszítjük el azt. Persze nem ilyen egyszerű, mert közben rengeteg új karakter jelenik meg a monitoron befolyásolva ezzel minket, hogy el ne veszítsük a karakterünket. A program arra is reagál, ha elveszítjük a karakterünk.

Az adott programban a mi karakterünk Samu lesz. Samut figyelemmel kell tartani. Ahogy fent említettem ezt a kurzorral fogjuk megtenni. Minél tovább tartjuk Samun a kurzort annál több másik karakter lesz a képernyőn, A feladat 10 percig tart és annál jobb vagy ha minél több kis karakter között is megtudod tartani a saját karaktered. Ha elveszítenéd abban az esetben belassul az új karakterek megjelenése még meg nem találod. Annál jobban teljesítettél, minél több pontod van a 10 perc végén.

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

Kód:

```
eturn a.exec();  
#include <QApplication>  
#include <QTStream>  
#include <QtWidgets>  
#include "BrainBWin.h"  
  
int main ( int argc, char **argv )  
{  
    QApplication app ( argc, argv );  
  
    QTextStream qout ( stdout );  
    qout.setCodec ( "UTF-8" );  
  
    qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( " ↵  
        Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;  
  
    qout << "This program is free software: you can redistribute it and ↵  
        /or modify it under" << endl;  
    qout << "the terms of the GNU General Public License as published ↵  
        by the Free Software" << endl;  
    qout << "Foundation, either version 3 of the License, or (at your ↵  
        option) any later" << endl;  
    qout << "version.\n" << endl;  
  
    qout << "This program is distributed in the hope that it will be ↵  
        useful, but WITHOUT" << endl;  
    qout << "ANY WARRANTY; without even the implied warranty of ↵  
        MERCHANTABILITY or FITNESS" << endl;  
    qout << "FOR A PARTICULAR PURPOSE. See the GNU General Public ↵  
        License for more details.\n" << endl;  
  
    qout << QString::fromUtf8 ( "Ez a program szabad szoftver; ↵  
        terjeszthető illetve módosítható a Free Software" ) << endl;  
    qout << QString::fromUtf8 ( "Foundation által kiadott GNU General ↵  
        Public License dokumentumában leírtak;" ) << endl;
```

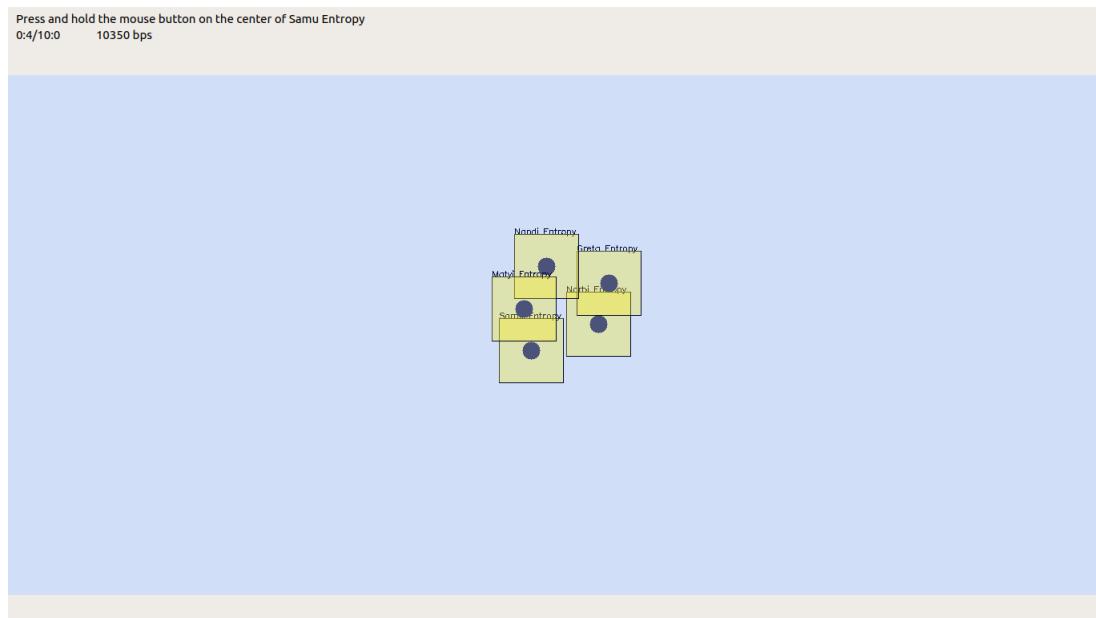
```
qout << QString::fromUtf8 ( "akár a licenc 3-as, akár (tetszőleges) ←
    későbbi változata szerint.\n" ) << endl;

qout << QString::fromUtf8 ( "Ez a program abban a reményben kerül ←
    közreadásra, hogy hasznos lesz, de minden" ) << endl;
qout << QString::fromUtf8 ( "egyéb GARANCIA NÉLKÜL, az ←
    ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ" ) << endl;
qout << QString::fromUtf8 ( "ALKALMAZHATÓSÁGRA való származtatott ←
    garanciát is beleértve. További" ) << endl;
qout << QString::fromUtf8 ( "részleteket a GNU General Public ←
    License tartalmaz.\n" ) << endl;

qout << "http://gnu.hu/gplv3.html" << endl;

QRect rect = QApplication::desktop()->availableGeometry();
BrainBWin brainBWin ( rect.width(), rect.height() );
brainBWin.setWindowState ( brainBWin.windowState() ^ Qt::←
    WindowFullScreen );
brainBWin.show();
return app.exec();
}
```

Ez ismét egy QT program, az összes szükséges fájl megtalaálható a forrásnál. Fent csak a main.cpp látható de itt a többiről is beszélünk. Először nézzük a BraintBTheard.cpp-t. Itt áll elő a kezdő pozíció. Létrejön a mi karakterünk és 4 másik karakter. Ezek úgy vannak minden elhelyezve, hogy minden egymás közelébe legyenek, hogy a feladatunk ne legyen túl könnyű. A run függvény a teszt indításáért felel. Itt méri az időt is, azaz a program addig fut amíg az idő a megadott időnl(10 perc) kisebb. Itt található még a pause függvény aminek a neve a válasz. A következő BraintBTheard.cpp található a karakterek kinézete, ezeknek a neve programon belül "hero" azaz hős. Itt adjuk meg a nevét, az elhelyeszkedését, színét, és a mozgásának a gyorsaságát. Továbbá a többi karakter születése, gyorsasága és további információkat róluk itt adunk meg. A BrainBWin.cpp -ben megadjuk a program nevét a verzió számát. majd az UpdateHeroes függvényben zajlik a a kurzorunk menetének vizsgálata, itt figyeli hogy rajta vagyunk e az egérrel a karakteren, hányszor veszítettük el a karakterünk, vagy hogy éppen fut e a teszt. Ezek utána a kövezkező függvény kirajzolja az ablakot. Itt van továbbra az óra megjelenítése vagy a pontszámunkké. Ezek után jön az egér funkciói. Például ha lenyomjuk akkor elindul. Vagy az elmozdítás követése, És a billentyűzetről bevitt karaktereket is itt dolgozza fel.



DRAY

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Először is a Python nyelvet mutatjuk be. A nyelv egy magas szintű programozási nyelv. 1991 ben került nyilvánosság elé, amit Guido holland származású programozó fejlesztett. A python interpreteres nyelv.

A Minst kézzel írott számok adatbázisa (6000 kép). Ez az alapja azoknak a programoknak ami képről ismeri fel a tárgyakat. A kézel beírt számokról a program el fogja dönteni hogy milyen szám, ez azért érdekes, mert a kézzel írott írás szinte mindenkinél másabb, viszont a programnak mindenkor tudnia kell, hogy melyik számot kell felismernie. A programhoz a TensorFlowet használjuk. A TensorFlow a Google által alkotott gépi tanulási rendszer. Sok helyen használják, az egyik leghasználtabb, az a google mapsban található utcakép. Ez neutrális háló helyett itt transzformációs gráfok találhatóak. A TensorFlow nyílt forráskódú, le kell töltenünk a használathoz. Nézzük a kódot. Először is a könyvtárakat amik kellenek, a from kulcsszóval fogjuk ezeket hozzáadni a programhoz. Utána beimportáljuk a Tensorflow könyvtárt. Ezután következik a main. Ebben van a kiíratás felépítése, hogy hogyan küldjük ki az eredményeket (a képen látszik.). A sess azzaz egy session segítségével fogjuk a tanítást végezni, hasonlóan mint a neurális hálózatnál. Az alaposság kedvéért, hogy minél pontosabb eredményt kapunk ezerszer futtatjuk a ciklust. Ztánna kiíratjuk mennyire lett pontos az eredmény. A programban először felugrik maga a kép ami a kézel írott számot taartalmazza(ehez a matplotlib-ot használjuk, hogy meg tudjuk rajzolni), ha ezt bezárjuk jön a következő kép, a képeket az aktuális mappában lementjük. A programnak elég nagy a pontossága, jól felismeri. Az eredményeket egy tömben tárolja el a program.

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

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse

# Import data
from tensorflow.examples.tutorials.mnist import input_data
```

```
import tensorflow as tf

import matplotlib.pyplot

FLAGS = None

def main(_):
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y = tf.matmul(x, W) + b

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

    cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, ←
        y_))
    train_step = tf.train.GradientDescentOptimizer(0.5).minimize(←
        cross_entropy)

    sess = tf.InteractiveSession()
    # Train
    tf.initialize_all_variables().run()
    print("-- A halozat tanitasa")
    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
        if i % 100 == 0:
            print(i/10, "%")
    print("-----")

    # Test trained model
    print("-- A halozat tesztelese")
    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print("-- Pontossag: ", sess.run(accuracy, feed_dict={x: mnist.test.←
        images,
                    y_: mnist.test.labels}))
    print("-----")

    print("-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a ←
        tovabbelpeshez csukd be az ablakat")

    img = mnist.test.images[42]
    image = img
```

```
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("4.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

print("-- A saját kezi 8-asom felismerése, mutatom a számot, a -->
      továbblepeshez csukd be az ablakat")

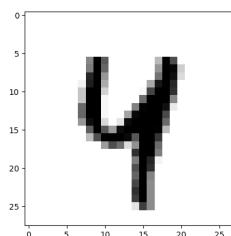
img = readimg()
image = img.eval()
image = image.reshape(28*28)

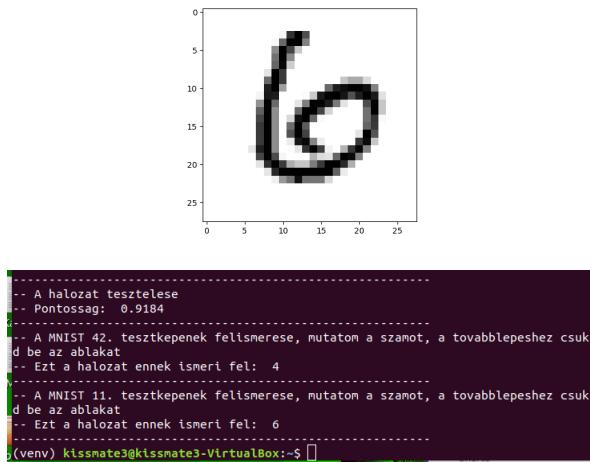
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ -->
        mnist/input_data',
                        help='Directory for storing input data')
    FLAGS = parser.parse_args()
    tf.app.run()
```





8.2. Mély MNIST

Python

Itt használom fel az első passzolási lehetőséget a SMNISTforHumansExp3 ban elért és közzé tett eredmény miatt.

8.3. Minecraft-MALMÖ

Megoldás forrása: https://bhaxor.blog.hu/9999/12/31/minecraft_steve_szemuvege

Kezdjük azzal, hogy mi is az a Malmö. Ez egy Microsoft projekt a neve Project Malmo. A lényege pedig, hogy a mesterséges intelligencia segítségével feldolgozzuk az adott környezetet. A kísérletet az ismert játékban a Minecraftban végezzük. A feladat lényege, hogy mesterséges intelligenciával övezzük fel a karaktert és elindítjuk utjára, a MI feladata, hogy a környezetet érzékelje a karakter és kikerülje az akadályokat, ne akadjon el. A szükséges fájlok megtalálhatók a Microsoft Githubján. A feladat megoldása a következő. Mindig adott egy kocka amin a karakterünk épp áll, ez a kocka kürüli 26 kockából álló területet fogja a program vizsgálni. Azaz összesen 27 kockát vizsgálunk. Ezeket a kockákat megkülönböztetjük. Ugye a játékban lehet fű (fold is az), levegő, magasfű, levél. Ezeket vizsgálva kell a programnak eldönteni hogy a karakter merre mozogjon. Ehez szükséges még parancsokat létrehoznunk, hogy a karakter hogyan mozogjon, mit tegyen ha akadályba ütközik, pl ha magas blokk, akkor ugorjon, vagy ha fa van előtte kerülje ki. Nézzen körbe merre tud menni.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Tutor:Földesi Zoltán

A lisp programozási nyelv a mesterséges intelligenciát kutatók kedvelt nyelve, eredetileg nem ez volt a célja, de hamar az MI kutatásban lett használatos. Neve jelentése a listafeldolgozás, mivel a programok felépítése láncolt lista(zárojelekkel választjuk el a listákat). A lisp egy kifejezésorientált nyelv.

Nézzük meg elsőnek Iteratív módon.

```
(defun faktorialisi (n)
  (do
    ((i 1 (+ 1 i))
     (prod 1 (* i prod)))
    ((equal i (+ n 1)) prod)))
```

Az elején a defun-al adjuk meg a függvény nevét és a változót amibe majd az érték érkezik. A do egy konstrukció, ezt iteratív programknál használjuk mint ez is(iteratív struktúra). a (+ 1 i)-ben növeli az i értékét 1 el utánna végzi el a szorzást aztán növeli az n et, amíg szükséges.

Rekurzív módon:

```
(defun faktorialisr(n)
  (if (= n 1)
      1
      (* n (faktorialisr (- n 1))))))
```

Az előző program rekurzív változata. Az elején a defunnal adjuk meg a nevet és a paramétert. Aztán if el vizsgáljuk hogy n egyenlő e 1 el, ha egyenlő akkor meghívja önmagát n-1 re és összeszorozza az n el.

Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

```
FAKTORIALISR
Break 2 [6]> (faktorialisi 5)
120
Break 2 [6]> (faktorialisr 5)
120
Break 2 [6]> []
```

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Itt használom fel a második passzolási lehetőséget a SMNISTforHumansExp3 ban elért és közzé tett eredmény miatt.

DRAFT

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

A könyvben legelőször az alapfogalmakkal fogunk foglalkozni. Először is nézzük a programozási nyelveket. ezeknek 3 szintjét különböztetjük meg. Ez a három a gépi nyelv, assembly nyelv és a magasz szintű nyelv (C++, C ...stb). Minket a magasz szintű programozási nyelv fog érintetni. Az ebben írt programokat forrásprogramoknak nevezzük. Ezt a gépnek értelmeznie kell és feldolgoznia gépi nyelvre. Ezt egy fordító program segítségével jahtjuk végre, Ez először tárgyprogramot hoz létre majd ebből lesz a gépi nyelv. Az átfordításnak 4 lépése van. 1. lexikális elemzés. 2. szintaktikai elemzés. 3. szemantikai elemzés. 4.kód-generálás. Az első lépésnél a forrást lexikális egyégekre bontjuk. A 2 lépésnél a szintaktika helyességét vizsgálja, hogy megelel e a szintaktika szabályainak. Mivel, ha nem helyes a szintaktika, nem lehet gépi nyelvet eköállítani, ugyanis nem fogja megérteni. A 3 lépés a program megértése szemantikailag. A 4. lépés pedig legenerálni a kódot. Ezt kapja meg a vezérlő operációs rendszer. A másik technika az interpereteres technika, az első 3 lépés itt megegyezik a fordító programokkal, a különbség, hogy itt nem készül tárgyprogramot. Sorba veszi az utasításokat és sorban értelmezi azokat majd végrehajtja. minden nyelvnek saját hivatkozási nyelve van. Ebben van a nyelv szemantikai és szintaktikai szabáylai definiálva. A szemantikai részt emberi nyelven szokták megadni, a legelterjettebb az angol nyelv ezen a téren. A következő pontban a nyelvek osztályozásába kapunk betekintést. 2 fő osztályra bontjuk őket. Imperatív nyelvek és Deklaratív nyelvek. Az utóbbi algoritmikus nyelv, még az előbbi nem algoritmikus nyelv. A könyvben részletesen ki vannak fejtve. A következő a jelölés rendszer. A szintaktika formális leírásához alkalmazzuk ezeket. Vannak terminális és nem terminális jelölések. A jelölések után következnek a kifejezések. Ezek szintaktikai eszközök. Ezeknek van értéke és tipusa. A kifejezések formálisan operandusokból, operátorokból és kerek zárojelekből állnak. Az operadusok a változók vagy függvény meghívása lehet. Az operátorok pedig a műveletek amit az értékekkel elvégzünk. A záró jelek egybe foglalják és a sorrendet befolyásolják. Egy ilyen kifejezésnek 3 alakját tudjuk felírn: Prefix (pl: + 7 2), infix (pl: 7 + 2), postfix (pl: 7 2 +). Az infix alakban az operátorok nem azonos errőségűek. HA egyértelmű infix kifejezést akarunk létrehozni akkor teljesen zárolj jelezni kell azt. Ez felül írja a precedencia táblázatot. A könyvben továbbá a vizsgáljuk milyen nyelv, hogyan használja és hogyan értékeli ki a kifejezéseket. A C egy kifejezés orientált programozási nyelv. Típuskényszerítés elvét használja. A C precedencia táblázatát is megírjuk a könyv 51 oldalán. Például: [] ez a tömb operátor, . minősítő operátor, -> mutatóval minősítő operátor, ++ és -- értéknövelő és csökkentő operandusok, sizeof() operátor típus vagy kifejezés hosszát adja meg. Vannak a matematikai operátorok, mint a szorzás, osztás, összeadás, kivonás. Hasonlító operátorok mint az egyenlő,

nem egyenlő, nagyobb vagy egyenlő...stb. A logikai operátorok: "vagy", "kizárt vagy", "és". A könyvben ezekhez találunk leírást, hogy hogyan használjuk. Az adattípusokkal is foglalkozunk a 2.4 es pontban. Az adattípus megadja, hogy a változó milyen típusú, azaz minden típusnév egy azonosító. 3 dolog határoz meg egy adattípust: tartomyán, műveletek, reprezentáció. A programozási nyelvekben lehetőség van definiálni típusokat. Lehet létrehozni is de minden nyelvben vannak beépített típusok. Ilyen például az int azaz az egész típus, ennek vannak variánsai például shor vagy long int, a bool a logikai típus, float vagy a double amik lebegőpontos számokat tudnak tárolni. A könyvben megnézzük az egyszerű és összetett típusok. A könyvben nagy sújt kap a saját típus létrehozása. Szót kell még ejteni a tömbökről, ezeknek is van típusa ugyan úgy mint a változóknak, a tömbök olyan típusú változókat tartalmaznak amilyen típusú a tömb, ezen belül a tömb indexével tudunk mozogni. A mutató típust is átvesszük. A nevesített konstans egy olyan eszköz a programozásban aminek 3 része van: Név, Típus, Érték. Ezt a 3 at mindig deklarálni kell. Ugye a konstans jelentése állandó, tehát ez a programban mindig a deklarálásnál megadott nevet, értéket és típust fogja tartalmazni. Ezeket érdemes beszélő nevekkel ellátni, és olyan értékeket adni amiket sokszor használunk. C ben ezt az alábbi módon kell: "#define név literál ". A változó a konstant "testvére" úgymond. Neki 4 komponense van, Név, attribútum, cím, érték. A változó ahogy a neve is mondja nem állandó. A név az azonosítója. ezzel hivatkozunk rá a programban. Az attributok közül a legfontosabb hogy milyen típusú, a típusokról már fentebb beszélünk, amilyen típusú olyan értéket tud tárolni. És tudunk címet adni neki, azaz hol helyezkedjen el a tárolóban. de az elhelyezést a gép végzi, de hogy mikor hozza létre az attol függ hogy hol deklaráljuk . Nézzük meg egy példát: int a=5 . ez egy egész típusú változó kezdőértéke 5 a neve pedig "a". Most nézzük meg a C nyelv alapelemeit. Vannak integrált típusok (int, char...stb). Származtatott típusok (tömb, függvény, union, mutató). Ezek az aritmetikai típusok. A tömböt az alábbi módon hozzuk létre: int a[elemszám]. megadjuk a típusát a nevét és hogy hány elemű. Az utasítások a következő rész. Az utasítások alkotják a programot. Ezekből épül fel egy algoritmus, ciklus, és szinte minden, az utasításokat fordítja le a fordítóprogram tárgyprogramra. Kér nagyobb részre tudjuk bontani, deklarációs utasítás és végrehajtható utasítás. A deklarációs utasítás a fordítóprogram miatt vannak. tőle kérnek szolgáltatást vagy egy üzzemmódbba való lépést. Befolyásolja a tárgykódot de nem kerül lefordításra. A végrehajtható utasítás, ahogy a nevében is benne van, bővégre lehet hajtani. Ezekből lesz generálva a tárgykód. Ezeket tudjuk csoportosítani például értékeadó utasítás, üres utasítás, ugró utasítás, elágazó utasítás, hívó utasítás, I/O utasítások ...stb. Ezekről a könyvben részletes leírást kapunk. Például az értékeadó utasításban értéket adunk vagy modosítunk egy változón vagy több változón. Az elágazó utasítások is szinte kihagyhatatlanok egy programban, ugye ezek az if és az else if feltételes utasítások. Ezek lehetővé teszik a programban a több irányú elágazást. A ciklusszervező utasításokat is ismerjük már ha nem kezdők vagyunk. Ugye ezek a for, while, do while ciklusok. A végtelen ciklus volt az első feladatunk, azt már ismerjük, vannak feltételes ciklusok, ugye itt addig fut a ciklus még a feltétel igaz. Van kezdő feltételes ls végfeltételes ciklus. Ezeknek a működését a könyvben megismertük. Van az előírt lépésszámú ciklus (a for()). Annyiszor fut le a ciklus, ahányszor előírtuk neki. Vannak összetett ciklusok Ez az előzőök kombinációja. A működésük nagyon bonyolult. 3 vezérlő utasítás van C ben. 1. Return: Ez szabályosan befejezzi a függvényt és vissza adja azt a vezérlést hívónak, legtöbbször értékkal tér vissza. 2. BREAK: ezt a cikluson belül alkalmazzuk, ha életbe lép akkor kilép a ciklusból és az utána lévő cikluson belüli utasításokat nem hajtja végre. Ezt iffel szoktuk alkalmazni. 3. Continue: ez is a ciklus magban van. ez is kilép, vagy újabb cikluslépésbe kezd, vagy megvizsgálja újra az ismétlődés feltételeit. Az 5. fejezetben a programok szerkezetét ismerjük meg, hogy milyen részekből áll. Itt ismerjük meg az alprogram feladata egy bemeneti adatcsoport leképzése vagy kimeneti adatcsoportot készít le, egy megadott specifikáció szerint. Ebben a fejezetben ismerjük meg a blokkokat. a blokk egy programegység, ami utasításokat foglal magában. A kezdetét és végét speciális karakter jelzi. A blokk bárhol elhelyezhető a programban. A 13. I/O fogjuk venni. Ez egy olyan rész ahol a program nyelvek nagyban eltérnek egymástól. Állimányuk a funkciók szerint 3 lehet. lehet input, output és input-output állomány. Az I/O során a programban az adatok a tár és a periféria kö-

zött mozognak. Ezeknek van egy bizonyos ébrázolási módja. Az adatátvitelnek 3 fajtáját alkalmazzuk: formátumos módú, szerkesztett modú és listázott modú adatátvietl. Ha állományokat alkalmazunk azt a következpképpen kell csinálnunk. 1. deklaráció, 2. összerendelés, 3. állomány megnyitása, 4. feldolgozás, 5. lezárás. A C-nyelvnek nem része az I/O erre egy standart könyvtárat használunk.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

A könyvben a C nyelv alapjaival fogunk megismерkedni az alapoktól. A könyv programok segítségével segít elsajátítani a C nyelv ismeretét. Ha eddig egyáltalán nem találkoztunk a C nyelvel, a könyv akkor is nagyon hasznos lehet, hiszen nagyon részletesen, mindenre odafigyelve adja át a tudást. Először kezdjük a változók típusaival és a program beli alkalmazásával. A programban különböző típusú ezeken belül különböző méretű változók vannak. Típusok: int -intiger, azaz egész típusú számot deklarálunk ezzel, mérete a gép egészétől függ. char -Ez egy karaktert tud tárolni, mérete 1 bájt. float - egyszeresen pontos lebegőpontos számot tud tárolni (törtet). double- kétszeresen pontos lebegőpontos számot tud tárolni, ezt hatalmas számokkal való számolásnál használjuk. Az int típustnak vannak fajtái még, ilyen a short int vagy a long int, van továbbá még unsigned int amivel a negatív számok helyett, csak pozitív értékeket tudunk tárolni, de abból többet mint a sima int-nél. Továbbá ami még lényeges lehet, az lokális és globális változó, ahogy a nevük is jelzi, a globális az egész programban használható, még a lokális csak a megadott helyen. A könyvben a helyes alkalmazásról rengeteg példát láthatunk. Vannak az állandók, ezeknek megadunk egy értéket, és ez az érték állandó lesz a programban, nem változik. A könyv következő fejezetében a Vezérlési szerkezetet vesszük. Ez határozza meg, hogy milyen sorrendben hajtódnak végre a műveletek. Először az utasításokat és a blokkokat nézzük. Fontos szabály, hogy minden utasítást ";" -al zárunk le. Így válik egy sor utasítássá. Ha az utasítások {}-el vannak körbe zárva, akkor azt egy bloknak nevezzük és 1 db utasításlént kezeljük. A blokkok közé soroljuk a ciklusokat és a az elágazásokat. Először az if utasítással ismerkedünk meg. Ha a megadott feltétel igaz, akkor a megadott utasítások fognak lefutni, ha nem igaz akkor az utasításokat figyelmenkívül hagyja. Ennek a bővítése az else-if. Itt meg tudjuk adni, hogy a program milyen utasításokat hajtson végre, ha a feltétel nem igaz. A következő a switch utasítás. A lényege hogy megadunk neki mintákat, és ha a vizsgált elem megegyezik valamelyik mintával, a mintánál megadott utasítások fognak lefutni. ezek után vegyük a ciklusókat. A ciklus egy olyan blok ami a feltételig újra és újra lefut. Kezdjük a while ciklussal, a while jelentése amíg, tehát amíg a kifejezés igaz addig lefut a ciklus, itt az lesz fontos, hogy először vizsgálja meg , hogy a feltétel igaz e, aztán fog lefutni az utasítás vagy utasítások. Ezzel megegyezik a for utasítás, a lényege és működési elve ugyan ez. A felépítése különbözik. A do while ciklus először végrehajtja az utasítást és aztán vizsgálja meg a feltételt, ha nem igaz akkor kilép a ciklusból. A break utasítással a futás közben is kitudunk lépni egy ciklusból ezt a ciklusokban lévő if-ekben szoktok alkalmazni. A goto -val pedig egy adott címclére ugorhatunk. De ezt nem sűrűn alkalmazzák, sokan nem is szeretik. Rendezni tudjuk ezeket a futási sorrend alapján. Van a sima ami sorban fut le, de van a címkezett usasítás blokk az előbb említett goto-val, itt az adott feltételektől függ, milyen sorrendben fut le a program. A Függelékbén az utasításoknál csoporthozosítjuk az utasítások fajtáit. A címzett utasításokhoz előtagként megadott címke kapcsolódik. A kifejezésutasítás, a nevéből adódoan kifejezésekkel épül fel. Az összetett utasítás megszünteti a korlátozást ahol a fordító csak egyetlen utasítást fogad el. A kiválasztó utasítások, a lehetséges esetek közül választ a feltételnek megfelelően (if, switch). A vezérlés átadó utasítások, amár fentiekben említett goto, continuem break, return parancsokat alkalmazza. Az Iterációs utasítások a ciklusokat alkalmazzák (while, do, for).

10.3. Programozás

[BMECPP]

A könyv témája a szoftverfejlesztés C++-ban. A legelső fejezetben azokat az új dolgokat fogjuk megnézni amiket C-ben nem tudtunk de C++ ban már lehetséges. Elsőre függvényparaméterek és visszatérési értékek vesszük. A C ben üres paraméterlistával definiált függvényt itt C++ ban void paraméter megadásával oldjuk meg.A visszatérési típusnál is eltér a 2 nyelv. Míg a C nyelvél int, addig C++ nem támpatja az alapmérétezett típust. Az int main fő függvénynek is 2 típusa van C++ -ban. Az első mikor nem adunk meg semmit, a másik mikor argumentumokat adunk a mainnak a parancssorból. A C++ nyelvben jelent meg először a bool azaz a logikai típus aminek értéke true vagy false. A feladatok között van egy ahol az volt a hiba, mikor utasításon belül deklaráltunk. C++ ban már ez is lehetséges. A példa erre, hogy az i értéket a forcikluson belül deklaráljuk: "for(int i=0; i kisebb mint 10; i++)". Hasznos lehet, ha a deklarációt a felhasználás előtt végezzük. C++ ban a függvényeket a nevük és a megadott argumentumokkal azonosítjuk. Azonos nevű csak akkor lehet, ha más az argumentum lista. C-ben a paraméterátadás érték szerint történik. A megkapott érték klonózva lesz és a másolatra fogunk hivatkozni, azaz a visszatérési érték nem befolyásolja a programunkat. Ez az adott példában a könyv jól szemlélteti. Ennek C++ ban a megoldása hogy az eredeti változót fogjuk átadni mégpedig a memória címével. Ezt a referencia jellel fogjuk teljesíteni. Így a változtatások, az eredeti változón játódnak végbe, és az új értéket kapja vissza a program. Ez a referencia C-ben nem létezik. A harmadik fejezetben megírjuk az Objektumokat és osztályokat. A C++ nyelv egy objektumorientált nyelv, ennek alapelveivel megírunk. A lényege hogy a függvényeket osztályokba foglaljuk az osztályok példányait nevezük objektumoknak. (objektumokkal hívjuk meg az osztályok függvényeit.) A 3.2 bekezdésben egy példán keresztül láthatjuk, az egységbázárás előnyeit. A struktúráknak így lesznek tagváltozói, és tagfüggvényei is. A tagfüggvényeket kétféle képpen adhatjuk meg. Osztálydefinícióban vagy struktúradefiníció kívül . Fontos még az adatrejtésről is beszélünk. Így csak az osztályon belül férhetünk hozzá, külső osztály nem férhet hozzá, az értéket függvényel tudjuk kiadni. függvény és változó is lehet ilyen, ezeket a private részbe írjuk, azaz ezeket privátá tesszük. A konstruktur onnan ismerjük fel, hogy ugyan az a neve mint az osztálynak és nem adunk típust neki, ez akkor fut le, ha meghívjuk az adott osztályt amiben benne van, egy objektummal, ezt gáran inizializálásra használjuk. A destruktor annyiban különbözik szintaktikailag, hogy a név előtt egy '~' jel található, és automatikusan meghívódik egy objektum befejeződésekor, ezt a memória felszabadításra szoktuk használni főként. A C-ben a dinamikus memória foglalás malloc-al vagy a free kulcsszavakkal történik. A C++ ban már operátorral tudunk lefoglalni memóriát, ez a new operátor, példa: "int *pelda; pelda= new int;" később ezt a változót fel kell majd szabadítanunk, vagy memóriászivágás lép fel. A dinamikus adattámagatás miatt szükséges sokszor, hogy megadjuk a mutató állapotát. Ez a NULL kulcsszó. Ha az állapot NULL akkor nincsen lefoglalt adat, ha ezt nem adjuk meg akkor a new szócskával lefoglalt területre mutat. A 3.5.3 fejezetben ismerkedünk meg a másoló konstruktőrral. A másolókonstruktur egy referenciát kap, amely megegyezik az osztálynak a típusával. A másolókonstruktur is rendelkezik mindenkel, amivel egy egyszerű konstruktur. Ugyan úgy tudunk inicIALIZálni vele objektumokat. Amiben viszont több hogy érték szerint adunk egy függvényparamétert neki, akkor a megadott változó lemásolódik, és ezt használjuk a függvény törzsében. Az osztályainak lehetnek friend függvényei vagy osztályai. Ez azt jelenti, hogy jogot adunk egy függvénynek vagy egy osztálynak, hogy hozzá férjen az adott osztály védett, azaz private és protected változóihoz és függvényeihez. Ezt a friend kulcsszóval tesszük lehetővé. A feljogosítandó függvény vagy osztály elé írjuk és ezzel fel is jogosítjuk. A tagváltozóknál az értékadás és az inicializálás nem ugyan az. Az inicializálás azaz alaphelyzetbe álltjuk. A létrehozásnál megadunk neki egy alap értéket. Az értékadás az az "=" jelel történik a programtol függően, ez a programon belül bárhol megtörténhet ahol a szabályok engedik. Említenünk kell a statikus tagokat, ezek a tagok az osztályhoz tartoznak és nem az osztályob-

jektumhoz. Továbbá lehetőségünk van struktúráknak, osztályoknak típusdefinícióra, a `typedef` szóval. A hatodik fejezetben vesszük az operátorokat és az operátortérhelést. A C nyelvben az operátorok műveleteket végeznek az argumentumokon. Az operátorok kiértékelési sorrendjét egy speciális szabályrendszer határozza meg, ezeket ugyan úgy mint matematikában, zárójelekkel írhatunk felül. A C++ rendelkezik új operátorokkal a C -hez képest. Ilyen a hatókör operátor aminek jelölése a `:::` ezt az osztályok hatólörének megadásánál használjuk. Ismerjük már a `*` operátor, ugye ezzel jelöljük a mutatókat. A `->` operátorral pedig mutató esetén hivatkozunk. Fontos különbség a C és a C++ között a függvénymellékhatása, A C nem képes erre, de a C++ igen. Ráadásul mivel az operátor speciális függvény ezért különböző argumentumok esetén túl tudjuk őket terhelni. A 10-es fejezet a kivitelezésről szól. Ilyen a hagyományos hibák kezelése. De mi is az a kivitelezés? A kivitelezés egy mechanizmus, amely ha hibát fedez fel, akkor a hibakezelő ágra ugrik. Ilyen a `try_catch` blokk. A `throw` kulcsal küldünk egy kidobást. A `try-catch` pedig elkapja, ha egy `catch` ág megegyezik az elkapott típussal. Ha nem kapja el, azt kezeletlen kivitelnek nevezzük. Van egymásba ágyazott `try-catch` blokk, így tudjuk a kidobásokat külön szinten kezelní

DRAFT

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Berners-Lee!

11.1. Bevezetés a mobilprogramozásba.

Az alábbi szövegben a Python programozási nyelv fogunk foglalkozni a megadott könyv alapján. Kezdetben a nyelv jellemzőiről olvashatunk. A Python nyelv, más programozási nyelvekkel (C++, Java, C) ellentétbe elég csak a forrást megadni, ugyanis a fordítási fázisra itt nincs szükség. A Python használható a neves platformokon, mint például Unix, Windows, MacOS ...stb. A nyelv alkalmas prototípus alkalmazások elkészítésére, hiszen sokkal kevesebb erőfelhasználással lehet benne dolgozni mint például a C++-ban vagy a Javaban. A Python nyelv egy magas szintű programozási nyelv, mégis egyszerűsége hasinlítható az awk vagy Perl nyelvekhez. A nyelvben a Python kódkönyvtárat használjuk. Az ebben lévő modulok, gyorsabbá teszik a programok fejlesztését. A modulok használhatóak rendszerhívásokra, hálózatkeresésre és fájkezelésre is. A nyelv könnyen olvasható alkalmazást készíthetünk. Ennek oka az, hogy az adattípusok engedik, hogy összetett kifejezéseket röviden tudunk leírni. Továbbá a más nyelvekkel ellentétbe a kódnak a csoportosításának tagolása tabulátorral vagy új sorral történik és nem kell definiálnunk a változókat vagy az argumentumokat. A Python nyelv szyntaxa behúzás alapú. Nem szükséges kapcsos zárójel vagy kulcsszavak használata. Egy blokk végét egy behúzással végezzük, ez lehet akár üres sor. Behúzással nem lehet kezdeni egy szkriptet. minden utasítás a sor végéig tart, így nem szükséges a C,C++ vagy a Java-ban ismert ";" használatára. Ha túl hosszú lenne egy sor, akkor "\n" jellel lehet ezt jelölni. Egy behúzás nem érvényes a folytatósorokra. A sorokat a nyelvben tokenekre bontja a nyelv, ezek között lehet tételezés ürek karakter. A token fajták: azonosító, operátor, kulcsszó ...stb. A nyelvben megkülönböztetjük a kis és nagy betűket. Pythonban objektumokkal reprezentálunk minden adatot és az ehez kapcsolatos műveleteket az objektum határoz meg. Az adattípusok hasonlóan a többi nyelvhez itt is lehetnek, sztringek, számok, ennesek, szótárak és listák. A szám típuson belül lehet egész szám, ami lehet lebegő pontos és komplex szám ezen belül is decimális, oktális vagy hexadecimális. A lebegőpontos szám a C++ ban ismert double-nek felel meg. A pythonba megtalálható a szekvencia is, ez egy nem negatív egész számokkal indexelt gyűjtő. A sztringet két féleképpen lehet megadni: idézőjelek között "példa" vagy aposztrófok között 'példa'. Az ennes tipusok vesszővel elválasztott gyűjteményei az objektumoknak. A lista lehet több különböző típusú elemkből. Az elemeket szöglletes zárójelek között kell felsorolni. A szótár pedig kulcsokkal azonosított rendezetlen halmaza az elemeknek. A változók Pythonba objektumokra mutató ferefenciák. A változóknak itt nincs típusa, így különböző típusú objektumokra lehet hívatkozni. A változóknak a "=" egyenlőség jellel lehet értéket adni. A "del" kulcsszóval törölhetünk egy változó hozzárendelést. A nyelvben két féle változót tudunk megadni, van a globális és a lokális. A globális változót úgy tudjuk létrehozni, hogy a "global", ezzel jelezve, hogy egy globális válzotó lesz. Továbbá fontos, hogy a globális változókat a függvény elején kell

felvenni. minden más , a függvényben létrehozott változó alapmérétezetten lokális lesz. A Python rövidre zárt kiértékelést hajt végre, ezt magyarázzuk egy példával: `a<b and b<=c and c==d` kifejezést a Pythonba így tudjuk írni: `a<b<=c==d` . A beépített típusaink között lehetőség van a típus konverzióra (ilyen az `int,float,long is`). Képesek vagyunk sztringből is számot képezni, ezt a használt számrendszer megadásával tudjuk. Pl `int('11',8)` . Ennek az értéke a 9 lesz. Sokféle műveletet tudunk a szekvenciákon végrehajtani, továbbá beépített függvényeket is alkalmazhatunk rajtuk. Ilyen függvényekkel már találkozhattunk számtalanszor más programozási nyelvekben. Ilyenek a max és min függvények, amivel szélsőértéket tudunk meghatározni, vagy a len() függvény amivel mekapjuk egy szekvencia hosszát. A szekvenciákat '+' -al tudjuk össze fűzni. Tudunk kifejezéseket is alkalmazni rajtuk, ezek az 'in' és a 'not in' , ezekkel tudjuk megnézni hogy eleme e vagy nem eleme a szekvenciának. Az elemeket indexekkel látjuk el, ezek alapján fogjuk majd elérni, ezt 0 tól szoktuk indítani. Van a negatív indexelés, ez a szekvencia végéről kezdve vissza felé halad. Intervallumot a ':' jellel tudunk megadni. Például ha `a=[5,6,7,8,9]` akkor ha `b=a[1:2]` az eredmény az 6,7 értékek lesznek. A könyen láthatjuk a további műveleteket mint az ismert `pop([j])` függvényt, ezzel eltávolítjuk a 'j'-ik elemet, ha nem adunk meg elemet, akkor az utolsó elem kerül törlésre. A reverse() függvény megfodítja a sorrendjét az elemeknek. append() függvénytel tudjuk bővíteni egy lista végét, míg az extend() függvényel egy másik lista elemeit fűzzük egy lista végéhez. A clear() pedig töröljük az egész listát. Most pedig következzenek a nyelv eszközei, ezek közül nézzük meg a print metódust, ezzel a metódussal változókat vagy egy sztringet írhatunk ki konzolra. A python nyelvben is van lehetőségünk az elágazásokra. Ezeket csak úgy mint más programozási nyelvekben if, elif és else kulcsszavakkal hívjuk meg. A szintaktika is hasonló: Első a kulcszó, azután a feltétel és végül pedig hogy mi történjen ha a feltétel teljesül. A másik nagyon fontos nyelvi eszközök a ciklusok. A programozásban az egyik legfontosabb és leghasznosabb metódus a ciklus, ezzel képesek vagyunk adatokat bejárni, feltölteni, megvizsgálni, keresni és még számtalan hasznos dolog. Elsőnek nézzük a for ciklust. A for ciklussal akár szótárakban, tömbökben, vektorokon is végig mehetünk, ha a cikust az előbbiek elemire definiáljuk. Fontos még szót ejteni a range függvényről. Ez a ciklus futása közben listát generál egész típusú értékekből. A következő ciklusunk a while ciklus, ez a feltételes ciklus, ugyanis a ciklus addig fut, amíg a megadott feltétel teljesül. Bónusz még, hogy a Python a break és continue kulcsszavakat is ismeri és támogatja. Használhatunk címkéket és ugráskat is. A címkék kulcsszava a label, ezzel tudunk a programban címkéket elhelyezni, amikhez a goto parancsal ugorhatunk. A Python programozási nyelvben is léteznek a függvények. Ezeket pedig a def kulcsszóval definiálhatjuk. A függvényt hasonlíthatjuk egy értékhez, hiszem tovább lehet adni és más függvény is megkaphatja.A függvények vannak paraméterei, ezeket számunkra megfelelő megkötésekkel szintaxisossal adhatunk meg. Fontos dolog, hogy a paraméterek az érték alapján adódnak át, kivételt képez a mutable típusú érték. Az argumentumokat a függvény hívásánál tudjuk megadni. A függvények rendelkeznek egy visszatérési értékkal, de ennesekkel is visszatérhet. Az Osztályokat és objektumokat is támogatja a nyelv. Ez azt jelenti, hogy tudunk osztályokat definiálni, ezeknek az objektumok lesznek a példányai. Az osztály attribútumai: függvények és objektumok. Az osztály lépes örökölni és örököltetni más osztályokból/nak. Az osztályt a "class" kulcsszóval definiáljuk. A ősosztályok definíciója az, hogy: már definiált osztályok, opcionális listái vesszővel elválasztva. Attributumokat tudjuk bővíteni az osztályban és a példányokban is. Az attribútumokat az osztály törzsén belül határozzuk meg. Az osztály metódusai hasonlóak mint a globális függvények, annyi különbséggel, hogy itt az első paraméternek kötelezően a selfnek kell lennie, ennek az értéke minden az adott objektumpéldány lesz, melyen a függvényt meghívíták. Az `__init__` egy speciális konstruktor metódus. A nyelv tartalmaz modulokat a könyebb fejlesztés érdekében. Ilyen például az appuifw, a messaging (SMS, MMS), camera, audio vagy a sysinfo modul. A váratlan helyzetekre itt is tudunk alkalmazni kivitelezést. Az adott kódot a try blokkban megadjuk, ezután egy expect jön, ami hiba esetén fog életbelépni. Végezetül pedig essen pár szó a PYS60 grafikus felületről, hiszen minden komolyabb program rendelkezik grafikus felüallettel. A GUI szolgál az alkalmazáson belül az infomációk elrejtezéséről, megjelenítéséről. De ezen felül még kezeli is a felhasználói interakciókat.

11.2. Java és C++

A következőbe a C++ és a Java programozási nyelvet fogjuk összehasonlítani a könyvben megadottak alapján. Alapjában mind a két nyelv egy magas szintű programozási nyelv. A Java nagyban hasonlít a C++ nyelvre, mivel sok minden átvett attól, ilyenek például a jelölések. A forrásszöveget olvasva látható, hogy a Java nyelv szyntaxa a C,C++ nyelvből fejlődött. Ahogy az előbb említettem, mind a két nyelv objektumorientált, de még a Java nyelv teljesen objektumorientált nyelv, addig a C++ egy többelvű programozási nyelv (azaz több programizási módszert támogat). Az objektum orientált nyelv az mikor a programban az adatokat és műveleteket objektumokban foglaljuk egybe. A programozási paradigma, a programozási mód. A program feléítésére használt eszközökészlet, tehát, hogy milyen egységek képzik a program alkotóelemeit. Ezek közül fogjuk megvizsgálni az Osztályokat, objektumokat és a példányosítást.

A Java teljesen osztályokból épül fel, de úgyan úgy tudjuk a C++ ban is használni az osztályokat. Egy osztályt a "Class" kulcsszóval hozunk létre. A Java-ban a legkisebb önálló elem az osztály. Javaban és a C++ ban is egy osztály az azonos típusú "dolgok" modelljét írja le. Egy adott program működése során fogja példányosítani az osztályokat, azaz konkrét példányokat hoz létre, ezek lesznek az objektumok. Java-ban az osztályok 2 részből állnak. Az első részben azokat a változókat deklaráljuk, amikkel az objektumunk állapotát le tudjuk írni. A második részben találhatóak az osztály metódusai. minden metódus meghívásakor, muszály megadni, hogy az osztály melyik példányára vonatkozóan hívjuk. C++ ban is egy osztálynak két fő része van, az első rész tartalmazza a z osztály attributumait, míg a második pedig ugyan csak a metódusok. Monst nézzünk egy példát Javaban és C++ ban lévő osztályt:

```
//java
public class Pelda{
    String szoveg;
    int szam;
    void peldametodus(int parameter) {
        szam+= parameter
    }
}

//C++
Class Pelda{
    public:

    void Hello() {
        cout<<"Hello";
    }
}
```

A hozzáférési jogokat Javaban minden a változónál vagy a metódusnál adjuk meg, míg C++ ban, ahogy a példában is látszik, hogy az Osztálynak lehet egy public része, protected és private része. A public esetén nyílvános lesz az elem, mindenki számára látható. A private akkor csak az adott osztály láthatja, ha pedig protected akkor a leszármazott. Ha nem adunk meg semmit, csak a foráásban lesz látható. A java támogatja a félnyilvános tagokat is, tehát ha egy tag jelöletlen, akkor nem lehet hivatkozni bármely osztállyal. Egy osztály egy másik osztállyal az extends kulcsszóval tud örökölni.

Egy objektum Példányosítással fog létrejönni, Javaban (és C++ ban is) a new kulcsszóval történik. Nézzünk egy példát az objektum létrehozásánál:

```
Peldaosztaly a=nev Peldaosztaly()
```

A new operátor után megadjuk, hogy melyik osztályt akarjuk példányosítani. A paramétereknél az osztály konstruktornak szánt paramétereket adjuk meg. A konstruktur az osztály egy speciális függvénye, onnan ismerhetjük fel, hogy ugyan az a neve mint az osztálynak, ezzel az osztály inicializálását hajtjuk végre. A példányosítás amiről már fent említettünk szót, az annyi, hogy ugye a new operátor segítségével, a memoriát foglalunk le, és az objektumnak a változóit ott tároljuk. Itt a tárterületek kezdőcímét kapjuk vissza. Egy objektumon belüli elemre úgy lehet hivatkozni, hogy az objektum nevét és az elem nevét egy pontal össze kapcsoljuk. Ha egy elem több részes, akkor ugyan így kell hivatkozni rá. Ha egy elem elő írjuk, hogy "static", akkor az az elem vagy elemek nem egy objektumhoz fognak tartozni, hanem az osztályhoz.

Egy osztálbyan a metódusokat az osztályhoz hasonlóan a "class" kulcsszavakkal létrehozott szerkezeten belül kell megadni. A metodus deklarációja módosítokkal adható meg, ilyen a visszatérési érték, a metódus paraméterei, a metódus neve és a metódus törzsének a leírása. Nézzünk egy példát egy metódusra és annak meghívására:

```
System.out.println(pelda.toString());
```

Ez a metódus egy sztringet ad vissza az objektumból. A metódusneveket túl lehetett terhelni. Azaz egy osztály több metodusát el lehet nevezni ugyan azon a néven, de csak akkor ha a szignatúrájuk nem azonos, tehát különböző. Ezt az takarja, hogy a formális paramétereik száma nem egyezik. A java fordító ezek alapján tudja, hogy melyik metódust kell meghívnia. Az osztályhierarchia az osztályok rokonsági viszonyainak összessége. Gyakran egy lefelénövekvő fával ábrázoljuk, az Object osztályból indulva. Ez egy kiemelt osztály, ami a java.lang csomagba tartozik. Az object azoknak az osztályoknak a szülője, akik nincsenek más osztályokból származtatva. Ez azt jelenti, hogy nincs megadva az extends kulcsszó. Most pedig említsünk pár szót az absztrakt osztályokról. A nyelvben az abstact modosító kulcsszóval jelöljük az absztrakt osztályokat. Egy absztrakt osztály annyiban különbözik a sima osztályuktól, hogy tartalmazhatnak absztrakt, törzs nélküli metódusokat. Egy absztrakt osztályt nem lehetséges példásosítani. Nézzünk egy példát egy absztrakt osztályra:

```
public abstract class Pelda{  
    private double kerulet;  
    protected abstract double keruletszamitas();  
    public double kerulet(){  
        if (kerulet==0) kerulet= keruletszamitas();  
        return kerulet;  
    }  
}
```

Most pedig nézzük a karakterkészletet, utasításokat, kifejezéseket. A Java nyelv és a C++ is rendelkezik azonosítókkal, Javaban és C++ ban is egy betűvel kell kezdődni egy azonosítónak, de utánna betű és szám is következhet, például "kabat" jó még a "2kabat" nem jó azonosító. Vannak kulcsszavak, ezek nem lehetnek azonosítók. Ilyen kulcsszavak például a változó típusok mint például az int, long, vagy a ciklusok kulcsszavai mint a for: for, do, while. Az utasítást vegyük most elő. Azt utasításoknak két alapvető fajtája van. ezek a kifejezés-utasítások és a deklaráció utasítások. A kifejezés-utasítás csak az alábbi fajtából képezhető. 1.értékkadás (= operátorral). 2.postfix, prefix ++ és -- operátorokkal képzett kifejezés. 3. metódushívás. 4. példányosítás. Míg a másik lokális változók létrehozását, inicializálását jelenti. Az utasításokat minden blokkokba tesszük Javaban is és C++ ban is. Javaban minden válzotó még a globális

változó is blokkban lesz (fő osztály). még C++ ban a globális változó blokkon kívül esik. Nézzünk példát utasításra ami a blokkon belül van:

```
{  
    int j2;  
    j2=10;  
    int b= j2+j2;  
}
```

Térjünk vissza a metódusokhoz: Egy metódus úgy épül fel, hogy elsőnek a különböző módosítok, a visszatérési értéke majd a neve , utánna a paraméterek következik. Ezután a blokkon belül a metódus törzsének leírása van. Most nézzük meg, hogy kódban hogy néz ki egy osztálynak a metódusa:

```
Class Pelda  
{  
    // valtozok  
  
    public int szamketszer(){  
        return szam*2  
    }  
}
```

A metódusok meghívását már fent megbeszéltük. Javaban és C++ is tól tudjuk terhelni a metodusaink nevét. Egy osztálynak több metódusát is elnevezhetjük ugyan úgy, de ebben az esetben a kapott paramétereknek kell különbözőnek lenni vagy a típusnak. Ez esetben a paraméterek és a típus szerint a Java fordító dönti el, hogy melyik metódust kell alkalmaznia.

Most pedig beszéljünk kicsit a konstruktőről újra és a destruktőről is. A konstruktur egy olyan program kód, ami autómatikusan lefut a peldányosítás során. Ahogy már említettem, a nevének azonosnak kell lenni az osztály nevével. Képesek paramétert fogadni és csak példányosításon keresztül hívhatók meg. Visszatérési típus egyáltalán nincs, még a void sem. Ennek úgymond az ellentetje a destruktur, ami szemétfelületű mechanizmusnak köszönhető már sok esetben nem szükség. C++ ban ez még kell. C++ ban a destruktort ~ al helöljük, még javaban ha szükség lenne mégis destruktur jellegű metódusra azt a Finalize kulcsszóval jelezzük.

Most rátérünk az interfészekre. Az interfész már az Objective-C nyelvben is jelen volt. Javaban az interfész egy új referencia típus, az absztrakt metódusok deklarációjának és konstans értékének az összesége. Magyarul a lényege annyi, hogy az osztályoktól annyival különbözik, hogy nincs változtatható, valódi tulajdonsága az adattagok éd metódusok implementációi. Fontos és alapvető tulajdonsága az interféseknek, hogy bennük a metódusok csak deklarálódnak, tehát megvaósítás nélkül szerepelnek, tehát csak ehy felületet definiál. Egy interfészr egy osztály implementálha minden interfész álltal specifikált metódushoz ad implementációt. A java.lang csomagban, a Runnable definiált interfészr implementálni kell olyan esetben, amikor azt szeretnénk elérni, hogy az osztály példányai több külön szalon fussenek. Az interfések között is lehetséges az öröklődés, ezt itt is kiterjesztésnek nevezzük, mint az osztályoknál, de 2 féle dologban mégis eltér azoktól: egy interfésznek több ősinterfésze lehet és nincs közös ősinterfész. Az interfések közötti öröklődés például a Collection, List. Most pedig nézzünk egy példát egy interfészre:

```
public interface Comparable<T>{  
    public int compareTo( T o)  
}
```

Interfészket tudunk létrehozni, ezt interfész deklarációval tudjuk elérni. A deklaráció az úgy néz ki, hogy az interface kulcsszóval kezdünk, aztán jön az interfésznek a neve, és végül az interfésznek a törzse:

```
interface edes{ }  
public interface{  
    int iz=1  
    void Joetvagyat();  
}
```

Az adatfolyamok kezelése a következő pontutnkn. azon belül is a bemenet és a kimenet azaz az I/O -t fogjuk most megnézni. C++-ban is és Javaban is az első programunk egy kiíratással történt. Java-ban a System.out objektum println eleme kell egy szöveg kiíratásához, míg C++ -ban a cout függvényel történik. Ha konzolról akarunk beolvasni, akkor Javaban Scannert kell létrehozni, még C++ -ban a cin függvényel olvasunk. A fájlba íratás már kicsit nehezebb, a javaban, mert míg a C++ ban ofstream-mel ki lehet íratni fájlba, Javában ezt 3 sorban kell megoldanunk. Létre kell hoznunk egy kimeneti objektumot, ami le repezentálja a kimeneti txt. majd out.println objektumot haszálunk a kiíratáshoz. Majd egy FileWriter-t használva adjuk meg, hogy melyik fájlba szeretnénk kiíratni. Javaban az adatfolyamok is objektumoknak felelnek meg, és az összes adafolyamtípus egy adott osztályhoz tartoznak. Ezekből pedig számtalan típus található. Az adatfolyamokat 3 féleképpen osztályozzuk. Az adatfolyam irányá szerint lehet bemeneti és kimeneti, Az adatfolyamokat alkotó adatok típusai szerint lehet bájt vagy karakterszervezésű adatfolyamok szerint. A 3. pedig a feladatuk alapján. A hierarchia legalján az Input/OutputStream, a Reader és a Writer absztrakt osztályok vannak. Az adatfolyamok kezelése az adatfolyamok megnyitása és lezárása, Vannak kírő műveletek, olvasó műveletek, rendező műveletek. Még egy jól megírt program futása közben is léphet fel hiba, Javaban és C++ ban is erre a megoldás a Kivitelezés. A kivitelezés céja, a program futása közben fellépő hibák kezelése. Ezt hívjuk exceptionnak. ez megszakítja a program futását a hiba esetén. A kivételkezelést az throw, vagy a trx-catch-finally szerkezzel tudjuk kivitelezni.

Végezetül pedig ejtsünk pár szót a multiparadigmás nyelvekről. A java nem multiparadigmális nyelv még a C++ az. A multiparadigmás nyelvek a többnyelvű programozási nyelvek, ami alatt egy olyan nyelvet értünk ami több programozási módszert is támogat. A c++ támogatja az imperatív, az objektumorientált és a generikus stílust.

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

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

A feladatot kezdjük először azzal, hogy mit csinál a program. A program le fog generálni 10 számot véletlenszerűen. Ezt polártranszformációval fogjuk elérni. A polártranszformációt használja a Java szám-generátora a véletlen szerű számok generálására.

Nézzük a kódot:

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

Az első sorban létrehozzuk a púlikus osztályunkat ami a Polárgemerátor nevet kapja. Ebben az osztályban két változót fogunk deklarálni. Az első egy boolean logikai változó, a másik egy double típusú változó. A logikai változó neve "nincsTarolt", ezzel azt fogjuk nézni, hogy van-e eltárolt változó, ezt a program elején True-val adjuk meg, mert az elején nem tárol változót. A "tarolt" változóval pedig a kiszámolt változónak az értékét fogjuk eltárolni.

```
public PolárGenerátor() {
    nincsTárolt = true;
}
```

Ez az osztály konstruktora, ezt onnan tudjuk legkönyebben hogy, ugyan az a neve mint az osztálynak. Ezzel végezzük a "nincsTarolt" változó inicializálását. Itt kap True értéket.

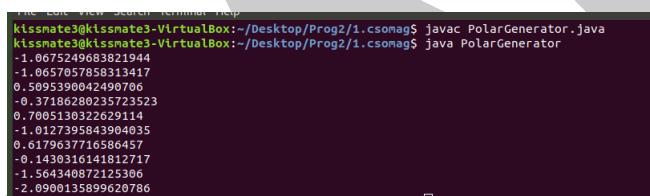
```
public double következő()
```

Ez egy metódus. Itt történik a matematikai számítás. Először egy if elágazással megnézzük, hogy van e tárolt elem, vagy nincs. Ha nincs tárolt elem (azaz a "nincsTarolt" értéke igaz.), akkor elvégzi a számításokat. Két darab értéket fogunk kapni, az első értéket vissza adjuk return -el, míg a másikat eltároljuk a double típusú "tarolt" változóba. Ha van tárolva érték, akkor rálépünk az else ágra. Abban az esetben az eltárolt értéket fogjuk vissza adni.

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

A program végén pedig lássuk a main függvényt. Ez a fő függvény, ez fog lefutni a legelőször. A "new" kulcsszóval létrehozzunk egy objektumot a PolarGenerátor osztálynak. Az objektum meghívásával fog lefutni a konstruktorunk, ami True értéket állít be, a logikai változónknak. Ezután jön a for ciklus, ez 10 alkalommal fog lefutni és minden alkalommal meghívjuk a public double következő() metódust. Így fogunk 10 darab véletlenszerű számot generálni.

A kódot 'PolárGenerátor.java' néven mentjük el, és futtatjuk az alábbi módon: 'javac PolárGenerátor.java'.



```
klssmate3@klssmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ javac PolarGenerator.java  
klssmate3@klssmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ java PolarGenerator  
-1.0675249683821944  
-1.0657057858313417  
0.5095390042490706  
-0.37186280235723523  
0.7005130322629114  
-1.0127395843904035  
0.6179637716586457  
-0.1430316141812717  
-1.564340872125306  
-2.0900135899620786
```

Most lássuk ezt C++ nyelven:

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

Ugyan úgy, ahogy a Javaban, itt is egy osztályal kezdünk. Az osztályon belül van, publikus és privát rész. A public részben találjuk a konstruktort ami inicializálja a "nincsTarolt" változót. Továbbá itt található az srand() függvény, amivel C++ ban véletlenszerű számokat generálunk. A másik a destruktur, ezt a '~' előtagból tudjuk. Ezt általában memóriafelszabadításra használjuk. Ez akkor fut le, ha töröljük az

objektumot, vagy lezárjuk. A private részben pedig a két változónk van, amiknek feladatait már a Java-s kódban leírtuk. A private és public között az a különbség, hogy a public az osztályon kívül is elérhető még a private csak osztályon belül használható. A PolarGen osztályban találjuk még a következő() függvényt, ahogy a Java-s kódban itt is ez a függvény fogja végezni a számítást. Ez ugyan úgy ha nincs tárolt érték akkor egy értéket vissza ad, egyet pedig eltárol, ha pedig van tárolt érték, akkor azt adja vissza.

```
int main (int argc, char **argv)
{
    PolarGen pg;
    for (int i= 0; i<10;++i)
        std::cout<<pg.kovetkezo ()<< std::endl;
    return 0;
}
```

A fő függvényünkben itt is az objektum található és a for ciklus, ami 10 alkalommal fog lefutni. A két kódsor között a két nyelv különbségei, szyntaxa de a két nyelv hasonlóságai is megfigyelhető.

12.2. Homokozó

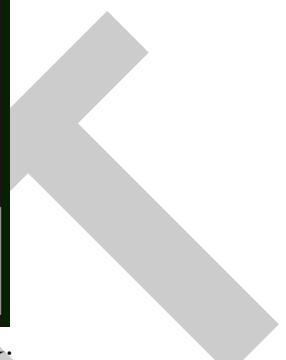
Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden más működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer,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

Tutor: Tóth Attila

A könyvünkben Welch fejezetében már foglalkoztunk az LZWBinfával, akkor C++ nyelven írtuk meg. Most át fogjuk írni Java nyelven. A működése ugyan az lesz. A program bemeneti adatokból fog felépíteni egy bináris fát. Egy fa akkor bináris, ha a fa minden csomópontjának maximum 2 gyermeké van. A bináris fa 0 és 1-es számokból épül fel. Fontos még tudni, hogy van egy kitüntetett elem, amit gyökérnek hívunk, innen tudunk elérni minden elemet, amit a fa tartalmaz. (A fa sok tulajdonsággal rendelkezik, van magassága, mélysége, elemszáma, szórása, ágak hossza, ágak száma).

A kód átírása után a java kódon (lásd a feladat alján) látszik a hasonlósá a C++ -os kóddal, nem sok változást lehetünk észre. Az első lépésünk az volt, hogy töröltük a pointereket és a referenciákat a C++ -os forrásból. Erre azért van szükség mert a Java nem tűri a pointereket és a referenciákat. Ezeket az objektumok referenciaival helyettesítjük. Lényeges különbség még a destrukturált a lefoglalt memória felszabadítására használtuk, de Javaban nem lesz erre szükségünk, ugyanis a Garbage Collector azaz a szemétfogolyító megteszí ezt helyette. A szemétfogolyító automatikusan felszabadítja a már nem használt memóriát. Különbség még, hogy Javaban nincs oerátorúterhelés, így ezeket függvényekre cseréljük. Ezen kívül pedig a C++ osztályokon belül lévő public és privát részt is elhagyjuk, itt külön külön kell elő írni a függvény, változó elé, hogy milyen a jogosultsága.

Most nézzük, hogyan is tudjuk a böngésző címsorából átadni az adatokat a binfának. Ehez Java Servletre lessz szükségünk. Első dolgunk telepíteni az apache tomcat az alábbi módon: https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-8-on-ubuntu-16-04?fbclid=IwAR0juqF-l5y9SxYBk-1-lgNMchwPq_WEeyssS5PH5ldOdMlaFJpUvMXP4 Miután ezzel kész vagyunk, a böngészőn keresztül megadjuk a bemenetet. Ezt "?text=" módon tesszük meg (pl ?text=01010011101010)

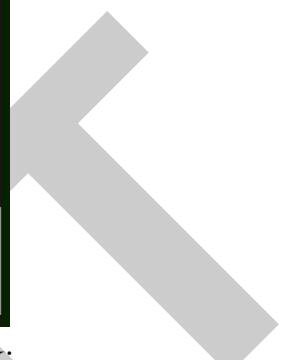
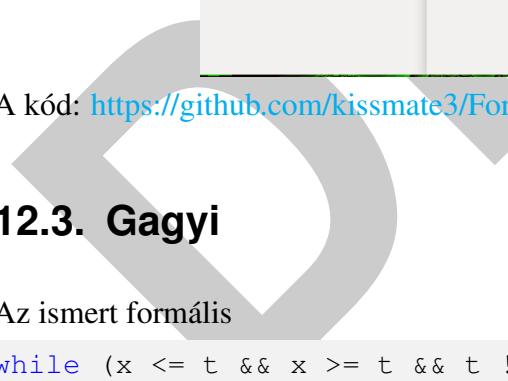


```

File Edit View Search Terminal Help
[ubuntu2,1-18.04.4 [1.300 kB]
Get:2 http://hu.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcurl4 amd64 7.58.0-2ubuntu3.8 [214 kB]
Get:3 http://hu.archive.ubuntu.com/ubuntu bionic-updates/main amd64 curl amd64 7.58.0-2ubuntu3.8 [159 kB]
Fetched 1.673 kB in 1s (1.229 kB/s)
Preconfiguring packages
(Reading database ... 278677 files and directories currently installed.)
Preparing to unpack .../libssl1.1_1.1.1-1ubuntu2.1-18.04.4_amd64.deb ...
Unpacking libssl1.1_amd64 (1.1.1-1ubuntu2.1-18.04.4) over (1.1.0g-2ubuntu4.3) ...
Preparing to unpack .../libcurl4_7.58.0-2ubuntu3.8_amd64.deb ...
Unpacking libcurl4_amd64 (7.58.0-2ubuntu3.8) over (7.58.0-2ubuntu3.6) ...
Selecting previously unselected package curl.
Preparing to unpack .../curl_7.58.0-2ubuntu3.8_amd64.deb ...
Unpacking curl (7.58.0-2ubuntu3.8) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Setting up libssl1.1_amd64 (1.1.1-1ubuntu2.1-18.04.4) ...
Checking for services that may need to be restarted...done.
Checking for services that may need to be restarted...done.
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up libcurl4_amd64 (7.58.0-2ubuntu3.8) ...
Setting up curl (7.58.0-2ubuntu3.8) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
kissmate3@kissmate3-VirtualBox:/tmp$ curl -o http://apache.mirrors.tonfish.org/tomcat/tomcat-8/v8.5.5/bin/apache-tomcat-8.5.5.tar.gz
  % Total    % Received % Xferd  Average Speed   Time     Time   Current
          Dload  Upload Total Spent   Left Speed
100  327  100  327    0     0  1368      0 --:--:-- --:--:-- 1368
kissmate3@kissmate3-VirtualBox:/tmp$ sudo mkdir /opt/tomcat
kissmate3@kissmate3-VirtualBox:/tmp$ sudo tar xzvf apache-tomcat-8*tar.gz -C /opt/tomcat -strip-components=1
gzip: stdin: not in gzip format
tar: Child returned status 1
tar: Error is not recoverable: exiting now

```

A kimeneten láthatjuk, hogy a Java és a C++ LZWBinfá kimenetje megegyezik:

```

kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/1.csomag
Open ▾  Open ▾
-----1(3)
-----1(2)
-----1(4)
-----0(5)
0(3)
-----1(1)
1(3)
-----0(4)
-----0(2)
-----0(3)
-----1(5)
-----0(4)
---/(0)
1(3)
-----1(2)
-----1(4)
0(3)
-----1(5)
-----0(6)
-----0(4)
-----0(5)
-----0(1)
-----1(4)
1(3)
-----0(4)
-----0(2)
depth = 6
mean = 4.3
var = 0.9486832980505138
-----1(3)
-----1(2)
-----1(4)
-----0(5)
-----0(3)
-----1(1)
-----1(3)
-----0(4)
-----0(2)
-----0(3)
-----1(5)
-----0(4)
---/(0)
-----1(3)
-----1(2)
-----1(4)
-----0(3)
-----1(5)
-----0(6)
-----0(4)
-----0(5)
-----0(1)
-----1(4)
-----1(3)
-----0(4)
-----0(2)
depth = 6
mean = 4.3
var = 0.948683

```

A kód: <https://github.com/kissmate3/Forr-sok/blob/master/LZWBinFa.java>

12.3. Gagyi

Az ismert formális

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

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

Javaban az Integer típusú számokat a java -128 és 127 között eltárolja a poolba. (Feltételezi a java, hogy a program sokat fog kisebb számokkal dolgozni, és ha Intiger típusú szám kell, akkor a poolbol kivesz.) Ha

a -128 és 127 közötti intervallumból szeretnénk értéket adni egy változónak, akkor az előre eltárolt pool-ból fog hozzá memória címet rendelni. Ez úgy történik, hogy ebben az esetben meghívódik az Intiger.value.Of függvény, ami kiosztja a címet. Ha a megadott tartományok kívül rendelünk a változóhoz értéket, például -129 akkor viszont ugyan ez a függvény létrehoz egy új objektumot. Ezek alapján ha x és t változó értéke egyenlő és ezen értéke az adott tartományon belül vannak, akkor azonos lesz a két változó címe. Viszont ha x és t értéke azonos, de nem a -128 és 127 tartományból lesz, akkor a két változó címe nem fog megegyezni, mivel két külön objektum fog létrejönni ezeknek a számoknak.

A megadott feltéellel írunk két példa programot: A feltétel a két eset kapcsán egyszer végtelen ciklusba lép, a másik esetben pedig a feltétel nem teljesül.

```
class elseoeset{
    public static void main(String args []) {
        Integer x=120;
        Integer t=120;

        while (x <= t && x >= t && t != x)
            System.out.println("A feltétel nem teljesül");
    }
}
```

Ebben az esetben láthatjuk, hogy az x és a t értéke is 120, azaz egyenlő, továbbá mind a két érték benne van a tartományba, tehát a poolbol ugyan azt a címet fogják megkapni. A feltétel így nem teljesül, tehát a program nem fog végtelen ciklusba lépni.

```
class masodikeset{
    public static void main(String args []) {
        Integer x=140;
        Integer t=140;

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

Itt is megegyezik, x és t értéke, de ez az érték már a tartományon kívül lesz, így két különböző objektumból kapják a változók a címeket, így az összehasonlításkor nem lessz azonos a cím, tehát a programunk végtelen ciklusba fog lépni

A JDK forráss:

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

12.4. Yoda

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

A Yoda egy programozási stílus, pontosabban Yoda feltételeknek hívjuk. A lényege pedig az, hogy az összehasonlító kifejezésben megcseréljük a változót és a konstanst. Eredetileg a konstanshoz hasonlítjuk a változót, azaz a konstans a bal még a változó a jobb oldalon helyeszkedik el. De a Yoda feltételek szerint pont fordítva, azaz a konstans lesz a jobb oldalon és a változó a bal oldalon. Ez a megszokott szintaxisztól eltérő. A nevét a Csillagok Háborújából ismert Yoda mesterről kapta, mivel furán beszéli az angolt, szavakat felcserél, nem a szokványos angol szintaxis szerint beszél. Nézzünk egy példát:

Álltalálosan:

```
if(érték == 20) {Tortenjen valami}  
// Ez egy hagyományos feltétel. Jelentése: Ha az érték egyenlő 20 al...
```

Yoda feltétel szerint:

```
if(20 == érték) {Tortenjen valami}  
// Ez egy nem hagyományos feltétel. Jelentése: Ha a 20 egyenlő az értékkel ←  
...  
...
```

Az első példa szerint a változó értéke 20 lesz, még a másik szerint hibát kapnánk a kód fordítása során. Ezért ha a Yoda stílusban írjuk a programot, ezt a hibát kiküszöböltük. További előnye a Yoda stílusnak még, hogy el tudjuk kerülni a nem biztonságos viselkedésű null típusokat:

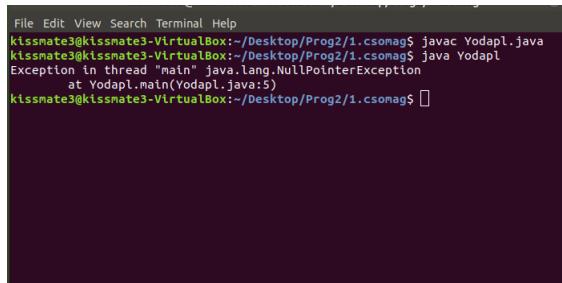
Yoda feltétel szerint:

```
String peldaString = null;  
if(peldaString.equals("foobar"))
```

A yoda feltételeknek vannak hátrányai. Azok akik kritizálják anyelvet, azt mondják, hogy olyan szinten rontja az olvashatóságát a programnak, hogy meghaladja a Yoda stílus előnyeit. A Swift programozási nyelv és néhány másik nem engedi meg a változó hozzárendeléseket egy feltételen belül. Továbbá amit fent előnyeket tüntettünk fel, az lehet hátrány is mivel a null muta hibákat elrejtjük ugyan, de ezek később jelentkezhetnek a megírt programban. Van egy hátrány ami C++ -ban van jelen. A hiba akkor jöhét elő, mikor nem alaptípusokat hasonlítunk össze.

Most pedig írjuk meg a programunkat ami java.lang.NullPointerException-t leállít, ha nem követjük a Yoda conditions-t!

```
class Yodapl{  
  
    public static void main(String args[]){  
        String peldastring = null;  
        if (peldastring.equals("pelda")){  
            System.out.println("Pelda kod")  
        }  
    }  
}
```



```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ javac Yodapl.java
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ java Yodapl
Exception in thread "main" java.lang.NullPointerException
        at Yodapl.main(Yodapl.java:5)
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ 
```

12.5. Kódolás from scratch

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

Ebben a feladatban a Balley-Borwien-Plouffe (rövidítve BBP) formulákról fogunk beszélni. Ez a formula a π (pí) képlete. A névét a képlet felfedezőjéről és a cikket megjelenítő kiadójának vezetőiről neveztek el. A felfedező Simon Plouffe volt 1995-ben. A képlet a következő:

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

A képlet arra szolgál, hogy egy kiválasztott Pi számjegyétől hexadecimális formában tudjunk kiszámítani további Pi számjegyeket. A képletet egy program találta, érdekesség, hogy ez volt az első lényegesebb formula, amit egy program talált.

Most beszéljünk a kódrol:

```
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();
}
```

Elsőnek egy String tipusú változót hozunk létre 'd16PiHexaJegyek' néven. Ezt a változót fogjuk használni a hexadecimális jegyek tárolására. Aztán következik a konstruktur, ami megkapja a megadott számot, ahonnan elkezdi a számítást. A számításokhoz 5 darab double tipusú változót deklarálunk. Ezek után láthatjuk a képletet a programba. A StrictMath.floor() függvényel vissza kapjuk az eredmény egész részét, A 'hexaJegyek' karaktertípusú válzotóba megadjuk, a hexadecimális jegyekhez szükséges betűket. Aztán egy while cikluson belül elvégezzük a számítást és egy if feltételen belül a megfelelő helyeken össze rakjuk az eredményt. A 'd16PiHexaJegyek = sb.toString();' sorban a kapott eredmény a toString() függvényel String be visszük át és így stringként kapja meg a 'd16PiHexaJegyek' változónk.

```
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);
}

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;
}
```

Ezek után következik egy double típusú függvény ami megkapja paraméterül a 'd' kiválasztott számot és egy egész számot. Ebben a függvényben számításokat fogunk végezni maradékképzéssel. ehez pedig a hosszú egész típusú n16modk függvényt hívjuk meg.

```
public String toString() {

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

A `toString` függvényt már a fentiekben említettük, ez `String` típusba adjja vissza az értéket. Végezetül a fő függvényben kiíratjuk az eredményt.

```
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ javac PiBBP.java
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ java PiBBP
6C65E5308kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$
```

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

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

A Liskov féle helyettesíthetőségi alapelve (röviden LSP) egyike a S.O.L.I.D objektum orientált tervezési elveknek. Az elv nevét a kidolgozója után kapta, aki Barbara Liskov volt. A lényege pedig, hogy a leszármazott osztályoknak a példányainak ugyanúgy kell viselkednie mint az őssosztályban lévő példányoknak. Ezt ugy tudnánk röviden megmagyarázni, hogyha egy programban az őssosztály valamelyik példánya helyett a leszármazott osztály példányát használjuk, akkor a program működésében semmilyen változás nem történhet. Példának vegyük Struc és a Gólya példáját. A Struc és a Gólya is a madár osztályába tartozik. A Madár osztály rendelkezik a repül függvényel, mivel tudjuk hogy a legtöbb madár repül. Itt jön majd egy hiba az öröklődésben, mert a struc és a gólya alosztály is megörökli a repül függvényt, pedig a Struc nem tud repülni. Így a program működésében nem lesz hiba, de hibás programtervet eredményez. Hasonlóan tudunk még példákat felhozni, ilyen a kör és elipszis, vagy a négyzet és a téglalap területsszámítása.

Most nézzünk egy C++ kódrészletet, amivel megsérjük ezt az elvet.

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

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

class Golya : public Madar
```

```
{ };

class Struc : public Madar
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Golya golya;
    program.fgv ( golya );

    Struc struc;
    program.fgv ( struc );
}
```

Most pedig nézzük meg Java-nyelven:

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

class Prog {
public void fgv ( Madar madar) {
    madar.repul();
}

};

class Golya extends Madar {};
class Struc extends Madar {};

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

        Golya golya = new Golya();
        kod.fgv (golya);

        Struc struc = new Struc();
        kod.fgv(struc);
    }
}
```

A kódban először létrehozzuk a Madár osztályt, ez lesz a szülőosztály. Létrehozzuk benne a repul() függvényt. Az osztály létrehozása után a Gyólya és a Struc leszármazott osztály. Itt mind a két osztály megkapja

a madar osztályt. A Liskov osztályban pedig példányosítunk.

13.2. Szülő-gyerek

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

A feladatunk az, hogy bebizonyítsuk egy programon keresztül, hogy egy ősosztály csak a saját metódusait tudja használni, a leszármazott osztály metódusait már nem. Azaz például az ősosztály csak azokat a függvényeket fogja tudni használni, ami a saját függvénye, egy új függvényt amit leszármazott osztályban hoztunk létre, azt nem fogja tudni használni.

Elsőnek nézzünk erre példát Java nyelven:

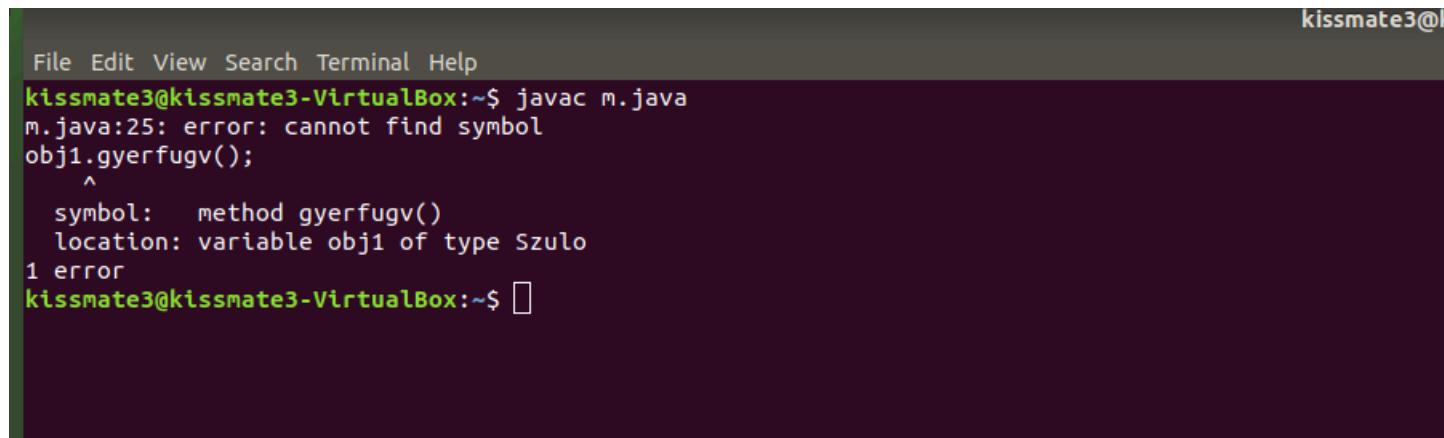
```
class Szulo
{
    public void szulofugv()
    {
        System.out.println("Szulo fuggveny");
    }
}

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

    public static void main (String[] args)
    {
        Szulo obj1 = new Szulo();
        Gyermek obj2 = new Gyermek();

        obj1.szulofugv();
        obj2.gyer();
        obj2.szulofugv();
        obj1.gyer();
    }
}
```

A kódot megnézve látjuk hogy az ősosztály a Szulo, az extends utasítással leszármazottja lesz a Gyermek osztály, mind a két osztályban létre hoztunk egy saját függvényt. A fő függvényünkhez mind a két osztálynak létrehozunk egy egy objektumot obj1 és obj2 néven. Ezek után az objektumokkal meghívjuk a függvényeket. Az első három hívás rendesen működik, de az utolsónál mikor a szulóvel akarjuk meghívni a gyermek függvényét beleütközünk a fent leírtakba, így bizonyítva van, hogy az ősön keresztül csak az ős üzenetei küldhetőek.



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The title bar of the terminal window says "kissmate3@kissmate3-VirtualBox:~\$". The main area of the terminal contains the following text:

```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~$ javac m.java
m.java:25: error: cannot find symbol
    obj1.gyerfugv();
           ^
      symbol:   method gyerfugv()
      location: variable obj1 of type Szulo
1 error
kissmate3@kissmate3-VirtualBox:~$
```

Most pedig nézzünk erre egy példát C++ nyelven:

```
#include <iostream>

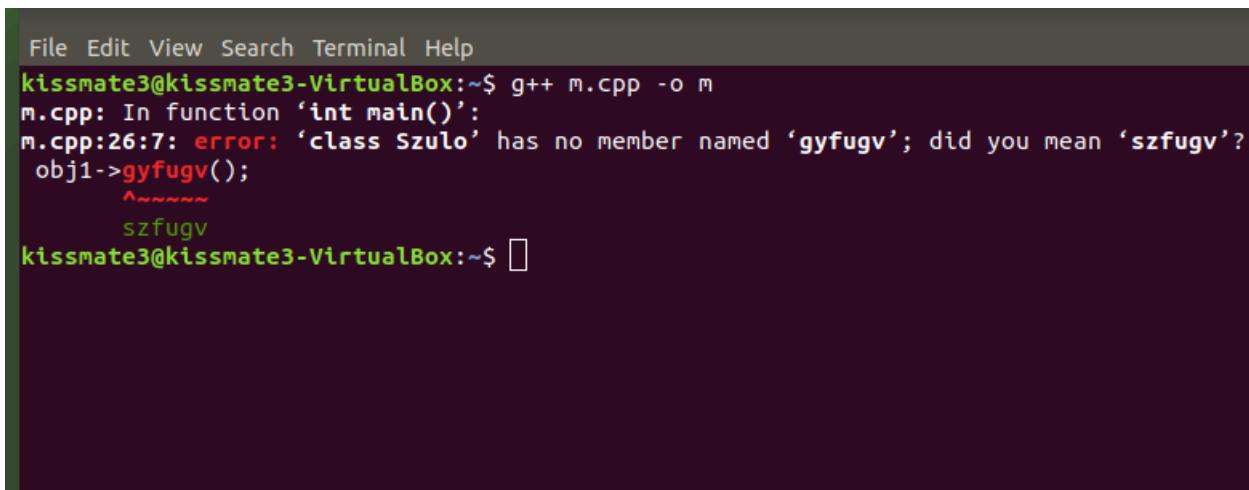
using namespace std;

class Szulo{

public:
void szfugv() {
    cout<<"Szulo fuggve"<<endl;
}
};

class Gyermek: public Szulo{
public:
void gyfugv() {
    cout<<"Gyermek fuggveny"<<endl;
}
};

int main(){
Szulo* obj1 = new Szulo();
Gyermek* obj2 = new Gyermek();
obj1->szfugv();
obj2->gyfugv();
obj2->szfugv();
obj1->gyfugv();
return 0;
}
```



```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~$ g++ m.cpp -o m
m.cpp: In function 'int main()':
m.cpp:26:7: error: 'class Szulo' has no member named 'gyfugv'; did you mean 'szfugv'?
    obj1->gyfugv();
          ^~~~~~
          szfugv
kissmate3@kissmate3-VirtualBox:~$
```

13.3. Anti OO

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

Az első csokorban megbeszélt BBP algoritmust futási idejét fogjuk most összehasonlítani négy programozási nyelven. Ez a négy nyelv a C, C++, C# és a Java. Az algoritmussal a 0 ik pozícióból a 10^6 , 10^7 és a 10^8 darab jegyeket fogjuk meghatározni. A számítások elvégzése után ezt az eredményt kaptam:

	Java	C	C++	C#
10^6	1.578	1.665	2.069	1.776
10^7	17.746	20.084	23.741	18.5
10^8	207.95	231.001	263.882	205.74

Az eredmények alapján arra jutottam, hogy a számítást az első két esetben a Java még a harmadik esetben a C# végezte el, De az első két esetben is nagyon kis különbség volt a Java és a C#. minden esetben a leglasabb a C++ volt. A C nyelv első esetben a második legjobb volt, de a másik két esetben közepe eredményt ért el. A számításokhoz használt számítógép egy I7 7700HQ nyolcmagos processzor 8gb DDR4 rammal, de mivel Virtuális gépen végeztem a számítást így 4 maggal és 4gb rammal történt a teszt.

```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ g++ PiBBP.cpp -o PiBBP
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./PiBBP
6
2.06969
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ g++ PiBBP.cpp -o PiBBP
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./PiBBP
7
23.7412
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ g++ PiBBP.cpp -o PiBBP
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./PiBBP
12
282.594
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ 
```



```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ gcc pi_bbp_bench.c -o pi_bbp_bench -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main()
 ^
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./pi_bbp_bench
6
1.665548
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ gcc pi_bbp_bench.c -o pi_bbp_bench -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main()
 ^
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./pi_bbp_bench
7
20.084121
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ gcc pi_bbp_bench.c -o pi_bbp_bench -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
 main()
 ^
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./pi_bbp_bench
12
231.001771
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ 
```



```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mono PiBBPBench.exe
6
1.766523
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mcs PiBBPBench.cs
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mono PiBBPBench.exe
7
18.506215
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mcs PiBBPBench.cs
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mono PiBBPBench.exe
12
205.748154
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ 
```

13.4. Hello, Android!

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

A feladatot az UDPORGban Racs Tamás által megosztott segítséggel tudtam elkészíteni, ezért tutor: Racs Tamás

A feladatunk az volt, hogy színeket változtassuk meg a SMINSTforHumans programban. Én ezt Windows 10 operációs rendszeren végeztem el az Android Studio ingyenes programot használva. Miután a feltelepítés megtörtént, a Studion belül megnyitottam a SMINSTforHumans projektet. A változtatásokat pedig a SMINSTSurfaceView.java állományban végeztem. Elsőnek a bgColor -ban írtam át a hátterek RGB kódjait (lásd az első kép). Majd további módosításokat hajtottam végre a textPaint, msgPaint, dotPaint és broderPaint változókon (lásd a második képen).

```
InterfaceView.java
private static final int NUM_OF_DIGITS = 10;
private float[] semValueDots = new float[2 * NUM_OF_DIGITS];

private float[] digitsCoords = new float[2 * NUM_OF_DIGITS];

private int successCnt = 0;
private boolean armed = true;
private long armedTime = 0;

int[] bgColor =
{
    android.graphics.Color.rgb( red: 210, green: 83, blue: 77),
    android.graphics.Color.rgb( red: 255, green: 169, blue: 83)
};

int bgIdx = 0;

private int decisionTimeDelaySum = 0;
private int semValueSum = 0;

private static String msg1 = "How many dots can you see?";
private static String msg2 = "Touch the appropriate number.";

android.graphics.Paint dotPaint = new android.graphics.Paint();

MNISTSurfaceView > cinit()

textPaint.setColor(Color.BLACK);
textPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
textPaint.setAntiAlias(true);
textPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
textPaint.setTextSize(50);

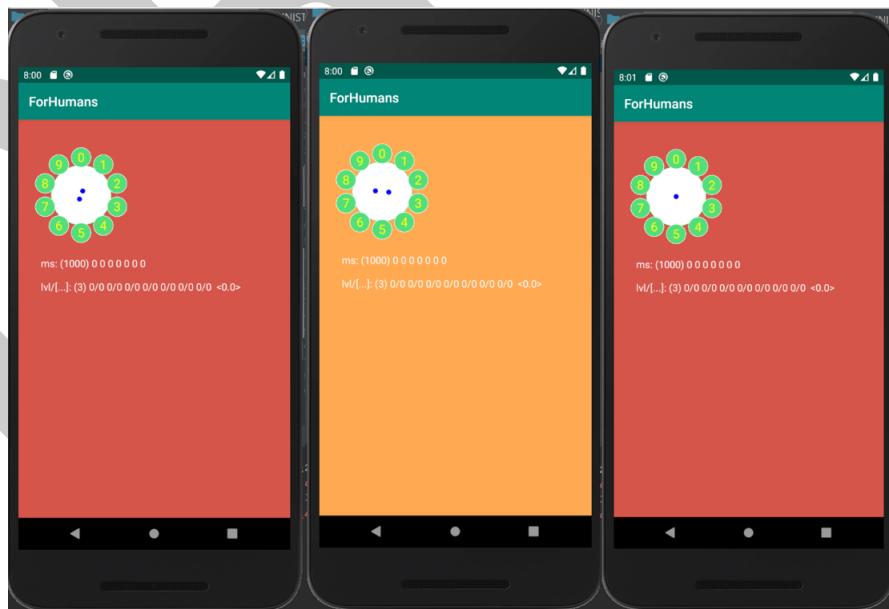
msgPaint.setColor(Color.WHITE);
msgPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
msgPaint.setAntiAlias(true);
msgPaint.setTextAlign(android.graphics.Paint.Align.LEFT);
msgPaint.setTextSize(40);

dotPaint.setColor(Color.BLUE);
dotPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
dotPaint.setAntiAlias(true);
dotPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
dotPaint.setTextSize(50);

borderPaint.setStrokeWidth(2);
borderPaint.setColor(Color.WHITE);
fillPaint.setStyle(android.graphics.Paint.Style.FILL);
fillPaint.setColor(android.graphics.Color.rgb( red: 45, green: 54, blue: 166));

STSurfaceView > cinit()
```

Ezek után a programot lefuttatva a szoftveren belül egy virtuális telefonon megkapjuk az eredményt. A program kinézetét a szöveg alatti képen tekinthetjük meg. Ha a saját telefonunkon akarjuk megnézni, akkor össze tudjuk kapcsolni azt a Studióval, ehez szükségünk lesz a telefonunk mobile driver-ére. Ezután a telefonunkon még engedélyezni kell a fejlesztői beállításokon belül az USB hibakeresést. Ezek után tudjuk a programmal összhangba hozni a telefonunkat. Jó játékot!



13.5. Ciklomatikus komplexitás

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

Először azzal kezdünk, hogy mi is az amiről beszélünk. A Ciklomatikus komplexitás egy szoftvermetrika. Sokszor szoktuk McCabe komplexitásnak is nevezni, alkotója után aki Thomas J. McCabe. Ő 1976-ban publikálta. A komplexitást egy konkrét számértékben fogja kifejezni egy meghatározott szoftver forráskódja alapján, a gráfelméleten alapszik a számítás. A ciklomatikus komplexitás értéke az alábbi:

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

A képleten belül az összefüggő komponensek számát, azaz a 'P'-t megszorozzuk kettővel. Az 'E' jelenti a gráf éléinek számát és az 'N' pedig a csúcsok számát. Egy programban ez azt jelenti, hogy egy gráfot egy függvényben megtalálható utasítások alkotnak. Ha egy utasítás után egyből lefut a másik akkor a két utasítás között él található.

A feladatban én az LZWBinfának számoltam ki a komplexitását. Ehez a Lizard programot használtam. A képen a CCN oszlopban látható az LZWBinfá függvényeinek a komplexitása:

```
File Edit View Search Terminal Help
ktssmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$ lizard LZWBinfFa.java
=====
NLOC    CCN   token  PARAM  length  location
-----
3       1      9      0      5 LZWBinfFa:::LZWBinfFa@5-90@LZWBinfFa.java
22      4     104     1      28 LZWBinfFa:::egyBelfieldolq@12-39@LZWBinfFa.java
4       1      26      0      7 LZWBinfFa:::kitrg42-48@LZWBinfFa.java
4       1      22      1      4 LZWBinfFa:::kitrg53-56@LZWBinfFa.java
5       1      21      1      5 LZWBinfFa:::csomopont@62-66@LZWBinfFa.java
3       1      8      0      3 LZWBinfFa:::csomopont:::nullasGyernek@76-72@LZWBinfFa.java
3       1      8      0      3 LZWBinfFa:::csomopont:::egyesGyernek@75-77@LZWBinfFa.java
3       1      11      1      3 LZWBinfFa:::csomopont:::ujNullasGyernek@80-82@LZWBinfFa.java
3       1      11      1      3 LZWBinfFa:::csomopont:::ujEgyesGyernek@85-87@LZWBinfFa.java
3       1      8      0      3 LZWBinfFa:::csomopont:::getBeltug90-92@LZWBinfFa.java
15      3     107     2      17 LZWBinfFa:::kitrg107-123@LZWBinfFa.java
5       1      21      0      5 LZWBinfFa:::getMelyseg@132-136@LZWBinfFa.java
6       1      32      0      6 LZWBinfFa:::getAtlag@138-143@LZWBinfFa.java
12      2      68      0      15 LZWBinfFa:::getSzoras@145-159@LZWBinfFa.java
11      3      51      1      12 LZWBinfFa:::rnelysseg@161-172@LZWBinfFa.java
12      4      66      1      12 LZWBinfFa:::ratlag@174-185@LZWBinfFa.java
12      4      78      1      12 LZWBinfFa:::rszoras@187-198@LZWBinfFa.java
3       1      14      0      3 LZWBinfFa:::usage@202-204@LZWBinfFa.java
64     14     400     1      95 LZWBinfFa:::main@207-301@LZWBinfFa.java
1 file analyzed.
=====
NLOC    Avg.NLOC  AvgCCN  Avg.token  function_cnt  file
-----
207      10.2      2.4      56.1        19  LZWBinfFa.java
=====
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or parameter_count > 100)
=====
Total nloc  Avg.NLOC  AvgCCN  Avg.token  Fun Cnt  Warning cnt  Fun Rt  nloc Rt
-----
207      10.2      2.4      56.1        19      0      0.00      0.00
ktssmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csomag$
```

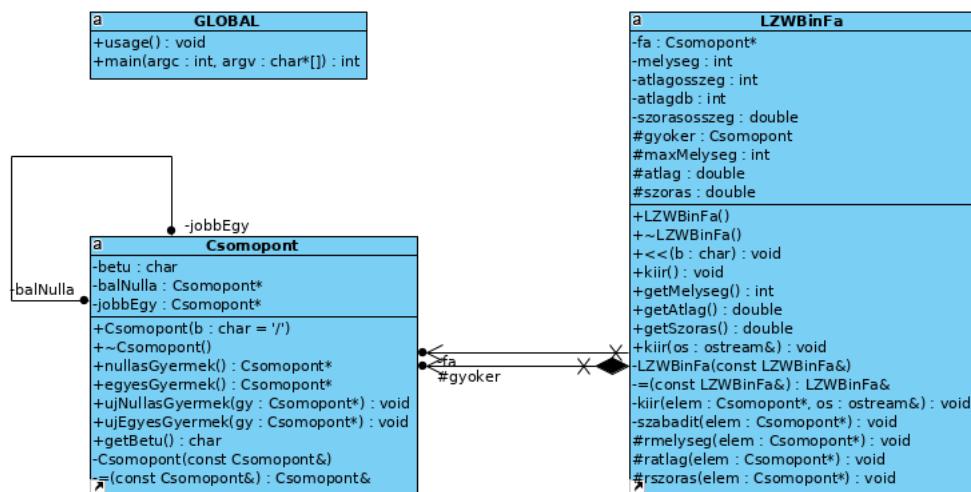
14. fejezet

Helló, Mandelbrot!

14.1. Reverse engineering UML osztálydiagram

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

A feladatban az első védési programnak, azaz a Binfának fogjuk megrajzolni az UML osztálydiagrammot. Az UML az OMG Egységes Modellező Nyelve szoftverrendszerök modelljeinek. Ebbe bele tartozik az UML felépítését és megtervezését. Ezen felül megjeleníti, dokumentálást elősegíti és részletes leírást ad a szoftverről. Az UML az OOSE, OMT és a Booch objektumorientált módszerek vezetésével lett kifejlesztve. Nézzük az LZWBina UML modelljét, amit a Visual Paradigm program segítségével készítettem:



Most beszéljük át a diagrammot. A diagram a program osztályait mutattja és a közöttük lévő kapcsolatot. minden egyes osztály egy téglalap. Az osztály neve a téglalap tetején vastag betűtíppussal szerepel, ezen felül

téglalap két részre van osztva, A felső részben szerepelnek az attribútumok. Itt mindegyik attribútumnak van egy előtagja, ezek mutattják a láthatósági jogokat. A + public, a - private és a # a protected. Az alsó részben vannak az osztály metódusai, az előtag itt is megtalálható. Az osztályok közötti kapcsolatokat két osztály összekötésével tudjuk jelölni, mi most a feladatban kért kompozíció és aggregáció kapcsolatával fogunk foglalkozni. Az aggregáció egész kapcsolatot jelent, egy speciális asszociáció. Tartalmazást jelölünk vele. Tehát ha veszünk egy objektumot az tartalmazza részben vagy egészben a másik objektumot. Úgy jelöljük, hogy két osztályt egy vonallal összekürtjük és a tartalmazó osztály oldalára egy üres rombuszt teszünk. Két aggregáció van. A gyenge és az erős, a gyenge olyan speciális aggregáció amikor a tartalmazó nem függ a tartalmazottól, azaz létezhet nélküle. Az erőss aggregáció a kompozíció. Itt a tartalmazó és tartalmazott szorosan megegyeznek. Együtt jönnek létre és együtt is szűnnek meg. Jelölése annyiban különbözik, hogy tömörrombuszt írunk.

14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

A feladatunk most pont az előző feladat fordítottja, ott egy kódból csináltunk UML osztálydiagrammot, most pedig az osztálydiagramból fogunk kódot létrehozni. Ehez használjuk az előbb generált Binfánkat. A feladathoz megint a Visual Paradigm programmal fogom elvégezni. Ugyanis a programban lehetőségünk van az UML ből kódot generálni. Ez azért lehetséges mert a diagram meghatározza az osztályok szerkezetét.

Az eredmény a következő:

The screenshot shows three code editors side-by-side, each containing a different class definition in C++:

- LZWBinFa.cpp:** Contains the implementation for the LZWBinFa class, which includes methods like `kiir`, `getMelyseg`, `getSzoras`, and `szabadit`. Most methods are marked with `// TODO - implement` and throw an exception if not implemented.
- Csomopont.cpp:** Contains the implementation for the Csomopont class, which includes methods like `Csomopont(char b)`, `nullasGyernek`, `egyesGyernek`, `ujNullasGyermek`, `ujEgyesGyermek`, and `getBetu`. Similar to LZWBinFa, many methods are marked as TODO.
- GLOBAL.cpp:** Contains the implementation for the GLOBAL class, which includes the `main` function and a `usage` method. Both are marked as TODO.

```

#ifndef CSOMOPONT_H
#define CSOMOPONT_H

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

#endif

```

```

#ifndef GLOBAL_H
#define GLOBAL_H

class GLOBAL {
public:
    void usage();
    int main(int argc, char* argv[]);
};

#endif

```

```

#ifndef LZWBINFA_H
#define LZWBINFA_H

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

#endif

```

A eredményt megkapva, láthatjuk, hogy a program váza, felépítése megegyezik az eredeti programával. A függvények törzsei értelemszerűen üres lesz. Ezzel a módszerrel könnyedén tudunk megtervezni egy program struktúrát, tehát aki szereti az ilyen táblázatok elkészítését az rengeteg időt tud magának megspórolni.

14.3. Egy esettan

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

A feladatot az elméettel fogjuk kezdeni. Ezért nézzük a 13 fejezet aminek a címe, hogy "A modellezés és a C++". A fejezet lényegében a C++ és az UML kapcsolatáról szól. Megtudjuk, hogy az UML napjaink egyetlen szabványos modellező nyelve a szoftverfejlesztésben. Mi az osztálydiagrammal fogunk foglalkozni. Ehez először megnézzük a 13.1 es fejezetben az osztályokat.

Az osztályok: Megtudjuk, hogy az osztályok a legfontosabb elemei az osztálydiagramnak, de ez elég egyértelmű lehetett. Egy osztályt téglalap jelöl aminek 3 fő része van. Az első szakasz megadja a nevét, a középső az osztálynak az attribútumait és az utolsó pedig a metódusokat. Egy osztály attribútumait a következő szintaxisossal adhatjuk meg: Először a láthatóság név, utánna a típus multiplicitás és az alapmérétezett értéket. Utána még adhatunk meg neki tulajdonságokat. Nézzünk erre egy példát:

- `b : float [2]=0.0`

A láthatóságnak négy esete van. `+, -, #, ~` ezek sorban a public, private, protected és a package de utóbbi nem található meg C++ -ban. A 13.2 a Kapcsolatokról ír a könyv. Az egyfajta kapcsolatot az UML -ben üres háromszögfejű nyíllal jelöli. Ha egy objektumból elszeretnénk érni egy hozzá tartozó objektumot, azt asszociációk hívjuk, ezt egy vonallal jelöljük. A könyv utánna az asszociációk speciális típusait veszi át, amikről előbbi feladatban már esett is szó. A 13.3 A kódgenerálást és a kódbiszaféjtést hozza. Az UML nyelv egyik leghasznosabb tulajdonsága hogy egy modellből autómatikusan kódot tud generálni. Erre visszafelé is képes az UML.

Most pedig nézzük az esettant:

A könyvben a feladat az hogy egy számítógép alkatrészekkel és konfigurációkkal foglalkozó boltnak készítsünk el egy alkalmazást. A program elsődleges célja, hogy nyilvántartásba legyenek az alkatrészek és a konfigurációk. Továbbá támogatnia kell a termékek betöltését az állományból és ki kell tudni listázni a képernyőre az adatokat. Nagyon fontos még hogy bővíthető legyen az alkalmazás ugyanis a cég gyorsan fejlődik, ezért a programunkat úgy kell megalkotni, hogy kezelni tudjuk az új termékcsaládot. Ezt két-komponensű megoldással fogjuk megvalósítani. Egy keresztrendszert valósítunk meg, ami támogatja az alkalmazás kezelését. A könyvben most nézzük meg, hogyan is kell megterveznünk ezt a programot.

Először vegyük át a követelményeket amit teljesítenie kell: 1. a forráskód maradjon privát. 2. támogassa a termékek betöltését és kiíratását a képernyőre. 3. A fontos attribútumok a beszerzési ár, név, típus és abeszérzés dátuma. Az aktuális árat függvényel számítjuk és a számításnak fügnie kell a termék típusától. 4. Lehet elemi és összetett termék is. Az összetett termék a konfiguráció, több mindenből épül fel. A könyveb egy UML osztálydiagram szemlélteti az osztálykapcsolatokat. Ennek és a források segítségével most atnézzük az osztályokat.

Kezdjük a product osztállyal. Lássuk az osztályt a forrás kód alapján:

```
#ifndef PRODUCT_H
#define PRODUCT_H

#include <iostream>
#include <ctime>

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

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

```
#endif
```

A product osztály a szülő osztálya a termékeknek. A leszármazottai lesznek az Összetett termék, Képernyő és a Merevlemez osztályok. Láthatjuk, hogy protected tagválzotók: (tehát a vevő csak olvashatják) beszerzési ár (int), beszerzési dátum (time_t), név (string) a product osztályban vannak. Virtuális metódusokra azért van szükségünk mert a fent említett leszármazott osztályok módosításokat fognak végezni. Most lássuk a függvényeket. A getAge() függvény meagadja a termék korát napokban. A getCurrentProduct művelet termékfüggő, ezért virtuális függvény lesz. A számítását majd a leszármazott osztályok fogják elvégezni. A getType() és a getCharCode() függvények a típusnév és a típuskód lekérdezését végzik. Ezek a függvények is virtuálisak lesznek mert leszármazott osztályokban kapnak majd implementációt. A formázott kiíratásért a Print függvény a felelős. A virtuális PrintParams függvény a beszerzési árat, dátumot, hogy a termék hány napos és a jelenlegi árat fogja kiírni.

```
#ifndef DISPLAY_H
#define DISPLAY_H
#include "product.h"

class Display: public Product
{
    int inchWidth;
    int inchHeight;

protected:
    void printParams(std::ostream& os) const;
    void loadParamsFromStream(std::istream& is);
    void writeParamsToStream(std::ostream& os) const;
public:
    Display();
    Display(std::string name, int initialPrice, time_t dateOfAcquisition, ←
            int inchWidth, int inchHeight);
    virtual ~Display() {}
    int getCurrentPrice() const;
    int getInchWidth() const;
    int getInchHeight() const;
    std::string getType() const {return "Display";}
    char getCharCode() const {return charCode;}
    static const char charCode = 'd';
};

#endif
```

A display osztály a képernyők adatait tárolják inchbe és a fent említett adatokat.

```
#ifndef HARDDISK_H
#define HARDDISK_H

#include "product.h"

class HardDisk: public Product
{
    int speedRPM;
```

```
protected:  
    void printParams(std::ostream& os) const;  
    void loadParamsFromStream(std::istream& is);  
    void writeParamsToStream(std::ostream& os) const;  
public:  
    HardDisk();  
    HardDisk(std::string name, int initialPrice, time_t dateOfAcquisition, ←  
        int speedRPM);  
    virtual ~HardDisk() {}  
    int getCurrentPrice() const;  
    int getSpeedRPM() const;  
    std::string getType() const {return "HardDisk";}  
    char getCharCode() const {return charCode;}  
    static const char charCode = 'h';  
};  
  
#endif
```

A HardDisk osztály a merevlemez gyorsaságát és a fent említett adatokat.

```
#ifndef COMPOSITEPRODUCT_H  
#define COMPOSITEPRODUCT_H  
  
#include <vector>  
#include <iostream>  
  
#include "product.h"  
  
class CompositeProduct: public Product  
{  
    std::vector<Product*> parts;  
protected:  
    void printParams(std::ostream& os) const;  
    void loadParamsFromStream(std::istream& is);  
    void writeParamsToStream(std::ostream& os) const;  
public:  
    CompositeProduct();  
    ~CompositeProduct();  
    void addPart(Product* product);  
};  
  
#endif
```

Itt az összetett termékekre vonatkozók vannak. Továbbá van a Productfactory osztály a beolvasásért felelős. Van még a ProductInventory osztály. Ez felel a termékek rendezéséért és megjelenítéséért.

Az eredmény:

```

File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/3.csomag/Esettan$ ./eset
Test1: create inventory and printing it to the screen.
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20191008, Age: 0, Current price: 30000, InchWidth: 13, InchHeight: 13
1.: Type: Harddisk, Name: WD, Initial price: 25000, Date of acquisition: 20191008, Age: 0, Current price: 25000, SpeedRPM: 7500
Press any key to continue...
Test2: loading inventory from a file (computerproducts.txt), printing it, and then writing it to a file (computerproducts_out.txt).
End of reading product items.The content of the file is:
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20011001, Age: 6580, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 4755, Current price: 28000, InchWidth: 10, InchHeight: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age: 4755, Current price: 70000
Items:
0. Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20011001, Age: 6580, Current price: 24000, InchWidth: 12, InchHeight: 13
1. Type: Harddisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 4755, Current price: 28000, SpeedRPM: 7000
3.: Type: Harddisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20050228, Age: 5334, Current price: 20000, SpeedRPM: 7000
The content of the inventory has been written to computerproducts_out.txt
Done.□

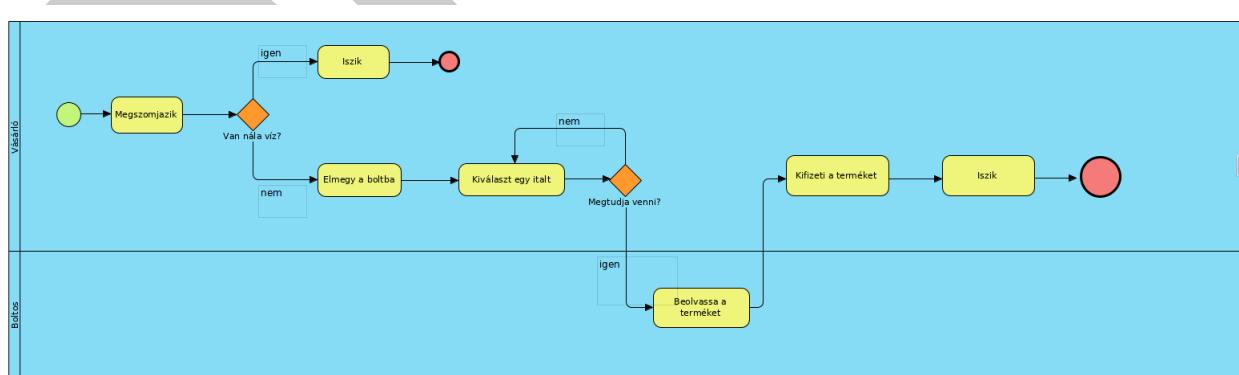
```

14.4. BPMN

Rajzolunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog> (34-47 fólia)

A BPNM, teljes nevén a Business Process Modeling Notation üzleti célú folyamatmodellező. Célja hogy a felhasználó egyszerűen megtudjon érteni egy folyamatot, vagy egy cég kezdeti folyamatterve is lehet, vagy egy implementáció megadása. Egy ilyen ábra létrehozása előtt nézzük meg, hogyan és miből áll a modell. A modell egy speciális folyamatábra. Különböző elemtípusokból épül fel. Ilyen az összeköttetés, tagolás és az adatfolyam. Az adatfolyamon beül van a kezdeti pont és a végpont (eseménytípusok), ezek között vannak az állapotváltozások. Az eseményeket különböző körökkel jelöljük. A tevékenység lehet összetett és alfolyamat. Jelölni négyzettel szoktuk. Az adatfolyam rendelkezik ájrólékkal: AND, OR, XOR...stb , ezt egy rombusszal szoktuk jelölni. Az összeköttetésnek három típusa van, 1. szekvencia: egy sima vonallal jelöljük. A tevékenység sorrendjét adjuk meg ezzel. 2. üzenet: Jelölése a szagatott vonal. Az információt szoktuk ezzel jelölni két résztvevő között. 3. szekvencia: Jelölése a pontozott vonal. Ezzel hozzárendelünk adatot és szöveget a modellhez. A tagolásnál pedig résztvevőkként szoktunk tagolni.

Az elkészített tevékenység BPMN-ben: (Az ábrát Visual Paradigmmal készítettem és egy bolti helyzetet ír le.)



14.5. TeX UML

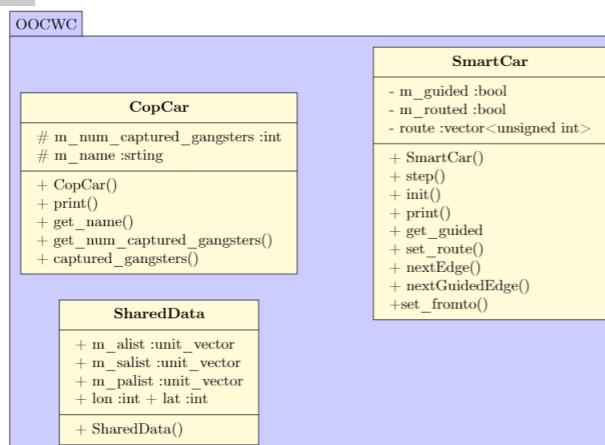
Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat)

A feladat elvégzéséhez a Tikz-uml csomagot használtam fel. A modellt pedig online Latex szerkesztőben készítettem aminek a neve Overleaf. A feladatban is megjelölt OOCWC projektből készítettem osztálydiagrammot. Az OOCWC egy régi program ami egy város közlekedését próbálja meg vissza adni. Én három osztályt modelleztem le: A CopCar, SmartCar és a ShareData-t. Most nézzük meg a smartCart osztályt hogyan is modellezük latexban:

```
\umlclass[x=7, y=0] {SmartCar}
{
    - m\_guided:bool \\
    - m\_routed:bool \\
    - route:vector<unsigned int>\\
}
{
    + SmartCar()\\
    + step()\\
    + init()\\
    + print()\\
    + get\_guided\\
    + set\_route()\\
    + nextEdge()\\
    + nextGuidedEdge()\\
    + set\_fromto()
}
```

Az \umlclass parancsal hozzuk létre magát a téglalapot. A [] zárójelek között a téglalap pozícióját adtuk meg. Ahogy a fenti feladatokban már beszélünk róla, a téglap három fő részét {} között adjuk meg. Az első kapcsoszárójel között adjuk meg az osztály nevét, A második között megadtuk az attríbutumokat. Jelen esetben -, azaz private a láthatóság. Utána a név és a típus jött. A \\ a sortörést jeletni. A harmadik részben a metódusok láthatóak felsorolva.

Eredmény:



15. fejezet

Helló, Chomsky!

15.1. Encoding

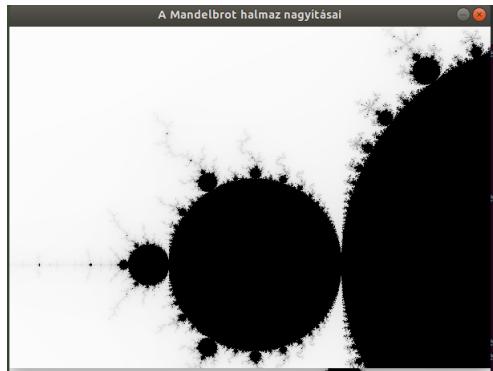
Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

A MandelbrotHalmazNagyítóval, könyünk első fejezeteiben már találkozhattunk. Mostani feladatunk az lesz, hogy a megadott forráskóddal le tudjuk futtatni. A már ismert kódtól ez a kód annyiban tér el, hogy használja az ékezes Magyar betűket. Ugye álltalában angol nyelven szoktunk kódolni ezért nem használunk ékezes betűket. Tehát ha csak egyszerűen futtatjuk ahogy egy egyszerű java programot szoktunk, 100 hibába ütközünk, és minden az ékezet okozta hiba. Először is, a program fájlainak a neveiben kellett átírnunk ékezes karakterekkel a neveket, ahol az nem volt megfelelő. Ezek után törököljük a forráskódban lévő hibákra. Először meg kell keresnünk a magyar karakterkészletet amit használhatunk encodingnál. Ezt az interneten könnyedén megtaláljuk. Én ehez az ISO-8559_2 karakterkészletet használtam. A fordítónak ez egy encoding kapcsolóval jelezük az alábbi módon:

```
$ javac -encoding "ISO-8559_2" MandelbrotHalmazNagyító.java
```

Ezt a karakter készletet használva le is fordul a program és tudjuk is használni:





15.2. Full screen

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

A feladat lényege, hogy egy full screen Java programot készítsünk. Tanárúr linkelt nekünk egy régi programot. Először ezt próbáljuk meg életre bírni. Az összes forrás megtalálható a linken. A program egy Labirintus játék. Lényege röviden annyi, hogy van egyy hősünk, akivel meg kell szerezni a kincset, de szörnyek üldöznek minket. és közben labirintusba vagyunk. A játék méri az időt, a pontokat és az életek számát. A program leglényegesebb része nekünk most a full screen lesz. A prgoramban ezt a teljesKépernyősMód() függvényel értük el.

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

    int szélesség = 0;
    int magasság = 0;
    setUndecorated(true);
    setIgnoreRepaint(true);
    setResizable(false);
    boolean fullscreenTámogatott = graphicsDevice.isFullScreenSupported ←
        ();
    if(fullscreenTámogatott) {
        graphicsDevice.setFullScreenWindow(this);
        java.awt.DisplayMode displayMode
            = graphicsDevice.getDisplayMode();
        szélesség = displayMode.getWidth();
        magasság = displayMode.getHeight();
        int színMélység = displayMode.getBitDepth();
        int frissítésiFrekvencia = displayMode.getRefreshRate();
        System.out.println(szélesség
            + "x" + magasság
            + ", " + színMélység
            + ", " + frissítésiFrekvencia);
        java.awt.DisplayMode[] displayModes
            = graphicsDevice.getDisplayModes();
        boolean dm1024x768 = false;
        for(int i=0; i<displayModes.length; ++i) {
```

```
        if(displayModes[i].getWidth() == 1920
           && displayModes[i].getHeight() == 1080
           && displayModes[i].getBitDepth() == színMélység
           && displayModes[i].getRefreshRate()
           == frissítésiFrekvencia) {
            graphicsDevice.setDisplayMode(displayModes[i]);
            dm1024x768 = true;
            break;
        }

    }

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

} else {
    setSize(szélesség, magasság);
    validate();
    setVisible(true);
}

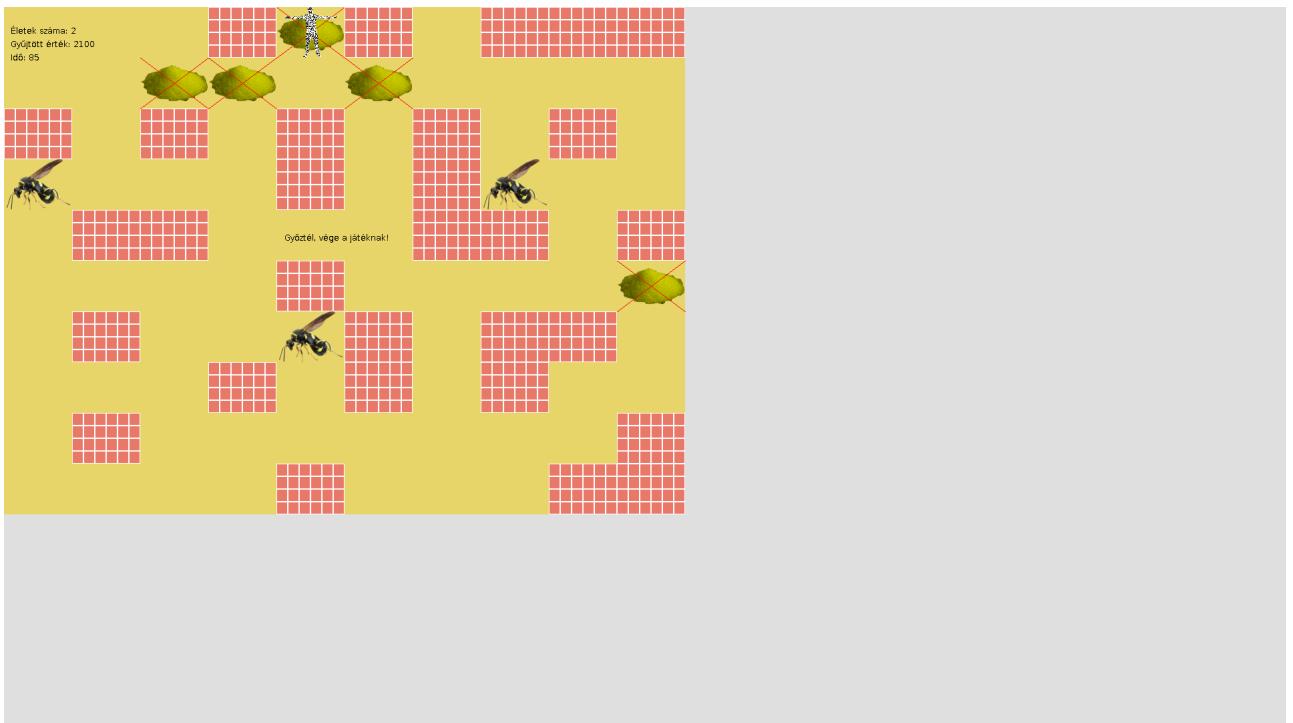
createBufferStrategy(2);

bufferStrategy = getBufferStrategy();

}
```

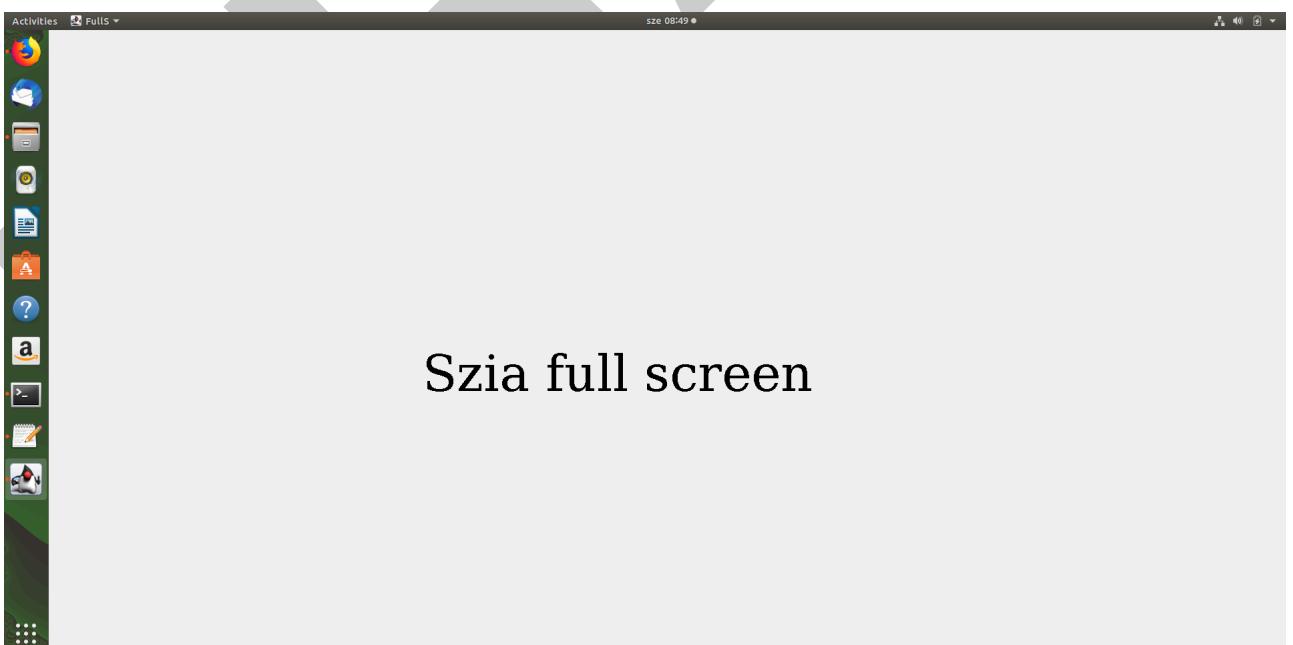
A program elején beállítjuk, hogy ne legyen keret és fejléc se, majd hogy átméretezés se legyen. Ezek után megnézzük, hogy át tudunk-e lépni teljesképernyős módba, ha igen akkor be is lépünk. `graphicsDevice.setFullScreenWindow()` függvényel megnézzük a képernyőnk jellemzőit majd ezt ki is íratjuk. A programtól kikérjük a lehetséges képernyő beállításokat. A játékhoz szereplő képek 1024X768 felbontásra vannak állítva, így a játéknak is ez a felbontás lesz, de mivel a mi képernyőnk felbontás 1920X1080. Ezért a displayt nekünk erre a felbontásra kell állítunk.

Lássuk az eredményt:



Probáltam magamtól írni egy teljes képernyős alkalmazást. Ehez a java.awt használtam. a teljes képernyőt az alábbi módon oldottam meg:

```
public FullS()
{
    setUndecorated(true);
    setSize(Toolkit.getDefaultToolkit().getScreenSize());
    setVisible(true);
}
```



Szia full screen

15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

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

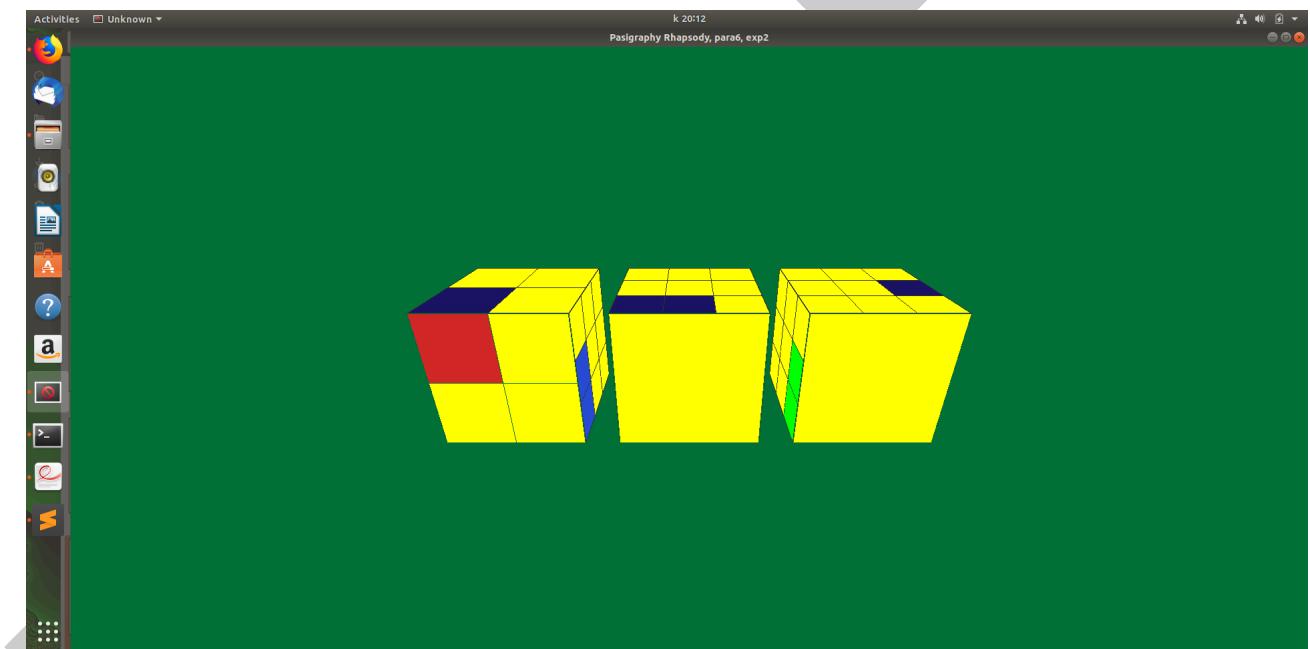
A forrást megtaláljuk tanárúr repojában.

A feladatunk, hogy a programot az adott leírás szerint módosítsuk. De ehez először le kell tudnunk futtatni, mert simán lefutattva hibát fogunk kapni. Ugyanis szükségünk lesz az alábbi kapcsolókra: -lboost_system -lGL -lGLU -lglut. Ezeket "sudo apt-get install libboost-all-dev"-al tudjuk beszerni. Ezek után tudjuk is futtatni a programot.

Most pedig eszközöljünk pár változtatást:

Először a színvilágot változtassuk meg. Ezt a kódban található glColor3f függvény paramétereinek megváltoztatásával tehetem meg. Az irányítást a keyboard() függvényben tudjuk változtatni. Én annyit változtattam, hogy a + és - helyett * és / el fogunk közelíteni és távolodni. A kockánk hátulját akarjuk látni forgatás nélkül, akkor pedig a "v" vel tudjuk ezt megtenni.

Lássuk:



15.4. Paszigráfia Rapszódia LuaLaTeX vizualizáció

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

Feladatunk középpontjába a már az előző feladatban megemlített vis_prel_para.pdf áll. Ezen kell apróbb módosításokat végeznünk. A forrásokat ismét tanárúr repójából vettük. A feladat elvégzéséhez ismernünk kell a LuaLatex nyelvet és kell egy program ami képes kezelni. Én az Online Overleaf LaTeX Editort használtam. A szín világ megváltoztatásához a prelpara.lua fájlban kell módosításokat eszközölnünk. Lássunk erre egy példát:

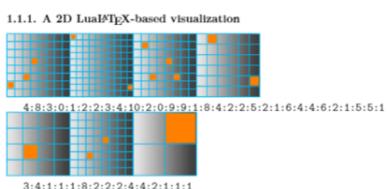
```
"\\shade[topslant, bottom color=green!50,top color=red!70] (0,0) rectangle ←  
+(1,1);",
```

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

Itt adtuk meg, hogy a kocka alja zöld, még a teteje piros legyen. A rács pedig narancssárga. A programon belül a többi esetnél is így tudjuk megváltoztatni a kockák árnyékát és a rácsok színét.

Az eredmény:

Eredeti:



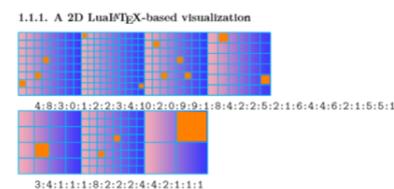
1.1.1. A 2D LuaLaTeX-based visualization

At the moment, we are experimenting with multiple TikZ based visualizations. The first one uses slant¹ and shift transformations.



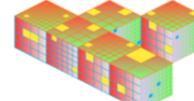
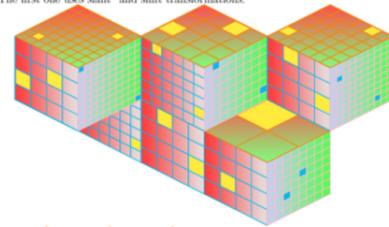
¹The idea of using yslant and xslant to achieve a 3D effect like appearance comes from Stefan Kottwitz's T_Sudoku 3D cube² example <http://www.texample.net/tikz/examples/sudoku-3d-cube/>.

Szerkesztett:



1.1.1. A 2D LuaLaTeX-based visualization

At the moment, we are experimenting with multiple TikZ based visualizations. The first one uses slant¹ and shift transformations.



¹The idea of using yslant and xslant to achieve a 3D effect like appearance comes from Stefan Kottwitz's T_Sudoku 3D cube² example <http://www.texample.net/tikz/examples/sudoku-3d-cube/>.

15.5. Perceptron osztály

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

A Perceptronnal is foglalkoztunk könyünk első fejezetében. A Perceptron a neuron idegejt megfelelője a mesterséges intelligenciában. Ezt tanulásra szokták alkalmazni, ugyanis képes feldolgozni és mintákat megtanulni a bemenetet, ami 0,1 ből áll.

Nézzük magát a programot:

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

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

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

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

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

double value = (*p)(image);

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

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

A program elején szükségünk lesz három darab könyvtárra. Szokásosan az iostream-et. Aztán a megjelenítés miatt az mlp.hpp-re a perceptron miatt lesz szükség, míg a png++/png.hpp a png formátumú képpel való munka miatt kel.

A könyvtárak után egyből jön is a fő függvényünk, amin belül le is foglaljuk a tárhelyet a képünknek amit parancssorból fogunk megkapni. A következő sorban a size válzoóba megadjuk a kép méretét. Ezt két függvény szorzatával fogjuk megadni (hossz*szélesség). Ezek után jön egy perceptron, ami paraméterül kap 3db réteget, Az első rétegben size db neuron lesz, aztán 256 neuron, és az utolsóba csak 1 neuron. Egy új képet hozunk létre, ami a size változóban lévő méretet fogja megkapni. Ez egy mutató lesz. A for ciklus segítségével a beolvasott képpel fogjuk feltölteni, mégpedig úgy, hogy az egyik for ciklus a kép szélességén a másik a magasságán megy végig. A value változónak átadjuk a kapott eredményt amit aztán kiíratunk. Ezek után már csak töröljük a perceptronról és a képet.

Ezek után így kell futtatni a programot:

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

Futtatásnál a Mandelbrot képet adtam neki. Az eredmény pedig:

```
Abort trap (core dumped)
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/4csomag$ ./perc mandel.png
0.500006
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/4csomag$ 
```

16. fejezet

Helló, Stroustrup!

16.1. JDK osztályok

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

Feladatunkban a Boost könyvtárat felhasználva kell kiíratnunk egy C++ program segítségével a JDK összes osztályát. Az első feladat a Boost könyvtárat feltenni, ha még nincs a gépünkön akkor azt az alábbi módon tudjuk megtenni: "sudo apt-get install libboost-all-dev". Azért van erre szükségünk, ugyanis az elérési utat és a kiterjesztést is ezel tudjuk elvégezni.

Most lássuk a C++ programot, amivel ezt megtudjuk valósítani. A feladathoz vegyük elő segítségnek a fénykard forrását.

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

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

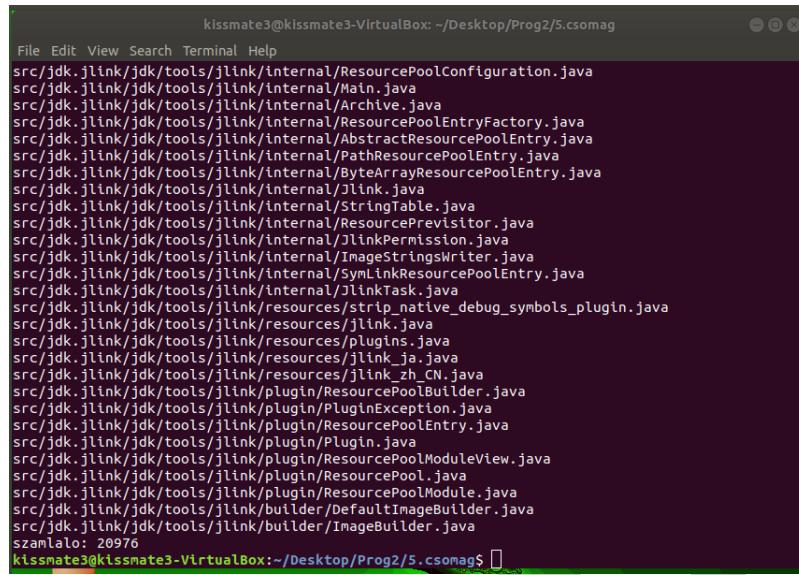
using namespace std;
using namespace boost::filesystem;

int fajlok = 0;

void read_file ( boost::filesystem::path path, std::vector<std::string>  ↵
    acts )
{
```

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

Ahogy fent említettem a kód alapja a fénykard.cpp. A programunk azt fogja csinálnni, hogy kírja a ".java" végződésű fájlokat, és ezek elérési útját, majd a végén hogy összesen hány darab van ezekből. A programban először elvégezzük a szükséges include-kat a boost mappából és amikre még szükségünk lesz (vector, iostream ...stb). Ezek után először is létre kell hozni egy egész típusú változót amivel számolni fogjuk a fájlok számát mégpedig úgy, hogyha van egy ".java" végződésű fájl, akkor növeljük egyel az értékét. A read_file() függvény segítségével fogjuk beolvasni a mappákat és azok tartalmait. If-ek használatával fogjuk megvizsgálni a kiterjesztést. És itt adjuk meg a zelérési utat is. A vectorba fogjuk belevenni az összes .java fájlt az elérési úttal együtt. A legvégén pedig ezeket kiíratjuk. Az eredmény a következő:



```
kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/5.csomag
File Edit View Search Terminal Help
src/jdk.jlink/jdk/tools/jlink/internal/ResourcePoolConfiguration.java
src/jdk.jlink/jdk/tools/jlink/internal/Main.java
src/jdk.jlink/jdk/tools/jlink/internal/Archive.java
src/jdk.jlink/jdk/tools/jlink/internal/ResourcePoolFactory.java
src/jdk.jlink/jdk/tools/jlink/internal/AbstractResourcePoolEntry.java
src/jdk.jlink/jdk/tools/jlink/internal/PathResourcePoolEntry.java
src/jdk.jlink/jdk/tools/jlink/internal/ByteArrayResourcePoolEntry.java
src/jdk.jlink/jdk/tools/jlink/internal/Jlink.java
src/jdk.jlink/jdk/tools/jlink/internal/StringTable.java
src/jdk.jlink/jdk/tools/jlink/internal/ResourcePrevisitor.java
src/jdk.jlink/jdk/tools/jlink/internal/JlinkPermission.java
src/jdk.jlink/jdk/tools/jlink/internal/ImageStringsWriter.java
src/jdk.jlink/jdk/tools/jlink/internal/SymlinkResourcePoolEntry.java
src/jdk.jlink/jdk/tools/jlink/internal/JlinkTask.java
src/jdk.jlink/jdk/tools/jlink/resources/strip_native_debug_symbols_plugin.java
src/jdk.jlink/jdk/tools/jlink/resources/Jlink.java
src/jdk.jlink/jdk/tools/jlink/resources/plugins.java
src/jdk.jlink/jdk/tools/jlink/resources/Jlink_ja.java
src/jdk.jlink/jdk/tools/jlink/resources/Jlink_zh_CN.java
src/jdk.jlink/jdk/tools/jlink/plugin/ResourcePoolBuilder.java
src/jdk.jlink/jdk/tools/jlink/plugin/PluginException.java
src/jdk.jlink/jdk/tools/jlink/plugin/ResourcePoolEntry.java
src/jdk.jlink/jdk/tools/jlink/plugin/Plugin.java
src/jdk.jlink/jdk/tools/jlink/plugin/ResourcePoolModuleView.java
src/jdk.jlink/jdk/tools/jlink/plugin/ResourcePool.java
src/jdk.jlink/jdk/tools/jlink/plugin/ResourcePoolModule.java
src/jdk.jlink/jdk/tools/jlink/builder/DefaultImageBuilder.java
src/jdk.jlink/jdk/tools/jlink/builder/ImageBuilder.java
szamlalo: 20976
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/5.csomag$
```

16.2. Változó argumentumszámú ctor

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

Feladatunk a már az előző csomagban is felelevenített Perceptron. Most az lesz a dolgunk, hogy az előző programunkat átírjuk úgy, hogy egy képet kapunk vissza. A fealadat megoldásához vegyük az előző Perceptronos kódot és azt írjuk át az alábbi módon:

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

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;

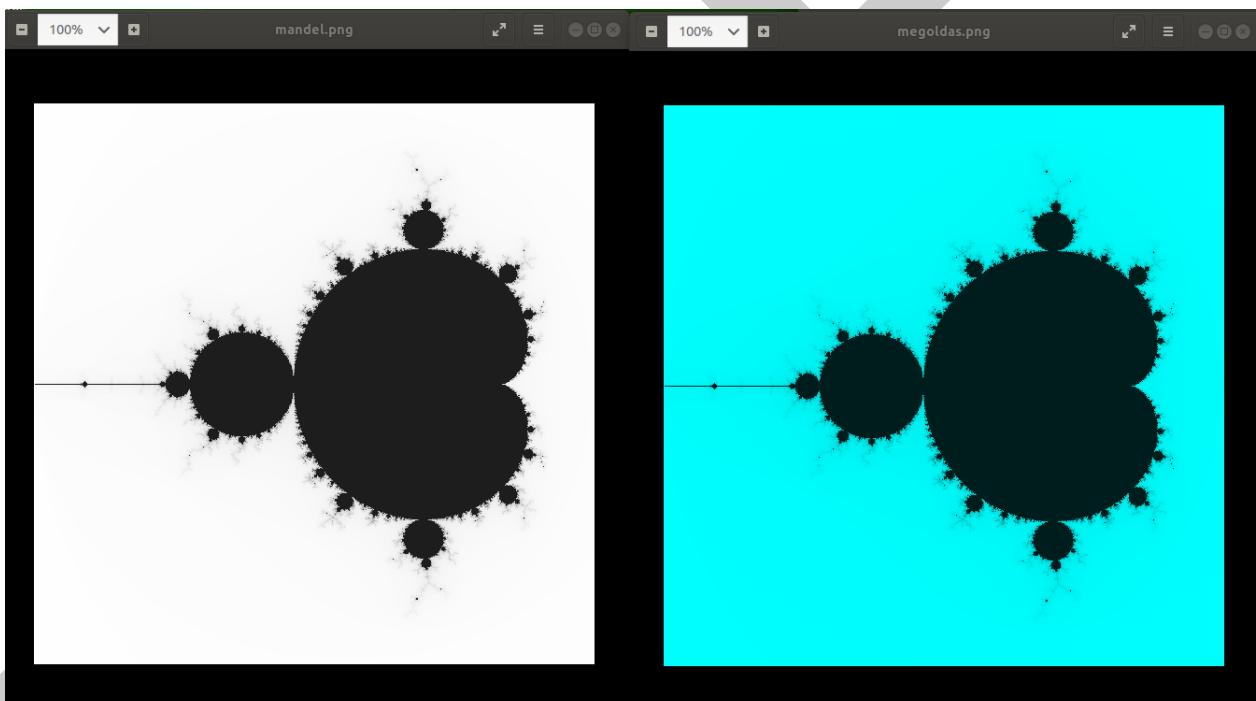
    double* NewKep = (*p) (image);

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

```
png_image.write("megoldas.png");

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

A kódunk nagyrésze már ismerős, most csak az új részeről és a változásokról fogok írni. Azért hogy egy képet kaphassunk, ezért létrehozunk egy double típusú NewKep mutatót. Ezek után két darab forciklussal újból végig megyünk a képünk szélességén és magasságán, közben átadjuk a NewKepnek az eredeti képünk adatait. A "png_image.write()" függvényel pedig "megoldas" néven megjelenítem a képet. Miután a kóddal kész vagyunk, le kell futtatnunk, de ha egyszerüen ./program képpen futtatjuk akkor hibákat kapunk, hiszen szükségünk lesz pár kacsolóra, ezt az alábbi módon kell: "g++ mlp.hpp perc.cpp -o perc -lpng -std=c++14". Sajnos azt tapasztaljuk, hogy még most sem fordul le. Ezt a problémát az mlp.hpp fájlban kell megoldanunk. Megkeressük a fájlban az alábbi sort: "double operator() (double image [])" és átírjuk a következőre: double *operator() (double image []). Ha ezek után futtatjuk és megadjuk neki a mandelbrotos képünket, akkora az eredmény a következő:



16.3. Hibásan implementált RSA törése és összefoglaló

Készítünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: <https://arato.inf.unideb.hu/batfai.m> (71-73 fólia) által készített titkos szövegen.

Az RSA feladatban tutor volt: Garbóczy Vajk.

A feladatunkat kezdjük azzal, hogy elmagyarázzuk, hogy mi is az az RSA. Az RSA egy eljárás ami egy nyílt kúlcstú titkosító algoritmus. Napjainkban ez a legelterjettebb és legismertebb titkosítási eljárása. Az egész világon majdnem minden informatikai vagy kommunikációs rendszerben ezt használják. Ezekben az adatok biztonságáról gondoskodik. Ha a való életben szeretnénk rá példát mondani, akkor említhetjük az internetes bankok rendszerét, a hitelkártyás rendszereket, az internetes kereskedelmet ami napjainkban

aranykorát éli, az emailek hitelesség vizsgálatát , de továbbá még ezzel biztosítják a web serverek és kliensek közötti biztonságot. Az RSA-t 1976 ban Len Adleman, Ron Rivest és Adi Shamir fejlesztettek ki, hármuk nevéből jött az RSA elnevezés. Mivel ezt a feladatot választottam esszének is, ezért kicsit kitérek az alkotókra is pár szóban. A három tudós az RSA kriptoszisztemá feltaalálása miatt több neves díjat is kaptak mint például a párizsi Kanellkasi Elméleti és Gyakorlati díjat, de az informatikában Nobel díjnak hívott ACM Turing díj címzettjei is lettek. A kutatók nem álltak meg az elején, hanem sok más kutatóval azóta is keresik az RSA gyenge pontjait, hogy ezzel is biztonságosabbá tegyék, Eddig azonban egyik támadási módszer sem járt sikерrel, ez annak tutható be, hogy a technika nagyon gyorsan fejlődik, és így a támadásokat az egyre fejlettebb technikákkal védi ki. Az RSA egy kriptográfiai algoritmus, most beszéljünk erről. A kriptográfia egy görög kifejezés aminek jelentése titkosítás, és mivel tujduk, hogy az RSA is egyike ezeknek, tudjuk, hogy ezeket az algoritmusokat titkosításra használják. Az emberiség már a kezdetek óta használ titkosítást, nagy szerepeket játszottak a másdoik világhárúban, de régebben is bizalmas információk továbbadására különböző titkos írásokat vagy titkos beszéd rendszert alkalmaztak. Napjainkban, az online térbe, ahol pénzünket és szeméyles információkat taroljuk, ha biztonságban akarjuk tudni ezeket, akkor szükségünk van az ilyen kriptográfiai algoritmusokra. Ezekben a titkosítás kódolással történik, ha pedig újra titkosítás nélkül akarjuk látni a titkosított adatot, akkor azt dekódolnunk kell. A kódoláshoz kriptográfiai kulcsokat használunk ami egy megadott hosszúsággal rendelkező bit sorozat vagy egy szám. A kriptográfiai algoritmusokat két részre osztjuk, vannak a szimetrikus és az asszimetrikus kúlcsoval rendelkező titkosítási algoritmusok. Az RSA asszimetrikus algoritmus. A szimetrikusról azt kel tudni, hogy egyetlen egy kúlcst van amit a kódolásra és a dekódolásra használunk. Míg az asszimetrikusnál egy nyilvános és egy magán kúlcossal rendelkezünk. A kódolás a nyilvános kúlcossal történik, de a dekódolás a magánkulccsal. Most pedig nézzük az RSA titkosító algoritmus működését. Az első lépésünk, hogy két nagyon nagy prímet véletlenszerűen kiválasztunk. Az egyik prím legyen "p" a másik meg legyen "q" . Fontos rényező, hogy a prímek legalább 100 decimális számjegyük legyenek. A második lépés a nyilvános és a magán kulcs modulusának kiszámítása az alábbi módon: $N=p \cdot q$. Ezek után az Euler-féle ϕ függvénynek kell az értékét kiszámítani N-re az alábbi módon: $\phi(N)=(p-1)(q-1)$. A negyedik lépésben válasszunk egy egész számot és jelöljük e-vel. e-re teljesülni kell , hogy $1 < e < \phi(N)$, továbbá hogy e és a $\phi(N)$ Inko-ja 1 legyen. Az e nyilvános lesz mivel a nyilvános kulcs kitevője. Tehát az N modulusból és az e nyilvános kitevőkből áll össze a nyilvános kulcs. A titkos kulcsnak az 5 lépésben szükségünk lesz egy d re, amire a kungruenciának teljesülnie kell. A képlet a következő: $d = e^{-1} \pmod{\phi(N)}$ ahol k egész szám. Így N modulus és d kitevőkből fog állni a titkos kulcs. Most pedig nézzük a kódot:

```
import java.util.Scanner;

public class RSA {
    public static void main(String[] args) {
        KulcsPar jszereplo = new KulcsPar();
        String szoveg;
        Scanner sc = new Scanner(System.in);
        System.out.println("Adja meg a titkosítani kívánt szöveget: ");
        szoveg=sc.nextLine();
        byte[] buffer = szoveg.getBytes();
        java.math.BigInteger[] titkos = new java.math.BigInteger[buffer. ←
            length];
        for (int i = 0; i < titkos.length; ++i) {
            titkos[i] = new java.math.BigInteger(new byte[] {buffer[i]}));
            titkos[i] = titkos[i].modPow(jszereplo.e, jszereplo.m);
        }
        for (java.math.BigInteger t : titkos) {
```

```
        System.out.print(t);
        System.out.println();
    }
    for (int i = 0; i < titkos.length; ++i) {
        titkos[i] = titkos[i].modPow(jSzereplo.d, jSzereplo.m);
        buffer[i] = titkos[i].byteValue();
    }
    System.out.println("\n" + new String(buffer));
}
}

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

A forráskódra nézve látjuk, hogy két főbb osztályunk van, az RSA és a Kulcspar. Az RSA osztályunkban fog történi a kódolás és a dekódolás is. Vegyük jobban szemügyre az RSA függvényt, először példányosítunk benne, aztán egy Scanner segítségével beolvassuk a titkosítandó szövegünköt. Ezek után következik az a egy BigInteger osztály, hogy nagy számmal tudjunk dolgozni. Majd következik a fent leírt számítási folyamat. Ehez a KulcsPar osztályra is szükségünk van. Ebben lesz a random() függvény, amivel kiválasztjuk a random prímeket. A számítások végén megkapjuk a titkos és a nyílt kulcsot is.

Activities Terminal k 13:33 ●
kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/5.csomag\$ java RSA
Adja meg a titkosítani kívánt szöveget:
kutya
94380736921605961946331709412117341617364220693810407899622298429727497427050532484699020230261202806283994965785585815
7413495464484830776598948270166810354571708746598664122758451843946919515961708124113099019657256393464462081722420766521402528789
2980512343354548432789778895083116983379843547872349354614979862591542446922989446527751359508625973780876744356679379
488610853439339861698338071758081798892116736089689266017136350884028480596695351735535432074738956294380087931230283
886286253808698706528072615784477260720322179592947164974446618177572912475398412771027292965888161245335182062536293808764899
27998198475854216734570164448948255369970465917253471822353804903625123635110340527405413264975003865208831939257098189
88194432346790448125456680254852741916
162667734957761893705181219123134196673531071613875102565534546708934823636576667861768175259695263785793070717108369392192
105258625984767323343158206989443384998234962849290084198346787214451321366412468805363031259828057297021108958268401376948
93985985720512116996823378985010533227506407358997458199449756037613811496973406812771974052091554279625820312230088185717985354303
184375331663907219906140242340575459692164833863119364754891239986480247580597489215063614514859301068264652558395320
70434389182415146698623378985010533227506407358997458199449756037613811496973406812771974052091554279625820312230088185717985354303
7067351563154777205982101135877654570972547066884983847818925654882032424878031237388493152312970813737628508985930536761156
0697971725027192115545010623455655031795
12823074199511521736305648887072589639216532089755567146910872617748301886046989836632510181604991114458275125091710462
291243419605336375170517764983826384738987317968877784686737363545974894404031749366730375375580883317371949458570181913798714776000315
520236156216744487807737343519972288443602352154863156343898539640005091524136867825958494984353388767114494
835744278751.0201367718515281387457997838681399533603545974894404031749366730375375580883317371949458570181913798714776000315
553080696006077373675971839877351844184893910337603540247870972894781413076188593423056356702651092194886962484817328621
97782684810865076307279139805148193542537237036597166666755986048279127308778207741314094544401488934623368119742567840932575
2852386881814839031941326392363927682
12802534301912805862001656414747504056197568243115348927736845185337640924616120127287034084991611894743622730722326071414032
098014391841228867158369751827984002040456581500887561411785922425887002961455644680288845235847937778035070189697086
→ ->
08211395354610276337921442993995714982519242017251757114228181362480498472909891270369242525197848376228729524450255134591856488
1133989450776281825951707547569991366254429867944367978731440785949855253535949791737526218693769887538072283960566836763715
884720153655995761147487375826026933568190630278659215334595358067553083677329228590002867364989644049349986226563807
7021586451134226138108798132641853185974
8445562052435965983233687856967971191158588764554542761428388196633736577282330175642578480457671667660021283154466709295
70887926216779335218337253412303651524483882351192173707279511946257334653037532847007148362833943494607
01611298904125756727252946157893521516497470452656453692377114737408274523957781850930062639309615031044
8137746913422310313954914055376182946596973947445882849421882427929462556464221616463092394436165714745242591297031478
7103708937349849862657637816899488087725044226358759768150632228800775648805969344101095824163197942969969341167
402545659955723101540154638088145852628652536758686191527361743329894126529227526595527445267905380105010110499612191750585
86146352504759485292757858459564401451
kutya
kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/5.csomag\$

17. fejezet

Helló,Gödel!

17.1. STL map érték szerinti rendezése

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

Feladatunk a mapot érték szerint rendezni. A map egy olyan asszociatív tároló, mely kulcs-érték párokat fog tartalmazni. Nem tudunk duplukált kulcsokat tárolni benne, ugyanis csak egy érték tartozhat egy kulcsnak. Az STL pedig a Standard Template Library, ez a szabványos C++ nyelv könyvtár. Ez biztosítja a megfelelő hash() függvényeket. A feladat azért érdekes, mivel a map kulcs szerint rendez. Tehát hogy érték szerint tudjuk rendezni szükségünk lesz egy kis változtatásra. A megoldáshoz a fénykardot hívtam újbol segítségül. Ebből kiindülva tekintsük meg a kódot:

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

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

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;
}

int main(){

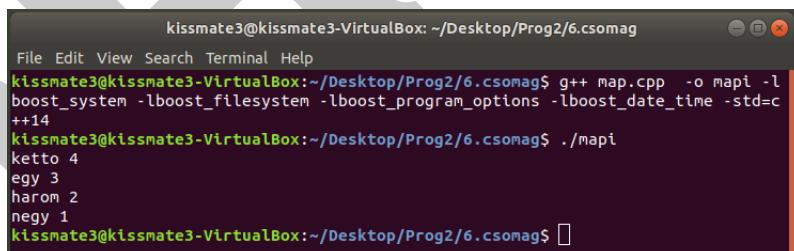
std::map<std::string, int> map;
map["egy"] = 3;
map["kettő"] = 4;
map["harom"] = 2;
map["negy"] = 1;

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

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

}
```

Most pedig lássuk a kódot. Az int main() -be létrehozzuk a mapunkat ami egy string és egy int értéket fog kapni (esetünkben például: "egy", 3). A kód alapján láthatjuk, hogy az érték szerinti rendezést úgy oldottuk meg, hogy a map kulcs-érték párokat egy pair vektorba rakjuk. Erre azért van szükség, mivel a vektoron belül már képesek vagyunk érték szerint csökkenőbe rendezni a párokat. A végén pedig egy speciális for ciklussal sorba kiíratjuk a vektorunkat. Megoldás:



```
kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/6.csomag
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/6.csomag$ g++ map.cpp -o mapi -lboost_system -lboost_filesystem -lboost_program_options -lboost_date_time -std=c++14
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/6.csomag$ ./mapi
kettő 4
egy 3
harom 2
negy 1
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/6.csomag$
```

17.2. GIMP Scheme hack

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

A feladatot a GIMP képszekesztővel csináltam ami egy pixelgrafikus képszerkesztőprogram. A Gimp egyik nagy előnye a szkriptelhetőség. A programban ez a Script-Fu. A Gimpben belül, a sémaknak a nyelve a Lisp, ezzel a nyelvel már találkoztunk a könyvünk során. Az elkészítendő feladathoz Bátfai tanárúr már készített egy sémát, mi is ezt fogjuk használni alapnak. De, hogy ezt a sémat tudjuk használni, ahoz a scriptet amit tanár úr megadott, be kell illesztenünk a scriptek mappába. Most pedig lássuk a kódot:

```
(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-NONE FALSE TRUE 5 .1 TRUE 500 500 (+ (+ ←
        500 (/ textfs-width 2)) 8) 500)

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

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

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

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

```
)  
  
; (script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Sans" 100 1000 1000 '(255 ←  
0 0) "Deep Sea")  
  
(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"  
)
```

Kezdjük a magyarázatot a kód aljáról, ahol regisztráljuk a Create menübe a scriptet. Utánna jön maga a menü amit, az előbb regisztrált menüpontal hozunk elő. Itt a képünk paramétereit tudjuk megadni. Ilyen a Text, amivel a háttérben szereplő szöveget adjuk meg, a Text 2 amivel a fő szöveget adjuk meg. Képesek vagyunk különböző betűtipusokat használni, és megadni a betű méretét. Továbbá itt adjuk meg a kép szélességét és magasságát, A fő szöveg színét és a háttér szöveg kinézetét. Ha ezt mind megadjuk, utánna le is tudjuk generálni a képünket. A kódot nézve a menü felépítése a következő: Megadjuk a típust például SF-String. Ezek után a Típus nevét: Text. Végezetül, hogy mit tartalmaz a szöveg. Ez egy alap érték lesz, tudjuk majd módosítani. Ez után jön a scriptünk ami a kép generálást fogja végrehajtani. Az első réteget a Texből hozzuk létre úgy hogy minden szögből ugyan azt látjuk, aztán erre kerül a második szöveg, amit rállesztünk az első rétre. Lássuk is az eredményt:



17.3. Alternatív Tabella rendezése

Mutassuk be a https://progpter.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang

A feladatunk most kedvez a foci rajongóinak, ugyanis a 2018-2019 Nb1 es tabella alternatív tabelláját kell megalkotni. Ehez Bátfai tanárúr 2011 alternatív tabellájának kódját dolgozzuk át. Először is beszéljük meg, hogy mi is az az Alternaty tabella. Tudjuk, hogy a rendes tabella úgy épül fel, hogy egy győzelem 3, a döntetlen 1 míg a vereség 0 pontot ér, mindegy kivel játszották a meccset. Viszont az alternatív tabella figyelembe veszi, hogy melyik csapattal szemben értél el győzelmet, döntetlent vagy vereséget. Mielőt neki látnánk a feladatnak szükségünk lesz egy kis frissítésre, mivel az NB1-ben voltak azóta változások. 16 csapatról 12 re csökkent, és már nem is ugyan azok a csapatok szerepelnek, mivel minden évben kiesik az utolsó kettő. Először a "Wiki2Matrix.java" kódban lévő mátrixot írjuk át a tavalyi keresztáblázat alapján. A mátrixot az alapján töltjük ki 0,1,2,3 számmal, hogy milyen színnel van a táblázatban. Az üres 0 pontot, a zöld 1 pontot, a sárga 2 pontot és a piros 3 pontot ér, ez alapján a mátrix így fog kinézni (a forrás wikipédia):

```
int [][] kereszt = {
    { 0, 1, 1, 3, 1, 1, 2, 1, 1, 1, 2, 3 },
    { 1, 0, 1, 1, 2, 1, 2, 2, 1, 1, 2, 3 },
    { 1, 1, 0, 3, 1, 2, 2, 1, 2, 2, 3, 3 },
    { 1, 2, 1, 0, 1, 1, 1, 2, 1, 1, 2 },
    { 3, 3, 2, 1, 0, 3, 3, 3, 3, 1, 2, 3 },
    { 3, 1, 2, 3, 1, 0, 3, 1, 2, 1, 3, 2 },
    { 3, 2, 1, 3, 1, 2, 0, 3, 1, 1, 2, 1 },
    { 2, 3, 1, 3, 1, 3, 2, 0, 3, 1, 1, 3 },
    { 2, 1, 3, 3, 2, 1, 1, 1, 0, 1, 2, 3 },
    { 1, 3, 1, 1, 1, 2, 1, 3, 2, 0, 3, 1 },
    { 2, 1, 1, 2, 1, 1, 2, 3, 2, 1, 0, 1 },
    { 1, 2, 3, 1, 1, 1, 3, 2, 3, 1, 0 }
};
```

Aztán frissítenünk kell a "AlternativTabella.java" forrásban a csapatok neveit, az elérte pontokat és a keresztáblázatnak megfelelően a csapat neveit. Továbbá a "Wiki2Matrix.java" ban kapott eredményt is be kell illesztenünk a double[][] L = { } közé. A feladatunk része volt még, hogy beszéljünk a java.lang Interface Comparable<T> szerepével. Erre a rendezés miatt lesz szükségünk, ugyanis a sort() függvény kulcsként alkalmazza sorbarendezésnél. Előnye, hogy nem kell összehasonlítani külön külön. Most pedig nézzük hogy is néz ez ki:

```
double [][] L = {
    {0.0, 0.07692307692307693, 0.0625, 0.0, 0.1, 0.125, ←
     0.1333333333333333, 0.15384615384615385, 0.1333333333333333, ←
     0.06666666666666667, 0.15384615384615385, 0.0, },
    {0.0833333333333333, 0.0, 0.0625, 0.25, 0.1, 0.0625, 0.1333333333333333, ←
     0.15384615384615385, 0.06666666666666667, 0.1333333333333333, ←
     0.07692307692307693, 0.1, },
    {0.0833333333333333, 0.07692307692307693, 0.0, 0.0, 0.1, 0.125, ←
     0.06666666666666667, 0.07692307692307693, 0.1333333333333333, ←
     0.06666666666666667, 0.0, 0.1, },
    {0.1666666666666666, 0.07692307692307693, 0.125, 0.0, 0.05, 0.125, ←
     0.1333333333333333, 0.15384615384615385, 0.1333333333333333, ←
     0.06666666666666667, 0.15384615384615385, 0.1, },
    {0.0, 0.07692307692307693, 0.0625, 0.125, 0.0, 0.0, 0.0, ←
     0.06666666666666667, 0.06666666666666667, 0.07692307692307693, 0.0, },
    {0.0, 0.07692307692307693, 0.125, 0.0, 0.1, 0.0, 0.0666666666666667, ←
```

```
0.15384615384615385, 0.0666666666666667, 0.1333333333333333, 0.0, 0.1, ←
},
{0.0833333333333333, 0.15384615384615385, 0.125, 0.0, 0.1, 0.125, 0.0, ←
0.07692307692307693, 0.0666666666666667, 0.0666666666666667, ←
0.15384615384615385, 0.1, },
{0.0833333333333333, 0.07692307692307693, 0.0625, 0.0, 0.1, 0.0, ←
0.1333333333333333, 0.0, 0.0, 0.1333333333333333, 0.15384615384615385, ←
0.1, },
{0.0833333333333333, 0.07692307692307693, 0.0625, 0.125, 0.1, 0.125, ←
0.0666666666666667, 0.15384615384615385, 0.0, 0.1333333333333333, ←
0.15384615384615385, 0.1, },
{0.0833333333333333, 0.0, 0.125, 0.125, 0.05, 0.0625, 0.0666666666666667, ←
0.0, 0.0666666666666667, 0.0, 0.0, 0.2, },
{0.1666666666666666, 0.15384615384615385, 0.125, 0.125, 0.1, 0.125, ←
0.1333333333333333, 0.0, 0.1333333333333333, 0.1333333333333333, 0.0, ←
0.1, },
{0.1666666666666666, 0.15384615384615385, 0.0625, 0.25, 0.1, 0.125, ←
0.0666666666666667, 0.07692307692307693, 0.1333333333333333, 0.0, ←
0.07692307692307693, 0.0, }
};

// az eredeti tabella sorrendje, 2011.03.11
String[] csapatNevE = {
    "Ferencváros",
    "Vidi",
    "Debrecen",
    "Honved",
    "Újpest",
    "Mezőkövesd",
    "Puskás Akadémia",
    "Paks",
    "Kisvárda",
    "Diosgyor",
    "MTK",
    "Haladas"
};
// az eredeti tabella pontjai, 2011.03.11
int[] ep = {
    74,
    61,
    51,
    49,
    48,
    44,
    40,
    39,
    38,
    38,
    34,
    30,
```

```
};

// az L matrix kiszövtsé sekori sorrend (a blogra kettibontottban ez a ←
// Wikis keresztőbla), 2011.03.11
String[] csapatNevL = {
    "Honved",
    "Debrecen",
    "Diosgyor",
    "Ferencvaros",
    "Haladas",
    "Kisvarda",
    "Mezokovesd",
    "MTK",
    "Paks",
    "Puskas Akadémia",
    "Ujpest",
    "Vidi"
};


```

```
File Edit View Search Terminal Help
Csapatok rendezve:
|-
| Ferencvaros
| 74
| Ujpest
| 0.1043
|-
| Vidi
| 61
| Ferencvaros
| 0.1038
|-
| Debrecen
| 51
| Vidi
| 0.1018
|-
| Honved
| 49
| Debrecen
| 0.1001
|-
| Ujpest
| 48
| Paks
| 0.0955
|-
| Mezokovesd
| 44
| Mezokovesd
| 0.0863
|-
| Puskas Akadémia
| 40
| Honved
| 0.0815
|-
| Paks
| 39
| MTK
| 0.0723
|-
| Kisvarda
| 38
| Puskas Akadémia
| 0.0695
|-
| Diósgyőr
```

A JDK forrás:

```
/**
```

```
* Sort a list according to the natural ordering of its elements. The ←
 * list
 * must be modifiable, but can be of fixed size. The sort algorithm is
 * precisely that used by Arrays.sort(Object[]), which offers guaranteed
 * nlog(n) performance. This implementation dumps the list into an array ←
 *
 * sorts the array, and then iterates over the list setting each element ←
 * from
 * the array.
 *
 * @param l the List to sort (<code>null</code> not permitted)
 * @throws ClassCastException if some items are not mutually comparable
 * @throws UnsupportedOperationException if the List is not modifiable
 * @throws NullPointerException if the list is <code>null</code>, or ←
 * contains
 *      some element that is <code>null</code>.
 * @see Arrays#sort(Object[])
 */
public static <T extends Comparable<? super T>> void sort(List<T> l)
{
    sort(l, null);
}
```

17.4. Gengszterek

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

Feladatunk újból a Robotatú Világnajnokságban fog végbemenni. De a feladat lényege a Lambda kifejezések. Ezeket akkor használjuk, ha egy programban csak egyszer akarjuk lefuttatni, tehár inline függvényt hozunk vele létre. Mivel csak egyszer futnak le nem rendelkeznek névvel. A visszítérési típus a fordító határozza meg. A lambda kifejezés definiálása a következő:

```
[] (paraméterek) -> visszatérési érték típus
{
    metódusok
}
```

Az OOWC-ben szereplő kódrészletünk az alábbi:

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

Láthatjuk, hogy a rendezést a sort() függvénytel végezzük. Ahogy látjuk, a sor() függvényünkön belül van a lambda kifejezés ami egy x és egy y objektumot vár. A lambda fogja vizsgálni és összehasonlítani azt,

hogy az x vagy az y nevű gengszterünk van e közelebb a rendőrhöz, és ezek alapján fogja sorba rendezni a sort() a gengszetereket. Tehát első helyen az a gengszter lesz aki a legközelebb fog lenni a rendőrhöz.

DRAFT

18. fejezet

Helló,!

18.1. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/SamuLife>

Feladatunkban a SamuCam-at fogjuk elő venni és feléleszteni és ezzel megnézni, hogy kezeljük a kamerát. Ez azért érdekes, mert itt nem szöveges formátummal foglalkozunk, hanem képekkel, ráadásul élőben. A SamuCam programunk argumentumként kap egy IP címet, és arról az IP címről fogja megnyitni a kamerát. Mivel a SamuCam egy QT program, nézzük meg, hogy keltettük életre. Először is a forrás fájlokat letöljük Bátfai Norbert tanárúrnak a repójából. Ezek után, még a futtatáshoz szükségünk lesz, egy XML fájlra, amit a fájlok közé kell tennünk. Mivel mi nem az IP címről, hanem a számítógépünk webcameráját akarjuk használni, ezért a SamuCam.cpp ben át kell írnunk a videoCapture.open() függvényt, VideoStream helyett 0 írunk:

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

Ezután, ahogy a QT programoknál szoktuk, qmake parancsal SamuLife.pro fájlt futtatjuk. Majd a make el, létrehozzuk a futtatható fájlt. Én VirtualBoxot használok, ezért nem műköött nekem, mivel nem ismerte fel a webcamot. Ezt egy VirtualBox bővítménnyel orvosoltam. Ezek után a program tökéletesen futott.

Most beszéljünk a kódrol:

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

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

void SamuCam::run()
{
    cv::CascadeClassifier faceClassifier;

    std::string faceXML = "lbpcascade_frontalface.xml";
```

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

cv::Mat frame;
```

Az alábbi részben történik a kamera meghívása, mégpedig a openVideoStream() függvény segítségével. Az előbb említett videoCapture.open (0); el nyitjuk meg a saját webcameránkat. A videoCapture.set() fügvények segítségével tudjuk megadni hogy a kamera hány FPS el működjön, továbbá a szélességet és a magasságot. Esetünkben 10 lesz az FPS. Ezek után jön a run() függvény. Itt fogjuk alkalmazni a Classifier, amivel egy arc elemzését fogjuk elvégezni. Ahogy itt látszik: std::string faceXML = "lbpcascade_frontalface.xml"; megkapja a fentiekben beszélt XML-t, ugyanis ez tartalmazza a megfelelő adatokat, hogy a program képes legyen felismerni, hogy arc objektum van a képen.

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

        if ( !frame.empty() )
        {

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

            std::vector<cv::Rect> faces;
            cv::Mat grayFrame;

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

            faceClassifier.detectMultiScale ( grayFrame, faces, 1.1, 4,
                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 );

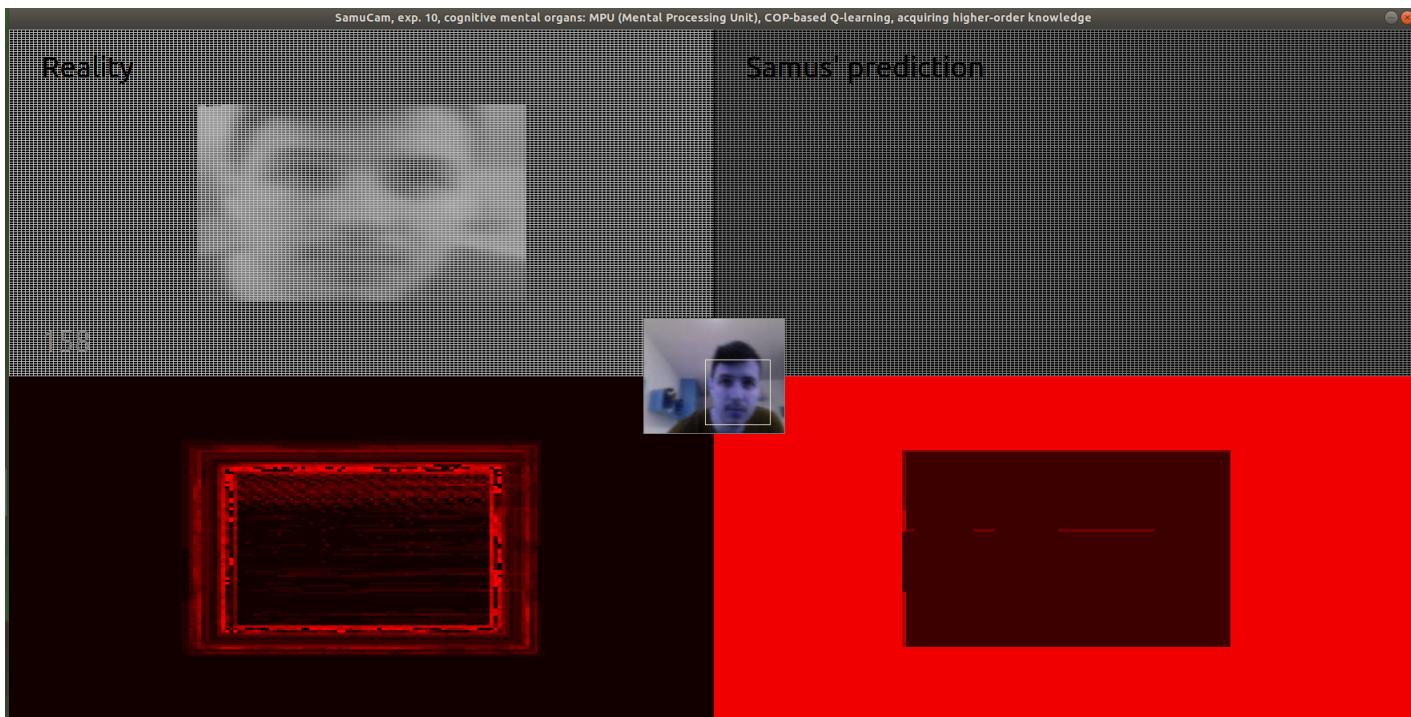
}

QThread::msleep ( 80 );

}

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

Mindezek után egy while ciklus fog futni. A ciklus feltételében az van, hogy addig fut a ciklus amíg a kameránk megvan nyitva. A cikluson belül még egy ciklus amiben egy if függvény segítségével megnézzük hogy van a képkocka, ha van akkor azt a resize() függvénnyel átméretezzük a megadott méretre. A COLOR_BGR2GRAY a képeket szürkévé alakítjuk át és eltároljuk a grayFrame-be. Utánna hisztogram kiegyenlítést hajtunk végre. Az egyenlítés után következik az arckeresés a bemeneti képeken, ezt a detectMultiScale() függvénnyel végezzük. Ha a program talál egy arcot, akkor azt eltárolja egy rectangle listában. Az első arcot amit felismer, abból létrehoz egy QImage-t. A kép adatait a SamuBrain kapja meg és dolgozza fel. Ezután egy keretet hozunk létre az arc körül. Ebből a SamuLifenak fogjuk tovább adni az újabb Qimaget. És ez fogja lefrissíteni a webcam képet ami meg fog jelenni. QThread::msleep (80); sor jelenti azt, hogy 80 milliszekundumonként történik a framek beolvasása. Most pedig lássuk az eredményt:



18.2. BrainB

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

A BrainB már ismerős program számunkra, ugyanis az előző fejezetben már találkozzunk vele. Azt tudjuk hogy a BrainB egy benchmark, amik teljesítmény vizsgáló programok. Az alábbi program minket játékosokat vizsgál és hasonlít össze. minden játékost a megszerzett pontjai alapján hasináltunk össze. A játék menetét, és hogy miként zajlik azt az előző fejezet 7 ik csomagjában az eről szóló fejezetben elolvashatjuk. Most a program a Qt slot-signal mechanizmust fogjuk vizsgálni. A slotok és a signalok az objektumok közötti kommunikációra szolgálnak. A QT-nek ez a mechanizmus egy központi elemem amit a Qt meta-object system tesz lehetővé. Egy signalt egy objektum bocsát ki, ha valamilyen változás történik. A slot-ot pedig akkor hívják meg, ha egy hozzá kapcsolt jel kerül kibocsátásra. A slot egy tagfüggvény. A Signalhoz való kapcsolást pedig a connect függvény teszi lehetővé. A szignatúrájuknak meg kell egyezni. A BrainB-ben két ilyennel találkozhatunk

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

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

Ebben a példában a jelentése az, hogyha a brainBThread-n belül bekövetkezik heroesChanged signal, akkor a updateHeroes függvényt lefuttatja. A második connect ugyan ez, itt viszont ha a endAndStats signal következik be. Ugye ez azt jelenti, hogy lejárt a játék idő. Ekkor megkapjuk a háték eredményét egy debug üzenetben. A játék kinézete az alábbi:



18.3. FOOCWC Boost ASIO hálózatkezelése

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

Feladatunk a scanf szerepének és használatának bemutatása, a megadott forráskódon keresztül. A scanf() egy függvény az std bemenetről olvas be nekünk a függvényben megadott paraméterek szerint. Haználatát úgy írnám körül, mint a már ismert printf függvény ellentétes irányú függvénye, azaz ez beolvas. Lássuk is, a kódcsipetünket:

```
{ STAT } { WS } { INT }          {  
    std::sscanf(yytext, "<stat %d", &m_id);  
    m_cmd = 1003;  
}
```

A kódban nem a scanf hanem a sscanf szerepel, amiről tudjuk, hogy formázott stringeket olvas be. A függvény működése az alábbi, először a szöveget kell megadni amit vizsgálni akarunk. ez a példánkban a yytext lesz, utánna követi a "<stat %d". ez lesz a formátum ami alapján kiolvassa a megadott karaktersorozatot. Utánna a m_id argumentum, amibe elhelyezük a konvertált értékeket. Tehát álltalánosan az scanf és a sscanf így néznek ki.

```
//scanf:  
int scanf(char *formatum, ...)  
  
//sscanf:  
int sscanf(char *string, char *formatum, argu1, argu2, ...)
```

A konzerviós karakterei a scanf-nek: A "d" az a decimális egész, az "i" az az egész szám, "o" az oktális szám, "u" az előjel nélküli egész, "x" az a hexadecimális szám, ezeknek az argumentum típusai az int* .

A char* típusuk pedig a a "c" ami a karakter, az "s" ami karaktersorozat. Ez nem az összes konverziós karakter. További karaktereket találhatunk az interneten, Képesek vagyunk a scanf-en belül szűrésre is. Ezt [] belül tudjuk megadni, például [^abc] azt jelenti, hogy az inputból minden beolvas kivéve az a,b,c karaktereket. A [a-zA-Z0-9] pedig a kisbetűket és a számokat nem olvassa be.

18.4. OSM térképre rajzolása

Ahogy a cím is jelzi, egy OSM térképet fogunk készíteni. A feladatot az Android Studio-ban végeztem el, a forrás és segítség a Programmer World youtube csatornától van. A programunk egy GPS helymeghatározó és nyomkövető program lesz. Ez annyit jelent, hogy a program a térképen megjelöli a helyet, hogy hol vagyunk, és másodpercenként követi a mozgásunkat, hogy merre megyünk és a jelölőkkel kirajzol egy utat. Viszont, hogy egy ilyen programot életre tudunk kelteni, nem elég a forrás kód, igényelnünk kell google developers oldalon egy API key-t. Ezután az Andorid Studion belül létrehozunk egy MapsActivity projectet java nyelven. Projektben a MapsActivity.java program mellett egy google_maps_api.xml is létre jött. Ahogy az xml neve is jelzi a megfelelő helyre be kell másolnunk a fent említett API keyt-t.

```
<string name="google_maps_key" translatable="false" templateMergeStrategy="preserve">APIKEY</string>
```

Most pedig lássuk a java forráskódunkat. A szükséges fájlokat az Android Studio beimportálta nekünk a project létrehozásánál. A kód az alábbi:

```
private final long MIN_TIME = 1000;
private final long MIN_DIST = 5;
//...

@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(this, R.raw.mapstyle));
    mMap.getUiSettings().setZoomControlsEnabled(true);

    locationListener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            try {
                LatLng = new LatLng(location.getLatitude(), location.getLongitude());
                mMap.addMarker(new MarkerOptions().position(latLng).title("Mate"));
                mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
            } catch (SecurityException e) {
                e.printStackTrace();
            }
        }
    }

    @Override
```

```
public void onStatusChanged(String provider, int status, Bundle ←
    extras) {
}

@Override
public void onProviderEnabled(String provider) {
}

@Override
public void onProviderDisabled(String provider) {
}

};

locationManager = (LocationManager) getSystemService(←
    LOCATION_SERVICE);
try
{
    locationManager.requestLocationUpdates(LocationManager.←
        GPS_PROVIDER, MIN_TIME, MIN_DIST, locationListener);
}
catch (SecurityException e)
{
    e.printStackTrace();
}
}
}
```

A fő függvényben azzal kezdjük, hogy létrehozunk private változókat. Ezekből a MIN_TIME=1000 -al adjuk meg, hogy 1 másodpercenként frissítsük helyzetünket még a MIN_DIST = 5 -el azt mondjuk, hogy 5 méterenként tegyünk le markert.A locationManager függvényünk figyeli az elmozdulást, ha érzékeli azt, akkor a megfelelő adatokat átadja a locationListener-nek ami a megfelelő helyre leteszei a makert. Én az alap google helymeghatárizón végeztem pár modosítást. Az első modosítást a térkép stílusán végeztem. Alapból a standart style van használva, de én az Aubergine style-t választottam. Van egy oldal, a mapstyle.withgoogle.com. Itt személyre lehet szabni a google mapsat, lehet állítani a témaját, hogy a térképen mik jelenjenek meg (utak, címek, jelek). Ezeket a beállításokat az oldal generálja nekünk egy json fájlba, így nekünk annyi a tehendőnk, hogy ezt hozzá csatoljuk a projecthez. A java forráskódban is hozzá kell kapcsolni, az alábbi módon: lásd a kódcsipetet. A setMapStyle függvényben beállítjuk, hogy a json dokumentumban megadott módon jelenjen meg a térkép. Az én json fájl nevem a mapstyle.json volt.

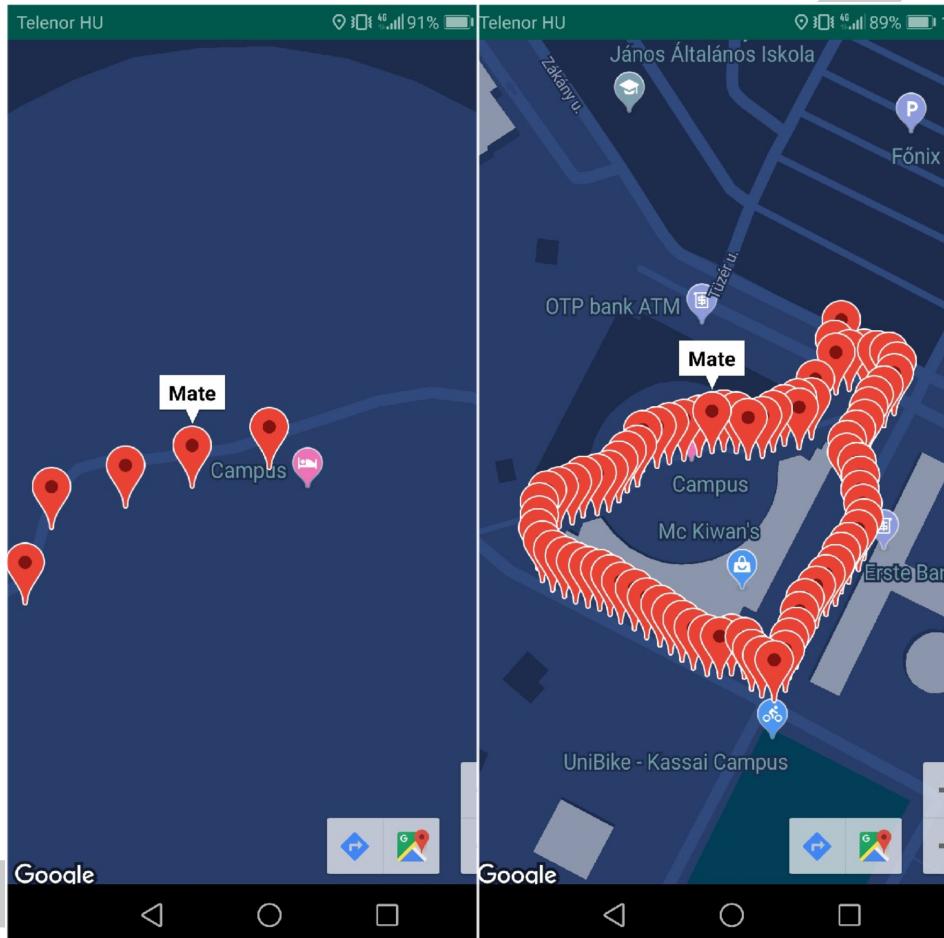
```
mMap.setMapStyle(MapStyleOptions.loadRawResourceStyle(this, R.raw.mapstyle) ←
);
```

Továbbá én még az alkalmázást két gombal is bővítem, amikkel lehet a térképen zoomolni. Ehez egy activity_maps.xml en belül létre hoztam a gombokat, megadtam a helyzetüket, a kinézetüket, majd a Java forrásban az alábbi módon megadjuk, hogy a gombok a gomb lenyomására hogyan reagáljanak. Én a Zoomolást a CameraUpdateFactory függvénytel értem el. A közelítést vagy távolítást pedig ifekkel oldottam meg egy függvényen belül.

```
public void onZoom(View view) {
```

```
if(view.getId()==R.id.zoomin) mMap.animateCamera( ←  
    CameraUpdateFactory.zoomIn());  
  
if(view.getId()==R.id.zoomout) mMap.animateCamera( ←  
    CameraUpdateFactory.zoomOut());  
}
```

Ha mindezzel kész vagyunk egy kis séta után így néz ki aaz eredmény:



19. fejezet

Helló, Lauda!

19.1. Android Játék

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

Ebben a feladatban egy egyszerű Android játékot fogunk írni. A feladat megoldásához az Android Studio programot használjuk. Ez egy ingyenes program ami integrált fejlesztő környezetet biztosít android rendszerhez. A játék amit én gondoltam elkészíteni egy egyszerű autós játék, amit "Flappy Car"-nak neveztem el, az autó mozgása alapján. A játék lényege, hogy az autóval elkapjuk a sárga pöttyöket, és kikerüljük a piros pöttyöket. Egy játékban 3 élet van, azaz a 3 piros pötty elkapása után a játék véget ér. Ahogy fent említettem az kocsi mozgását a híres Flappy Bird játékban lévő madárhoz hasonlítanám, hiszen a képernyő érintésére a kocsi emelkedik. A program elkezdését először is azzal kezdtem, hogy az elképzéléseim szerint megszerkeztettem a kinézethez szükséges képeket, mint a háttér, a kisebb ikonok, vagy az autó. Miután ezzel végeztem jöhetett a kód.

A programunk fő felépítése 4 java kódból és 3 layout-ból áll. A program kezdőlapját xml ben terveztük. Most nézzük meg, hogy is épül fel xml ben egy layout megtervezése:

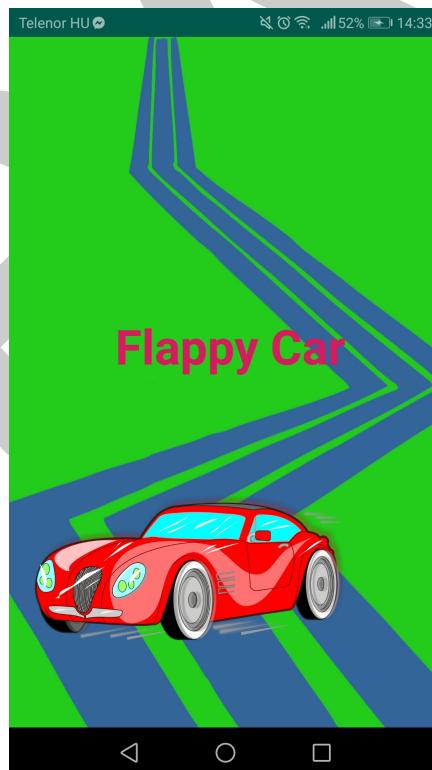
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/cark"
    tools:context=".CarActivity">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:layout_below="@+id/textview1"
        android:layout_marginTop="43dp"
        android:src="@drawable/car_icon"
        />

    <TextView
```

```
    android:id="@+id/textview1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="230dp"
    android:text="Flappy Car"
    android:textAlignment="center"
    android:textColor="@color/colorAccent"
    android:textSize="40sp"
    android:textStyle="bold" />
</RelativeLayout>
```

Első fontosabb sorunk a "android:background="@drawable/cark". Itt adjuk meg a layout háttérét, amit én szerkesztettem. 2 objektumunk van még a layoutban, egy szöveg és egy kép. A kódban láthatjuk, hogyan tudjuk szerkeszteni ezeket. megadhatjuk egy kép szélességét, magasságát, margót és persze, hogy melyik képet akarjuk oda rakni. A szövegnél továbbá tudunk típuszt, betű méretet és magát a szöveget. Be tudjuk állítani, hogy a képernyő melyik részén helyezkedjen és még sok szerkesztési lehetőségünk van. Én a kód alapján az alábbi kezdőképernyőt hoztam létre:



Ezt a kezdő képernyőt 5 másodpercig fogjuk látni, aztán indul a játék. Ezt a CarActivity.java-ban az alábbi run függvényben adjuk meg:

```
public void run()
{
    try
```

```
{  
    sleep(5000);  
}
```

A program "lelke" a CarView.java-ban található. nézzük is meg mi történik ebben.

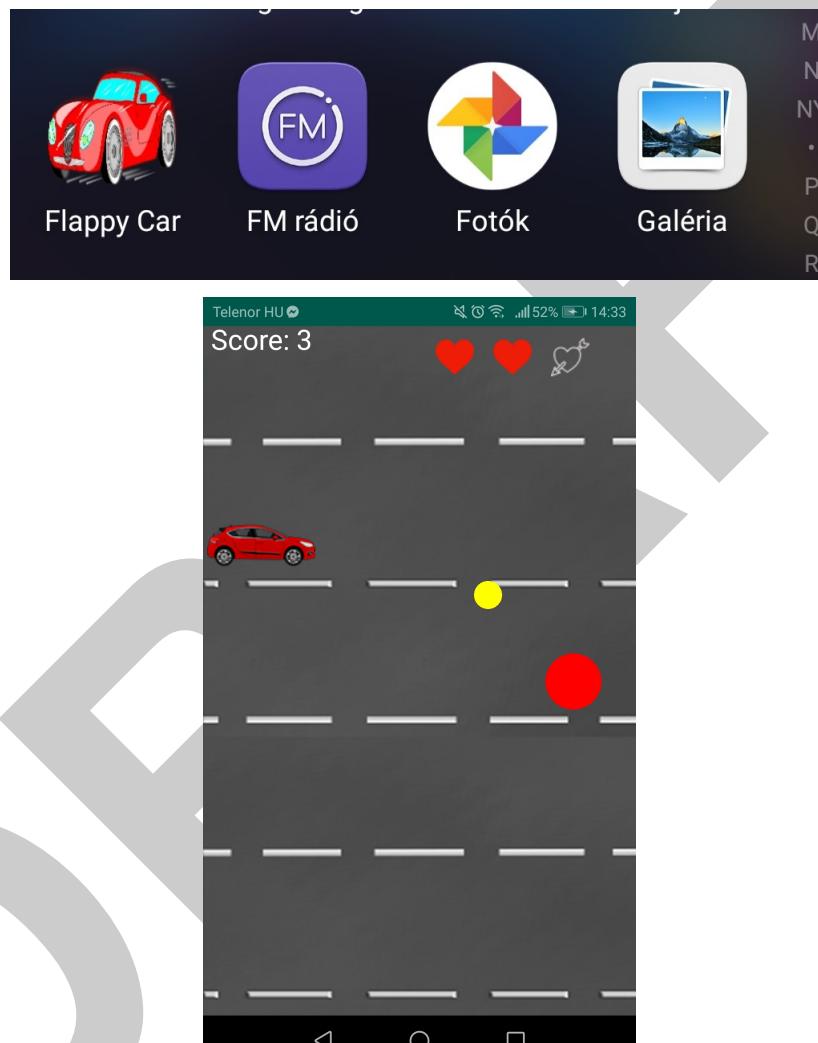
```
public CarView(Context context)  
{  
    super(context);  
  
    car[0]= BitmapFactory.decodeResource(getResources(),R.drawable.car) ←  
        ;  
    car[1]= BitmapFactory.decodeResource(getResources(),R.drawable.car1 ←  
        );  
  
    backgroundImage= BitmapFactory.decodeResource(getResources(),R. ←  
        drawable.background);  
  
    yellowPaint.setColor(Color.YELLOW);  
    yellowPaint.setAntiAlias(false);  
  
    redPaint.setColor(Color.RED);  
    redPaint.setAntiAlias(false);  
  
    scorePaint.setColor(Color.WHITE);  
    scorePaint.setTextSize(70);  
    scorePaint.setTypeface(Typeface.DEFAULT);  
    scorePaint.setAntiAlias(true);  
  
    life[0]= BitmapFactory.decodeResource(getResources(),R.drawable. ←  
        hearts);  
    life[1]= BitmapFactory.decodeResource(getResources(),R.drawable. ←  
        heart_grey);  
  
    carY=550;  
    score=0;  
    lifeCounterOfCar=3;  
}
```

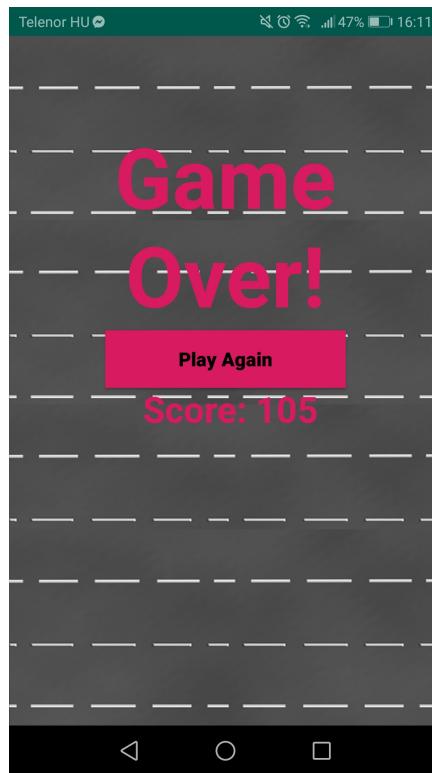
Nem az egész kódot másoltam be, de az egészről fogok beszélni. A program elején deklaráltuk a megfelelő változókat a fent látható CarView() függvény jön amiben megadjuk, hogy a kocsinknak két esetete van és hogy melyik esetben melyik ikon legyen. Aztán a kis golyók színeit adjuk meg és a pontok mérésének a kinézetét(fehér színnel és a szöveg méretét). Mivel szeretnénk az élet elvesztését is jelölni így a life[] is egy tomb és ugyan úgy két eset lesz(ha elveszünk egy életet egy szürke szív lesz helyette). Ezek után megadjuk, hogy a kocsink milyen pozícióba kezdje a játékot. Ezek után megadjuk, hogy 0 pontról indulunk és három életünk van. Ezek után jönnek a fontosabb függvények:

```
protected void onDraw(Canvas canvas)  
  
public boolean hitBallChecker(int x, int y)
```

```
public boolean onTouchEvent (MotionEvent event)
```

Az onDraw() függvényünk fogja generálni a sárga és piros pontokat random függvénnnyel. Itt határozzuk meg a kocsi mozgásterét, és a hitBallCheckerrel() vizsgálni hogy milyen színű pontot kaptunk el. Ha sárga akkor növeljük a pontokat, ha piros akkor csökkentjük az életet. Az onDraw() függvényen belül tudjuk állítani a pöttyök nagyságát és sebességét, a kocsi sebességét. onTouchEvent() figyeli hogy van-e érintés a képernyőn, ha nincs a kocsi vissza esik a képernyő aljára, ha van akkor emelkedik. A program végén, ha kikapunk egy menübe kerülünk, ahol újra tudjuk kezdeni a játékot, és látni a pontjainkat. A telefon menüpártjában lévő ikont és játék nevét a Mainfest-be kell állítanunk. Lássuk hát az eredményt:





19.2. Port scan

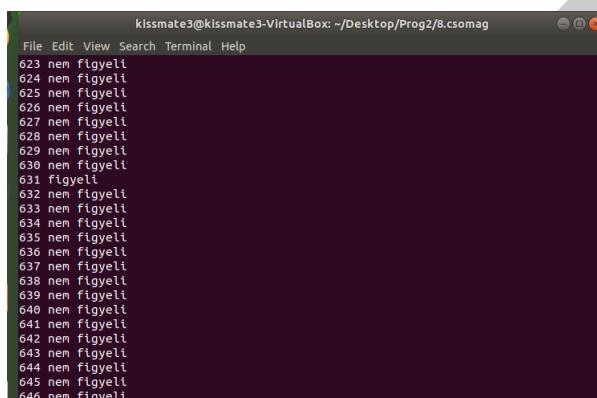
Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére!

A Port Scan azt mutatja meg, hogy egy hálózatnak melyik portjai nyitottak a kommunikációra. Mi most meg fogjuk nézni a kódor, amit tanár úr adott meg:

```
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");  
  
        }  
}
```

}

A programunk azt nézi át, hogy melyik portunkat figyeli épp a számítógép. Láthatjuk hogy a program egy for ciklusbol, azon belül egy try-catch-ból áll. De a program lényege a try blokkon belül van, mégpedig a "java.net.Socket socket = new java.net.Socket(args[0], i);". A for ciklus 1024 ig fut, tehát az ez alatti portjainkat fogjuk vizsgálni. A vizsgálathoz az argumentumként megkapott IP-címmel próbálunk meg egy TCP kapcsolatot létrehozni. A try blokkban említett sor fogja ezt megpróbálni. Ha sikerült a kapcsolat létrehozása akkor kiíratjuk, hogy "figyeli". Ezek után az új socketet amit nyitottunk be is zárjuk, hogy feloldjuk a lefoglalt portot. Ha viszont a kapcsolat nem sikerült a kapcsolat, akkor az Expection lép életbe, ami kiírja, hogy nem figyeli a portot a gép. A program futtatásánál a következő eredményt kapjuk:



```
kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/8.csomag
File Edit View Search Terminal Help
623 nem figyeli
624 nem figyeli
625 nem figyeli
626 nem figyeli
627 nem figyeli
628 nem figyeli
629 nem figyeli
630 nem figyeli
631 nem figyeli
632 nem figyeli
633 nem figyeli
634 nem figyeli
635 nem figyeli
636 nem figyeli
637 nem figyeli
638 nem figyeli
639 nem figyeli
640 nem figyeli
641 nem figyeli
642 nem figyeli
643 nem figyeli
644 nem figyeli
645 nem figyeli
646 nem figyeli
```

Most pedig beszéljünk a kivételekről. A kivételeket a program futása közben használjuk, és a futás közbeni hiba hatására megszakíthatják a program futását. Ilyenkor egy objektum jön létre, ebbe lesz a hiba típusa, programunk állapota. Két féle kivételről beszélhetünk. Egyik a program futása közben jön létre, erre példák az indexelési kivételek, hibás számítások okozta kivételek. A másik fajtája amik nem a futás alatt jöttek létre, ilyen például az I/O kivételek. A fenti programunk abban az esetben fog kivételelt adni, ha a portunkat nem figyeli a portot. Ha megnézzük esetünkben mien kivételelt kapunk, láthatjuk, hogy ebben az esetben I/O exceptiont kaptunk.

19.3. Junit teszt

A https://propater.blog.hu/2011/03/05/labormeres_otton_avagy_hogyan_dolgozok_fel_egen_pedat_poszt_kézzel_számított_mélységét_és_szórását_dolgozd_be_egy_Junit_tesztbe poszt kézzel számított mélységét és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

Feladatunkat kezdjük azzal, hogy mi is az a JUnit. A JUnit a Java nyelnek egy egyszégeszt keresztrendszerre. A lényege, hogy megvizsgáljuk, hogy egy program nekünk megfelelően fog e működni. Jelen esetben a Tanár úr által megadott ProgPateres eredmények alapján fogjuk elvégezni a BinFánk javás verzióján. Ehez az Eclipse programot használtam. Miután megszereztük a szükséges bővítményeket teszthez, lássuk a kódot:

```
package Binfam;

import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;
class BinfaTest {
    LZWBinFa binfa = new LZWBinFa();
```

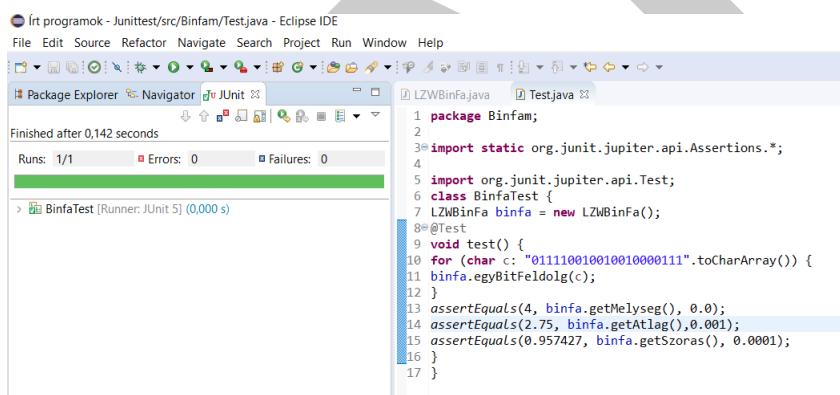
```

@Test
void test() {
    for (char c: "011110010010010000111".toCharArray()) {
        binfa.egyBitFeldolg(c);
    }

    assertEquals(4, binfa.getMelyseg(), 0.0);
    assertEquals(2.75, binfa.getAtlag(), 0.001);
    assertEquals(0.957427, binfa.getSzoras(), 0.0001);
}
}

```

Láthatjuk, hogy a fő függvényünk elején példányosítjuk az LZWBinFa-nkat. Ezután a test() függvényen belül egy speciális for cikluson belül körbe járjuk bitenként a tanár úr álltal megadott bit sorozatot és a binfa.egyBitFeldolg()-al feldolgozzuk a biteket. Ezek után jön maga a teszt, Ugye tudjuk, hogy a megadott bitsorozatnak tanár úr lapon kiszámolta az eredményeit. Tehát a assertEquals() -el ellenőrizzük, hogy az LZWBinFA által adott eredmény megegyezik e a tanár úr álltal kiszámolt eredménnyel. A assertEquals() úgy épül fel, hogy megadjuk az értéket amit kapunk kell, aztán az értéket amit vizsgálunk, végül pedig hogy mekkora lehet az eltérés. Ha minden megegyezik akkor a teszt 0 hibát fog vissza adni, tehát sikeres lesz a teszt. Viszont ha az értékek nem egyeznek, akkor vissza kapjuk a hibákat. Lássuk is az eleményt:



19.4. AOP

Szój bele egy átszövő vonatkozást az első védési programod Java átitratába! (Sztenderd védési feladat volt korábban.)

Feladatunkat kezdjük azzal, hogy mi is az az AOP, teljes nevén Aspect-Oriented Programming. Az AOP programozási paradigma. Akkor használjuk, mikor egy már meglévő programot úgy módosítunk, hogy nem nyúlunk bele a forrás kódba, ezt hívjuk átszövésnek. Az átszövessel tudunk módosítani a program működésén, függvényein. Mi most az LZWBinfa javas verziójának kiíratásán fogunk módosítani. Ehez az AspectJ nyelvet fogjuk használni. A futtatás előtt telepítenünk kell az aspectj-t. Az eredeti forráskódunkban postorder bejárásban, szóval mi preorder bejárásra fogjuk átírni. Lássuk a kódot, hogy is néz ez ki:

```

privileged aspect BinfaAspect{
    void around(LZWBinFa fa, LZWBinFa.Csomopont elem, java.io.PrintWriter os) ←
    :
    target(fa) && call(public void LZWBinFa.kiir(LZWBinFa.Csomopont, java.io. ←
    PrintWriter )) && args(elem, os) {

```

```
if (elem != null) {
    ++fa.melyseg;
    for (int i = 0; i < fa.melyseg; ++i)
        os.write("----");
    os.write(elem.getBetu () + " (" + (fa.melyseg - 1) + ") \n");
    fa.kiir(elem.nullasGyermek (), os);
    fa.kiir(elem.egyesGyermek (), os);
    --fa.melyseg;
}
}
```

A kódunk a privileged kulcsszóval indul, erre azért van szükségünk, hogy az aspect hozzá férjen az LZW-Binfa privát részeihez. Ezután jön az aspect és annak a neve, ezekkel jelöljük, hogy ez egy átszövés. Az aspect törzse a void típusú around függvény. Ez a függvény kapja meg paraméterül a szükséges tagokat a Binfábol. Ezek után jön az, hogy melyik függvényen akarunk modosítást végre hajtani. A függvénynek meg kell adnunk az összes paraméterét és az around paramétereit. Ezek után a kiir függvényt modosítuk úgy, hogy postorder helyett preorder kiiratás legyen. Ha ezzel megvagyunk, már csak futtatásnál hozzá kell füznünk a szöveget a binfához. Ehez szükségünk lesz az aspectjrt.jar mappára. Nézzük is az eredményt:

```
java -classpath ./aspectjrt-1.6.7.jar: LZWBinFa teszt.txt -o eredmeny2.txt
```

```
depth = 12
mean = 7.5625
var = 1.9653244007033546
```

```
depth = 12
mean = 7.5625
var = 1.9653244007033546
```

Láthatjuk, hogy a kiíratás máshogyan történt mint az eredeti binfába, tehát sikeres volt az átszövés.

20. fejezet

Helló, Calvin!

20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13/hello_sbol Háttérként ezt vetítsük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

Feladatunkat kezdjük azzal, hogy mi is az a MNIST. Az MNIST egy adatbázis amiben nagyméretű, kézzel írott számjegyek találhatók. Ezeket képfeldolgozó rendszerek tanítására használják, egyik fő célja a gépi tanítás. Maga az adatbázis 10 000 teszt képet, és 60 000 tanuláshoz való képet tartalmaz. Mi is ezt az adatbázist fogjuk használni. A programunk a MNIST segítségével megröpobálja felismerni az általunk megrajzolt számot. A gépi tanuláshoz tensorflowt használunk. A Tensorflow egy nyílt fórráskódó platform a gépi tanuláshoz. Most pedig lássuk a programunkat és hogy hogyan is kelt életre.

Ahoz hogy a programunk életre keljen, le kell töltenünk magát a tensorflowt és hozzá a pythonot, ugyanis a forrás kódunk a python. Ezt az alábbi módon csináltuk:

```
//tensorflow telepítése:  
sudo pip3 install tensorflow==1.14  
  
//python telepítése:  
apt -get install python3
```

Ha ezekkel készen vagyunk, már futtathatjuk is a kódunkat. Elsőre hibát fogunk kapni, ugyanis a programunk futtatásához szükségünk van a matplotlib könyvtárra, amit ha letöltünk már fut is a program. Na most pedig nézzük a kódot:

```
from tensorflow.examples.tutorials.mnist import input_data  
  
import tensorflow as tf  
  
import matplotlib.pyplot as plt  
  
FLAGS = None  
  
def readimg():  
    file = tf.read_file("sajat.png")  
    img = tf.image.decode_png(file, 1)
```

```
return img

...
img = mnist.test.images[42]
image = img
plt.imshow(image.reshape(
    28, 28), cmap=plt.cm.binary)
plt.savefig("4.png")
plt.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

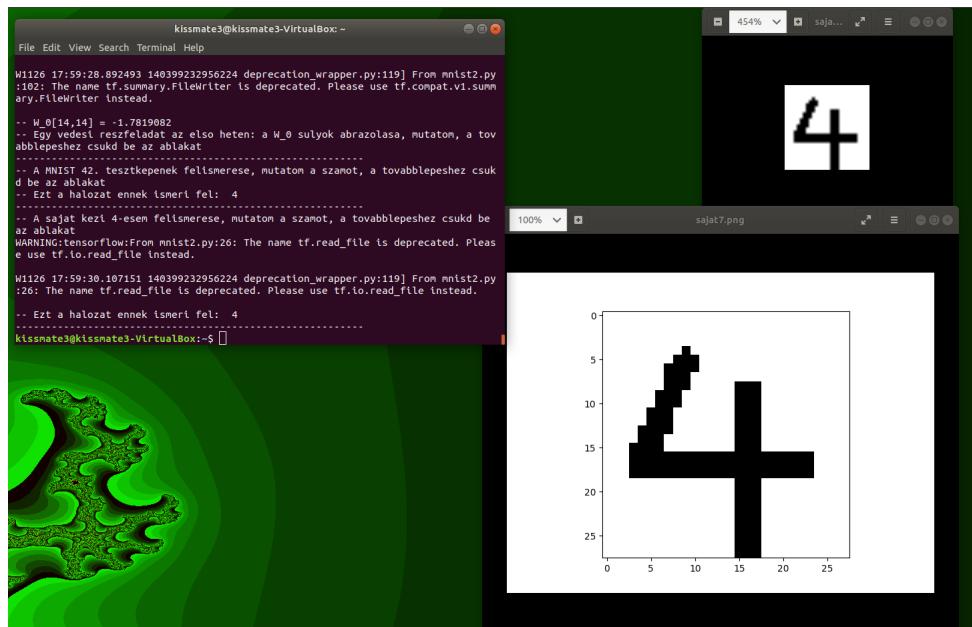
...
img = readimg()
image = img.eval().flatten()
for i, img in enumerate(image):
    image[i] =abs(image[i]-255)

plt.imshow(image.reshape(28, 28), cmap=plt.cm.binary)
plt.savefig("sajat7.png")
plt.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")
```

A kód elején beimportáljuk a tensorflow-ot és a matplotlib-et. Ezek után következik az általunk rajzolt kép beolvasása, amit sajat.png névre keresztelünk el. A következő kódrészletben történik az összehasonlítás. Az alap képünk 28x28 ezért a Mnist teszt képet átkonvertálja 28x28 as képre, majd a sess.run() fogja elvégezni az összehasonlítást, és ennek lesz a vissza térsí eredménye a felismert szám. A végén látjuk, hogy az eredményt lementjük a sajat.png néven és kiíratjuk a felismert számoz.

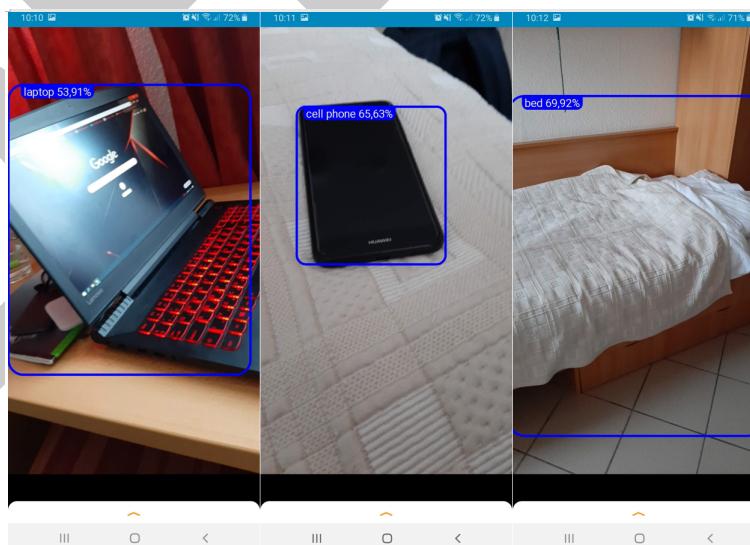


20.2. Android telefonra a TF objektum detektálója

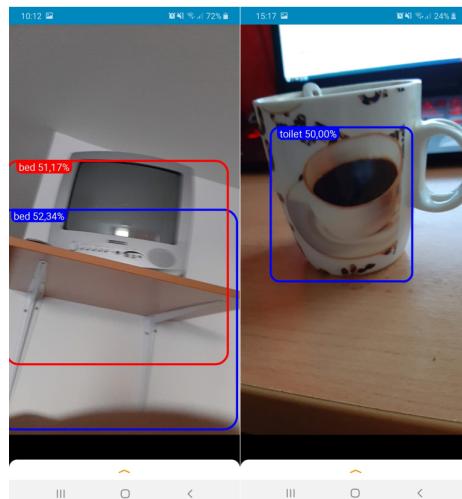
Telepítük fel, próbáljuk ki!

Ebben a feladatban a dolgunk minden össze annyi volt, hogy egy TF objektum detektálót telepítünk a telefonra. A telepítéshez letöltöttem a program apk fájlját, és az alapján telepítettem, de a program megtalálható a Google Play Áruházban is. A program lényege, hogy a telefonunk kameráján keresztül, a program megpróbálja felismerni az adott tárgyakat, vagy például, hogy ember e az adott objektum. Ehez egy %-ot csatol, hogy mennyire biztos abban, hogy mi az adott objektum. A tesztelés során a program elég jól teljesített, de nem hibátlanul.

Az elvégzett teszten jól felismerte a telefont, laptopot, bögrét, ágyat. Lássuk is a képeket:



De ahogy mondtam nem tökéletes. A poharat néha wc nek ismerte fel, vagy a tv-t és a polcot ágynak, de más szemszögből azonnal korrigált.



20.3. Minecraft MALMO-s példa

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lásd pl.: <https://youtu.be/bAPSu3Rndl8>, https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innen_tol_mi, https://bhaxor.blog.hu/2018/10/28/minecraft_

Feladatunk ebben a részben a világ egyik legismertebb és legtöbb játékoszt számláló játékkal fogunk foglalkozni. A Malmo projekt, a mesterséges intelienciával való kísérletezés és kutatás platformja a Minecraft játékot felhasználva.

Először is nézzük, hogy kell feltelepíteni. Én Windows 10 re telepítettem. Hogy a Malmo fussion a gépünkön le kellett tölteni magát a Malmo projeket, ami a Microsoft github repójában megtalálható. Ezen kívül szüksünk van a Pythonra, OpenJDK-t és az XSD. Miután kész a letöltés és beállítottuk a megfelelő környezeti változókat jöhet a Malmo életrekelezése. A Malmos mappánkon belül nyitunk egy powershellt és végezzük el a következő lépéseket:

```
.\launchClient.bat
```

Egy kis időt várva el is indul a Minecraft. Küldetést úgy tudunk neki adni, hogy egy másik powershellbe a Python_Examples mappán belüli küldetéseket futtatjuk. De előtte nézzük meg jobban a Malmo-t. A mappán belül van egy pdf, ami a példák alapján leírja a malmo működését a példák alapján. Például az első példa alapján megtudjuk, hogy az agents_host.sendCommand() parancsokkal tudjuk irányítani a karakterünket például: agents_host.sendCommand("jump 1"). De alapmérézetben a mozgásért a ContinuousMovementCommands felel például a turn[-1,1] el fordul balra.

Most pedig nézzük meg mi az a Mission XML. Az XML a háttérben dolgozik, ez határozza meg magát a küldetést, én a példa_6 os feladatot hajtottam végre, most ennek az XML kódját tekintsük át:

```
<?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>Cliff walking mission based on Sutton and Barto.</Summary>
```

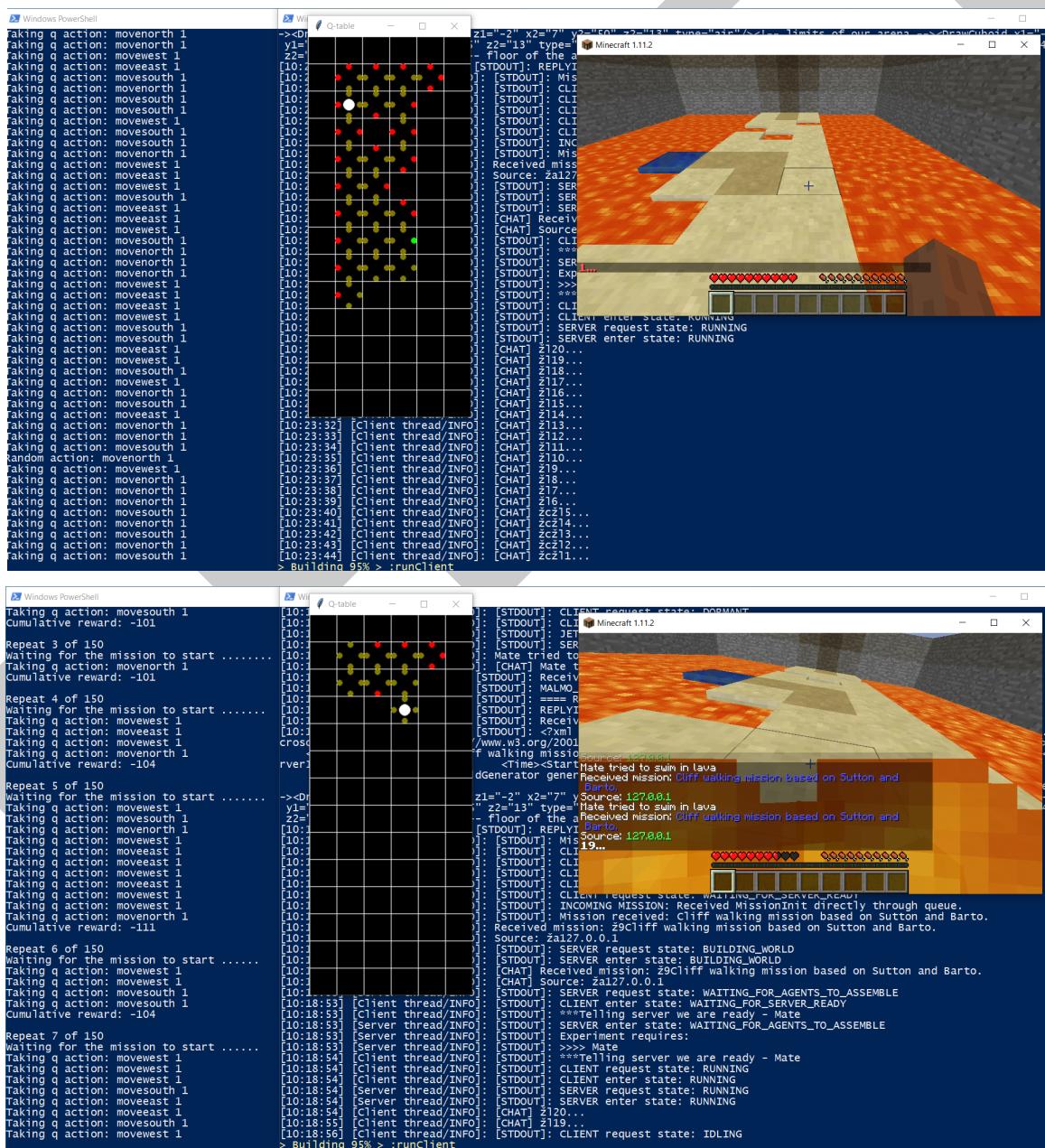
```
</About>

<ServerSection>
    <ServerInitialConditions>
        <Time><StartTime>1</StartTime></Time>
    </ServerInitialConditions>
    <ServerHandlers>
        <FlatWorldGenerator generatorString="3;7,220*1,5*3,2;3;,biome_1"/>
        <DrawingDecorator>
            <!-- coordinates for cuboid are inclusive -->
            <DrawCuboid x1="-2" y1="46" z1="-2" x2="7" y2="50" z2="13" type="air" />           <!-- limits of our arena -->
            <DrawCuboid x1="-2" y1="45" z1="-2" x2="7" y2="45" z2="13" type="lava" />          <!-- lava floor -->
            <DrawCuboid x1="1" y1="45" z1="1" x2="3" y2="45" z2="17" type="sandstone" />      <!-- floor of the arena -->
            <DrawBlock x="4" y="45" z="1" type="cobblestone" />       <!-- the starting marker -->
            <DrawBlock x="4" y="45" z="7" type="lapis_block" />       <!-- the destination marker -->
        </DrawingDecorator>
        <ServerQuitFromTimeUp timeLimitMs="20000"/>
        <ServerQuitWhenAnyAgentFinishes/>
    </ServerHandlers>
</ServerSection>
```

Ahogy látható itt tudjuk megadni hogy az idő mennyitől induljon, továbbá a timeLimitMs-ben, hogy mennyi időnk van a küldetésre. Világot generálni a WorldGeneratorral tudunk. a mi esetünkben egy lapos világot hoztunk létre ami biome_1, azaz az általános biom, de létre tudnánk hozni piros, kék ,havas világokat is. A DrawingDecoratoron lehetővé teszi, hogy formákat hozzunk létre blokkokból, ez a dekoráció. Ilyenek a DrawCuboid, DrawLine, DrawItem, DrawSphere, DrawBlock. A mi esetünkben a DrawCuboid és a DrawBlock.al találkozhatunk, ezekben megadjuk a helyzetét, és a block vagy cuboid típusát, például type="lava".

```
<AgentSection mode="Survival">
    <Name>Mate</Name>
    <AgentStart>
        <Placement x="4.5" y="46.0" z="1.5" pitch="30" yaw="0"/>
    </AgentStart>
    <AgentHandlers>
        <DiscreteMovementCommands/>
        <ObservationFromFullStats/>
        <RewardForTouchingBlockType>
            <Block reward="-100.0" type="lava" behaviour="onceOnly"/>
            <Block reward="100.0" type="lapis_block" behaviour="onceOnly"/>
        </RewardForTouchingBlockType>
        <RewardForSendingCommand reward="-1" />
        <AgentQuitFromTouchingBlockType>
            <Block type="lava" />
            <Block type="lapis_block" />
        </AgentQuitFromTouchingBlockType>
    </AgentHandlers>
</AgentSection>
```

Az AgentSection adhatjuk meg, hogy a játék módja milyen legyen, esetünkben ez a Survival lesz. A Name-al a karakter nevét adjuk meg. Az AgentStart-on belül adjuk meg a küldetésünk kiinduló pontját. RewardForTouchingBlockType azt adjuk meg, hogyha lavaba lépünk -100 pontot kapunk, ha a megadott blockra, (esetünkben a lapis blovk) akkor 100 pontot kapunk, és minden egyes megtett után -1 pontot. AgentQuitFromTouchingBlockType al azt adjuk meg, hogy az időn kívül mikor lépjen ki a program. A python kódban építjük fel magát a mozgást, hogy találja meg a karakterünk a megfelelő utat a megadott blokkig.



20.4. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a [Ez a feladatunk az előző példánk nehezített verziója. Ugyanis a számításba növeljük az iterációt 1000 ről 20000re. A forráskódot a Tensorlow oldalán találhatjuk itt is. De hogy működjön a programunk, kell egy kicsit hozzá írni.](https://arato.inf.unideb.hu/batfai.norb8/fóliáját!</p></div><div data-bbox=)

```
img = readimg()
image = img.eval()
image = image.reshape(28*28)
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.binary)
matplotlib.pyplot.savefig("sajat.png")
matplotlib.pyplot.show()
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
```

A matplotlib.pyplot.savefig("sajat.png") - vel történik a sajáz képünknek a beolvasása, itt it átméretezzük 28*28 as képre.

IV. rész

Irodalomjegyzék

DRAFT

20.5. Általános

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

20.6. C

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

20.7. C++

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

20.8. Lisp

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

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

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