Univerzális programozás

Programozás Gyorstalpaló

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

COLLABORATORS			
	TITLE : Univerzális programozás		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Benyovszki, Balázs	2019. november 11.	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-03-04	Turing fejezet befejezve.	bbalazs
0.0.6	2019-03-11	Chomsky fejezet befejezve.	bbalazs
0.0.7	2019-03-18	Caesar fejezet befejezve.	bbalazs
0.0.8	2019-03-25	Mandelbrot fejezet befejezve.	bbalazs

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-04-01	Welch fejezet befejezve.	bbalazs
0.1.0	2019-04-08	Conway fejezet befejezve.	bbalazs
0.1.1	2019-04-22	Schwarzenegger fejezet befejezve.	bbalazs
0.1.1	2019-04-29	Chaitin fejezet befejezve.	bbalazs
1.0.0	2019-05-09	Teljes könyv befejezése!	bbalazs

Ajánlás

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

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

Tartalomjegyzék

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: http://esr.fsf.hu/hacker-howto.html!
- Olvasgasd aztán a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- C kapcsán a [KERNIGHANRITCHIE] könyv adott részei.
- C++ kapcsán a [BMECPP] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a The GNU C Reference Manual, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! 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.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.

II. rész

Tematikus feladatok

Bátf41 Haxor Stream
A feladatokkal kapcsolatos élő adásokat sugároz a https://www.twitch.tv/nbatfai csatorna, melynek permanens archívuma a https://www.youtube.com/c/nbatfai csatornán található.

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/2.1feladat

```
Végtelen ciklus 1 szál 100%-on
#include <stdio.h>

int main()
{
for(;;){

    }
}
```

A VÉGTELEN CIKLUS SOHA NEM ÉR VÉGET SEMMI FELTÉTEL NINCS BENNE EGYSZERŰ FOR(;;) VEL MEGOLDHATÓ, SOHA NEM ÁLL LE KÜLSŐ BEAVATKOZÁS NÉLKÜL CTRL+C GOMBKOMMBINÁCIÓ SZÜKSÉGES LINUXON A MEGÁLLÍTÁSÁHOZ. ÁM EZ A KÓD ÍGY 1 szálat PÖRGET FOLYAMATOSAN 100%.ON.

```
Végtelen ciklus 1 szál 0%-on
#include <stdio.h>
#include <unistd.h>  //Ez a fájl tartalmazza a sleep parancsot az  ↩
   includeolása elengedhetetlen a program megfelelő működéséhez.

int main()
{
for(;;){

    sleep(1);

    }
}
```

A KÉSŐBBI LABDAPATTOGTATÁS FELADATNÁL JÖTT KAPÓRA AZ ISMERETE DE PL A WINDWOS RENDSZER IS A VÉGTELENSÉGIG VÁR A PARANCSRA. A SPROGRAMOT ERŐFORRÁSHATÉKONYABBÁ TEHETJÜK HA HASZNÁLJUK A SLEEP(TETSZŐLESEG MÁSODPERC) SORT AMIHEZ SZÜKSÉGES **#include unistd.h** HEADER FILE. EZ TETSZŐLEGES MP-RE ALTATJA A VÉGTELEN CIKLUST EZÁLTAL NEM PÖRGETI A szálat 100%-ON.

```
Végtelen ciklus összes szál 100%-on
#include <stdio.h>
#include <stdlib.h>

int main()
{
    #pragma omp parallel

    for(;;){

//-fopenmp a frodításhoz szükséges parancs

    }
}
```

A programot megírhatjuk úgy is, hogy minden szálat 100%-on pörgessen a végtelen ciklusunk ezhez csak a #pragma omp parallel sort kell használnunk és amikor fordítjuk akkor szügséges még egy kapcsoló a -fopenmp.

Az eredményeket a programok futtatása közben a linuk beépített erőforrásfigyelőjével követhetjük.

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

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

Megoldás videó:

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

```
Program T100
{

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

  main(Input Q)
  {
```

```
    Lefagy(Q)
  }
}
```

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.

Ilyen program nem létezik mert ha a program tartalmaz végtelen ciklust akkor a T1000 kiírja, hogy lefagy viszont ha nem akkor pedig az if miatt saját magát fogja lefagyasztani hiszen végtelen ciklusba lép. Ellentmondást kapunk a végén tehát ez bebezonyírja, hogy ilyen programot nem lehet írni.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/2.3%20v%C3%A1ltoz%C3%B3k%20%C3

PageRank

```
#include <stdio.h>

int main()
{
    int a = 0;
    int b = 0;
    printf("Adja meg az a szamot: ");
    scanf("%d" , &a );                          // Tegyük fel, hogy ez a szám a ↩
        3.
    printf("Adja meg a b szamot: ");
    scanf("%d" , &b);                           // Tegyük fel, hogy ez a szám a ↩
        6.
    b = b-a;                                    // Vegyük a két szám  ↩
       különbségét (6-3) ami 3.
    a = a+b;                                    // a-t tegyük eggyenlővé az  ↩
       eredeti a val és a két szám különbségével ami (3+3) jelen esetben 6  ↩
       eredménnyel zárul.
    b = a-b;                                    // A b válltozónkat tegyük  ↩
       eggyenlővé a módisított a válltozó és a b különbségével (6-3) amely  ↩
       művelet 3 eredménnyel zárul.
    printf("a=%d%s",a,"\n");
    printf("b=%d%s",b,"\n");

}
```

Cseréljünk fel két válltozót bármiféle segédválltozó vagy logikai utasítás nélkül. Bekérünk a standard kimenetről két válltozót (ehez kell az stdio könyvtár) a júzertől amiket okos matekozással cserélünk fel, bármiféle segédválltozó nélkül. Kommentekkel prezentált módon.

2.4. Labdapattogás

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

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

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/2.4%20labdapattogtat%C3%A1s

```
Labdapattogás if-el

#include <stdio.h>
#include <curses.h>
#include <unistd.h>
// -lncurses kapcsoló kell amikor fordítjuk külömben hibákat kapunk és nem fordul le.

int
main ( void )
{
    WINDOW *ablak; // Ez lényegében egy változó deklarálás. Ami az albak mérete változót hozza létre tisztán.
    ablak = initscr (); // Eggyenlővé teszi az ablak változót az ablak méretével.

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx ); //A függvény átadja az ablak adatait az my-ba és az mx-be.

        mvprintw ( y, x, "O" );  // Kirajzolja a labdát a karakteres konzolra. A manuál szerint hasoló a printf-hez.

        refresh ();
        usleep ( 100000 ); // Altatjuk a programot, hogy követhetőek legyenek az események. Minnél nagyobb a szám annál lassab b a labda.
        clear(); //Imprúvment kitörli az előzőleg lerakott labdákat, hogy a hatás meglegyen

        x = x + xnov; // Labda vízszintes pozíciója
```

```
        y = y + ynov; // Labda függőleges pozíciója

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

        }

    }

    return 0;
}
```

Meg is írtuk a programot if-es feltételekkel. A **curses** könyvtár felhasználásával, amely tartalmazza a nekünk szükséges parancsokat mint pl **WINDOW *, initscr (), getmaxyx (),mvprintw, refresh ()** . A program egy karaktert mozgat a standart outon mintha ezzel labdapattogtatást imitál ezt enterek és szóközök majd maga a labda folyamatos írásával éri el.

```
Labdapattogás if nélkül
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define SZEL 78     // A programunk a SZEL szó helyére autómatikusan be ↩
   fogja illesztenia 78 számot. Akkor jó, ha sok helyen használnánk ezt ↩
   mert akkor nem fog kelleni mindenhol átjavítani ha esetleg gikszer van.
#define MAG 22

int putX(int x,int y) // 2 integerrel dolgozó funkció.
{
    int i; //Ciklusszámláló.

    for(i=0;i<x;i++) // Forciklus ami lefelé tolja a labdát. Addig megy ↩
       amég az i nem érte el az x azaz a konzolablak magasságt, hogy ne ↩
       menjen ki a kabda a képernyőről.
    printf("\n");

    for(i=0;i<y;i++) // Forciklus ami oldalirnyba tolja a labdát. Addig ↩
       megy amég az i nem érte el az x azaz a konzolablak szélességét, hogy ↩
        ne menjen ki a kabda a képernyőről.
    printf(" ");
```

```
    printf("X\n"); // Maga a labda.

    return 0;
}

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

    while(1) //Végtelen ciklus ami írja a labdát.
{
    system("clear"); // Törli az eddig leírt labdákat, hogy ne csúfolja ↩
       össze a képernyőt.
    putX(abs(MAG-(x++%(MAG*2))),abs(SZEL-(y++%(SZEL*2)))); // Meghívjuk a ↩
       putX funkciónkat és felparaméterezzük. Abszolútértékbe helyezzük a m ↩
       űveleteket mert negatív képernyőhosszt azért nem szeretnénk kapni.
    usleep(50000); // Altatjuk egy kicsit a programunkat, hogy ne legyen ↩
       villámgyors a labdapattogás.
}

    return 0;
}
```

Az if nélküli verziót abszolútérték felhasználásával írtuk meg, hogy ne mennyünk ki a koordinátarendszer + feléből, hiszen a monitorunk nem tudja megjeleníteni a negatív koordinátákat. Az elején meghatározzuk a képernyőnk várható szélességét ezt a fordító majd be fogja helyettesíteni a megfelelő helyre. Ezeket a kommentben leírt helyzetekben érdemes használni.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/2.5%20sz%C3%B3hossz

```
Gépi szó
#include <stdio.h>
int main()
{
    int a=1;
    int n=0; // Itt fogjuk tárolni az int méretét.
    while(a!=0)
    {
        n+=1; // Minden körbe növeljük 1-el.
        a=a<<1; // Minden körben tolja 1-el balra.
    }
    printf("Megoldas:%d%s",n,"\n"); // Kiiratjuk a végeredményt a standard ↩
       kimenetre
```

```
}
```

Az alábbi kód megmutatja nekünk az int változó méretét, hogy hány biten tárolja a gépünk az **int** változókat. Ezt a következőképp mutatjuk meg. A ciklusunk addog fog menni amég az **a int** válltozó el nem éri a 0-át. Ezt a **bitshift** operátorral érjük el amely addig tolja az 1-est amég ki nem kerül a válltozóból. Még kell még továbbá számolnunk azt, hogy ezt hány kör alatt éri el a **bitshift** ezt egy új válltozó bevezetésével érjük el amit minden egyes körben amég fut a **while ciklus** növeljük az értékét 1-el. Igy megkapjuk a végeredményt ami 32.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/2.6%20pagerank

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

void kiir (double tomb[], int db) // Kiirató függvény annyi sorr ír ki ↩
   ahányat megadunk neki paraméterben amikor meghívjuk.
    {
    int i;
    for (i=0; i<db; i++)
    printf("PageRank [%d]: %lf\n", i, tomb[i]); // kiírja az i sorszámot és ↩
        a kiszámolt pagerank tömb i-edik elemét.
    }

double tavolsag(double pagerank[],double pagerank_temp[],int db)p
    {
    double tav = 0.0;
    int i;
    for(i=0;i<db;i++)
    if((pagerank[i] - pagerank_temp[i])<0)
        {
        tav +=(-1*(pagerank[i] - pagerank_temp[i]));
        }
    else
        {
        tav +=(pagerank[i] - pagerank_temp[i]);
        }
    return tav;     //A függvény ezzel az értékkel fog visszatérni miután ↩
       lefutott.
    }

int main(void)
```

```
    {
    double L[4][4] = {                    // Ebben a 2 dimenziós tömbben van ↩
        a linkmátrix ez tartalmazza a linkeket
        {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};        // A majdani végeredményt  ↩
       fogja tartalmazni
    double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};       //  ↩
       Presztízsértékeket tárolja ez a tömb.

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

    for (;;) // Az egész számolást egy végtelen ciklusba tesszük.
    {
        for(i=0;i<4;i++) // Átmásoljuk a PRv értékeit a PR tömbbe.
             PR[i] = PRv[i];
        for (i=0;i<4;i++) // elvégzi a szorzást a linkmátrix megfelelő  ↩
           elemét összeszorozza a presztízstömb megfelelő elemével majd  ↩
           ezeket az értékeket tároljuk a PRv tömbben
        {
            double temp=0;
            for (j=0;j<4;j++)
                temp+=L[i][j]*PR[j];
            PRv[i]=temp; // A PRv tömb elemébe eltároljuk a mátrixszorzás  ↩
               végeredményét.
        }

        if ( tavolsag(PR,PRv, 4) < 0.00001) // Ha a távolség függvényünk  ↩
           kissebb mint 0.00001 akkor kilép a végtelen ciklusból. Ha a  ↩
           linksorunk elérte az oldalt leáll.
            break;
    }
    kiir (PR,4); // Végeredményt írja ki.
    return 0;
}
```

A porgram azt csinálja, hogy összeméri azt, hogy menyi oldal mutat a másikra és az adott oldalra mutató linkeknek a presztizsét is leelörzi, hogy megtudjuk, hogy mennyire releváns az oldal és rangsort álít. A google ezzel az algoritmussal futott be. Lényege, hogy nem lehet árverni. Mert ha létrehozunk egy csomó weboldalt ami 1-re mutat azzal sem érünk semmit mert ha sok alacsony relevanciájú oldal mutogat össze vissza annak kevesebb hatása lesz a pagerankra mintha egy rangos oldal mutat rá valamire.

2.7. 100 éves a Brun tétel

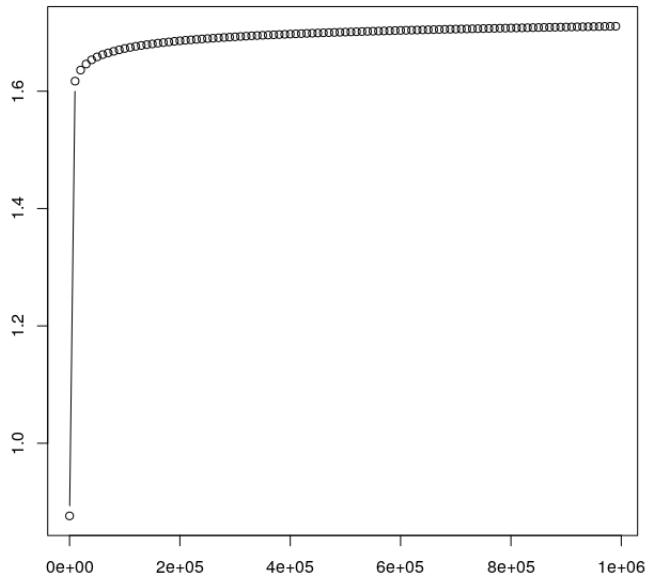
Írj R szimulációt a Brun tétel demonstrálására!

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

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/2.8%20Burn%20t%C3%A9tel

```
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>

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 tétel lényege, hogy végtelen számú ikerprím létezik. Először tisztázzuk mi is az a prímszám. A prímszám az aminek csak 1 és önmaga az osztója. Az ikerprím pedik 2 olyan prímszám melyeknek a külömbsége 2. A tétel ezeknek a reciprokösszegét adja össze. De a Brun tétel segítségével se lehet bebizonyítani azt, hogy végtelensok ikerprím van e.



2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára! Az R eurngy oylan programozási nyelv amit statisztikai számításokhoz és ábrázoláshoz használnak.

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

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

```
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>

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

for (i in 1:kiserletek_szama) {
```

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

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

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

    }
    musorvezeto[i] = mibol[sample(1:length(mibol),1)]
}
nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

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


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

Régen egy tévés vetélkedőműsorban veltettek egy kérdést mégpedig azt. 3 ajtó közül kell választani és csak az eggyik mögött van nyeremény és a résztvevő választ egy ajtót és azt kinyitják neki, de az ajót mögött nincs semmi de ekkor még nem áll le a játék a műsorvezető felajánlja azt, hogy még1x lehet nyitni. A tévés vetélkedő válasza az volt, hogy ez megduplázza a nyerési seélyt. Ez nagy vihart kavart a matematikusok körében és sok matematikus aki az ellenkezőjét vallotta csak ez a szimuláció győzte meg arról, hogy valóban növeli az esélyeket.

3. fejezet

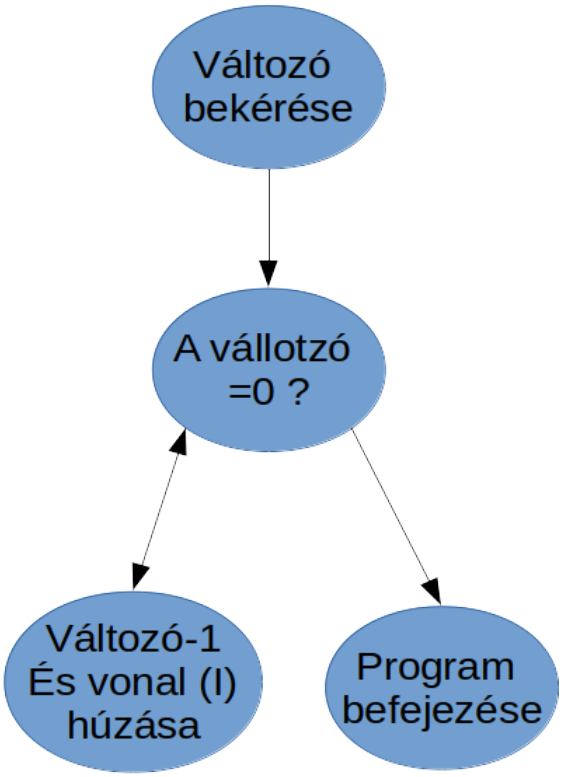
Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/blob/master/turinggep.cpp



Csinálnunk kellett egy állapotmenet gráfot arrol, hogy egy turing gép, hogyan is tud 10-es azaz decimális számrendszerből unárisba azaz 1-es számrendszerbe válltani. Tulajdonképpen mi is az az unáris számrendszer? Az unáris zámrendszer a lehető legegyszerűbb számrendszer amivel egész számokat lehet ábrázolni. Gyakorlatilag az eggyes számokat "volnalak" reprezentálják. pl a 3 a III és így tovább. Gépünk a folyamatábrán azt csinálja, hogy addig húzza a vonalakat amég az elején beírt decimális szám el nem éri a 0-át majd kilép.

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

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

Megoldás videó:

Megoldás forrása:

Noam Chomsky amerikai nyelvész nevézez fűződik a generatív grammatika. Chomsky 4 osztályba sorolta a nyelvtanokat ezek pedig a következők: rekurzíve felsorolható nyelvtanok, környezetfüggő nyelvtanok, környezetfüggetlen nyelvtanok és reguláris nyelvtanok. Ebben az esetben nekünk a környezetfüggő nyelvtan lesz a fontos. Lényeg, hogy a nyil jobb és bal oldalán is megjelenhetnek terminális szimbólumok.

```
A feladat megoldása.
S, X, Y nemterminálisok
a, b, c terminálisok


 S → abc, S → aXbc, Xb → bX, Xc → Ybcc, bY → Yb, aY -> aaX, aY → aa   ↩
    képzési szabályok

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

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:https://github.com/BenyBalazs/Prog1/delete/master/forditas/asdasd

```
Lefordul
#include <stdio.h>

int main ()
{
```

```
    int i;

    for (i = 0; i < 1; i++)
}
```

```
Nemfordul
#include <stdio.h>

int main ()
{

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

Minden nyelvnek megvan a maga nyelvtanja pl. a magyar nyelvben sem mindegy az, hogy valamit, milyen szóhasználattal és szavakkal írunk le. Ugyan így a programozási nyelveknek is megvan a maguk nyelvtana és helyesírásellenörzője ami jelen esetben a fordítóprogram gcc, g++ stb. És szintés hasonlóan az élő nyelvhez ezek is folyamatosan változnak bővülnek funkciókkal. Erre egy jó példa a fenti 2 kód. A C nyelv régi változatában nem lehetet deklarálni a forciklus fejében a cikliusszámlálót csak értéket adni neki. Az új verzióban azomban már ez lehetséges. A fordítónk a legújabb verziót használja alapértelmezetten ezért ha a C89-es verzióval szeretnénk futtatni akkor szükséges a **-std=c89** kapcsoló.

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

```
LexikálisElemző
    %{
#include <stdio.h>
int realnumbers = 0;
%}
digit   [0-9]
%%
{digit}*(\.{digit}+)?   {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
 yylex ();
 printf("The number of real numbers is %d\n", realnumbers);
 return 0;
}
```

Megírtuk a lexikális elemzőnket ami meg adja majd nekünk, hogy mennyi valós számot ütöttünk be ezt az elején deklarált integerbe fogja majd írni a ++realnumbers segítségével. Megadjuk, hogy mi számit digitnek egy intervallumban. Felkészítjük a printf-et, hogy ki kell írnia egy stringet és egy double számat majd kiiratjuk a yytext tartalmát ami string és ezt az atof(yytext) átalakítjuk double-é. A mainbe a yylex (); elkezdi az analízist ami egy integert ad vissza. A végén pedig kiírjuk a realnumbers változó értékét. Ehez használjuk a lexert hiszen "Óriások vállán állunk" nem írunk meg mégegyszer egy működő programot.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

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

Futtatas:
$ gcc l337d1c7.c -o l337d1c7 -lfl
(kilépés az input vége, azaz Ctrl+D)

Copyright (C) 2019
Norbert Bátfai, batfai.norbert@inf.unideb.hu

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <https://www.gnu.org/licenses/>.
*/
%{
  #include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <ctype.h>

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

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

  {'a', {"4", "4", "@", "/-\\"}},
  {'b', {"b", "8", "|3", "|}"}},
  {'c', {"c", "(", "<", "{"}},
  {'d', {"d", "|)", "|]", "|}"}},
  {'e', {"3", "3", "3", "3"}},
  {'f', {"f", "|=", "ph", "|#"}},
  {'g', {"g", "6", "[", "[+"}},
  {'h', {"h", "4", "|-|", "[-]"}},
  {'i', {"1", "1", "|", "!"}},
  {'j', {"j", "7", "_|", "_/"}},
  {'k', {"k", "|<", "1<", "|{"}},
  {'l', {"l", "1", "|", "|_"}},
  {'m', {"m", "44", "(V)", "|\\/|"}},
  {'n', {"n", "|\\|", "/\\/", "/V"}},
  {'o', {"0", "0", "()", "[]"}},
  {'p', {"p", "/o", "|D", "|o"}},
  {'q', {"q", "9", "O_", "(,)"}},
  {'r', {"r", "12", "12", "|2"}},
  {'s', {"s", "5", "$", "$"}},
  {'t', {"t", "7", "7", "'|'"}},
  {'u', {"u", "|_|", "(_)", "[_]"}},
  {'v', {"v", "\\/", "\\/", "\\/"}},
  {'w', {"w", "VV", "\\/\\/", "(/\\)"}},
  {'x', {"x", "%", ")(", ")("}},
  {'y', {"y", "", "", ""}},
  {'z', {"z", "2", "7_", ">_"}},

  {'0', {"D", "0", "D", "0"}},
  {'1', {"I", "I", "L", "L"}},
  {'2', {"Z", "Z", "Z", "e"}},
  {'3', {"E", "E", "E", "E"}},
  {'4', {"h", "h", "A", "A"}},
  {'5', {"S", "S", "S", "S"}},
  {'6', {"b", "b", "G", "G"}},
  {'7', {"T", "T", "j", "j"}},
  {'8', {"X", "X", "X", "X"}},
  {'9', {"g", "g", "j", "j"}}

// https://simple.wikipedia.org/wiki/Leet Amiből válogatnahtunk, hogy mit ↩
   mire cserél a program.
  };

%}
%%
```

```
.   {

      int found = 0;
      for(int i=0; i<L337SIZE; ++i)
      {

        if(l337d1c7[i].c == tolower(*yytext))
        {

          int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

          if(r<91)
            printf("%s", l337d1c7[i].leet[0]);
          else if(r<95)
            printf("%s", l337d1c7[i].leet[1]);
          else if(r<98)
            printf("%s", l337d1c7[i].leet[2]);
          else
            printf("%s", l337d1c7[i].leet[3]);

          found = 1;
          break;
        }

      }

      if(!found)
         printf("%c", *yytext);

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

Milyen király lenne ha írnánk egy olyan programot ami képes az imputot átalakítani l33tesre azaz rajosan kicserélni eggyes betűket számokra egyfajta "titkosítás"-t elérve. Ez szintén a lexerrel érjük el. Megadjuk az eseteket, hogy mire cserélhetőek fel a karakterek az elején egy tömbbe. Utána az imputunkat a lexerrel megírt kód kicseréli rajosra. Nem egy efelktív titkosítás theát csak poénból érdemes használlni komoly dolgokra nem mert egy gép segítségével hamar fel lehet törni az ilyesfajta totkosítást.

3.6. A források olvasása

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

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

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

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 fenti kód ellentéte
```

ii.
```
for(i=0; i<5; ++i) //0-tól 5 ig megy és minden körben növeli az i ↩
   értékét 1-el. 5x fut le
```

iii.
```
for(i=0; i<5; i++) //0-tól 5 ig megy és minden körben növeli az i ↩
   értékét 1-el. 5x fut le
```

iv.
```
for(i=0; i<5; tomb[i] = i++) //0-tól 5 ig megy és minden körben növeli ↩
    az i értékét 1-elés az i értékét betölti a tömb i-edik helyére. 5x ↩
    fut le
```

v.
```
for(i=0; i<n && (*d++ = *s++); ++i) // A forciklus feltételébe nem ↩
   bool értéket adtunk meg hanem egy értékadást a ciklus ki fog akadni ↩
   .
```

vi.
```
printf("%d %d", f(a, ++a), f(++a, a)); // A hiba az, hogy nem tudjuk , ↩
   hogy az ++a pontosanm it fog csinálni.
```

vii.
```
printf("%d %d", f(a), a); // Visszaadja az f függvény által módosított ↩
    vállotzót és az a válltozót.
```

viii.
```
printf("%d %d", f(&a), a) // Az f függvény az a válltozó helyét kapja ↩
   meg a memóriában és ezt a lokációs adatot módosítja és kiírja az ↩
   eredeti válltozót
                    ;
```

Megoldás forrása:

Megoldás videó:

Tan tap magy

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$
(\forall x \exists y ((x<y)\wedge(y \text{ prím})))$ //végtelen sok ikerprímszám van

$(\forall x \exists y ((x<y)\wedge(y \text{ prím})\wedge(SSy \text{ prím})) )$ //végtelen sok prímszám van

$(\exists y \forall x (x \text{ prím}) \supset (x<y)) $ //éges sok prímszám van

$(\exists y \forall x (y<x) \supset \neg (x \text{ prím}))$ //véges sok prímszám van
```

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
  int a; // Egész bevezetése
  ```
- ```
  int *b = &a; //Egész referenciája
  ```
- ```
  int &r = a; // Egész referenciája
  ```
- ```
  int c[5]; //5 elemű tömb
  ```
- ```
  int (&tr)[5] = c; //Egészek tömbjének referenciája
  ```
- ```
  int *d[5]; //Egészre mutató mutatók tömbje
  ```
- ```
  int *h (); //Egészre mutató mutatót visszaadó függvény
  ```
- ```
  int *(*l) (); //egészre mutató mutatót visszaadó függvényre mutató mutató
  ```
- ```
  int (*v (int c)) (int a, int b)] //egészet visszaadó és két egészet kapó ↩
     függvényre mutató mutatót visszaadó, egészet kapó függvény
  ```
- ```
  int (*(*z) (int)) (int, int); //függvénymutató egy egészet visszaadó és ↩
     két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó ↩
     függvényre
  ```

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. dubble ** háromszögmátrix

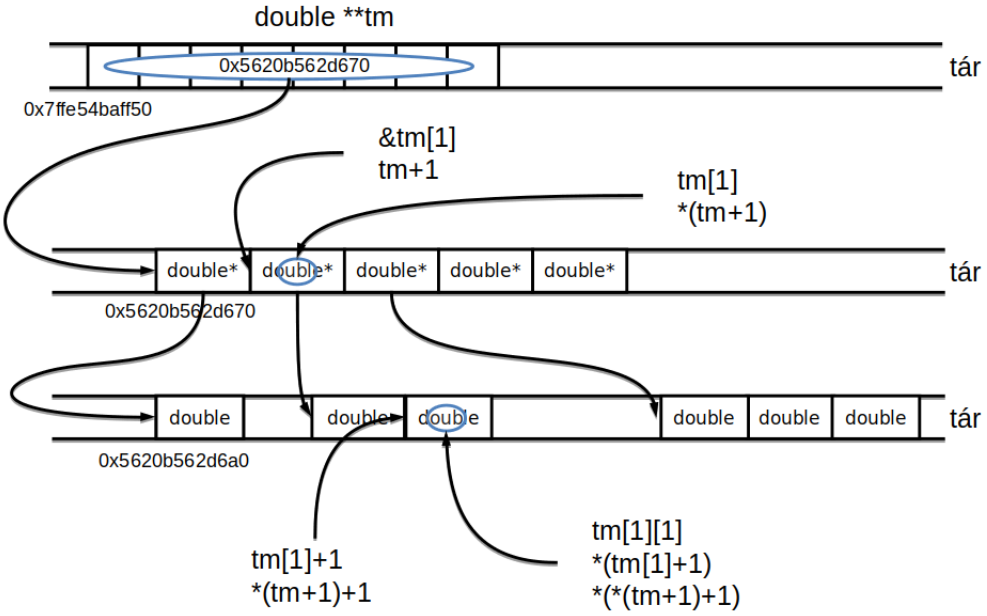
Megoldás videó:

Megoldás forrása:

Ahoz, hogy ezt a feladatot meg tudjuk csinálni tudnunk kell, hogy mi az a háromszögmátrix ennek is a számunkra fontos tulajdonságait. A bemeneti mátrixunknak négyzetesnek kelll lennie tehát ugyanannyi sora és oszlopa van. A háromszögmátrisz az egy olyan mátrix melyenk felső átlóján csupa 0 szerepel. Az alábbi kód pedig ezt valósítja meg.

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

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

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

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

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

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

```
    }

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

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

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

    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;
}
```



4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

// Nevesített konstansok

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

  char kulcs[MAX_KULCS];
  char buffer[BUFFER_MERET];
 //Nevesített konstansok behelyettesítése a tömbméretek helyére

  int kulcs_index = 0;
  int olvasott_bajtok = 0;


  int kulcs_meret = strlen (argv[1]); //Kulcs méretét eggyenlővé tesszük a ↩
     parancssorról beolvasott karakterekkel a strlen csinál a beolvasott ↩
```

```
      szövegből számot.
  strncpy (kulcs, argv[1], MAX_KULCS); //Kimásolja a stringet amire az argv ↩
     [1] mutat de csak a MAX_KULCS méretű másolás engedélyezett.

  while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET))) // ↩
     Lényegében megmondja, hogy mennyi biteal dolgozunk. Addig megy amég  ↩
     nem marad mit beolvasni.
    {

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

      buffer[i] = buffer[i] ^ kulcs[kulcs_index];
      kulcs_index = (kulcs_index + 1) % kulcs_meret;
      //Végrehajtja az exorozós titkosítós mókát.

    }

      write (1, buffer, olvasott_bajtok);

    }
}
```

Ez a módszer egy nagyon régi titkosítási módszer de mai anpig alapul szolgál egy csomó fejlettebb titkosítóhoz. A program bemenetül kér egy szöveges fájlt és egy kódot ezeknek a bineáris számait nézi és ahol külömböző értéket talál oda 0-át fog rakni ahol eggyező értéket talál oda egy 1-est helyez el ezt addig folytatja amég végig nem ér a szövegen.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

```
public
    class ExorTitkosító {

    public ExorTitkosító(String kulcsSzöveg, java.io.InputStream bejöv ↩
       őCsatorna, java.io.OutputStream kimenőCsatorna) //Fájlból olvasunk  ↩
       és fájlba írunk szóval ezek kellenek.
            throws java.io.IOException {

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

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

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

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

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

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

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Úgy működik mint a fenti csak javában. Fogja és a parancssori argumentumba megadott szövegfájl bitjeit össze exorozza a kóddal amit szintén a parancssorban adunk meg. A mainben meghívjuk a ExorTitkosító classt amit felül fejtettünk ki. Újdonság továbbá a try és a catch használata is ami a try hiba esetén dobja a folyamatot tovább és ami alatta van azt nem hajtja végre hanem a catch blokkban lévő utasítás fog végrehajtódni. A program futtatásához Linuxon szükségünk van javára ezért ezt fel kell telepíteni az alábbi parancs használtaával **sudo apt-get install openjdk-8-jdk** nyilván mivel javát használunk eszért a java fordítójával fordítunk az alábbi módon: **javac totkosito.java**. Futtatni **java titkosito *a 8számjegyű kód* <*titkosítando*.txt > *titkosított*.txt**.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

```
PageRank
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 5
#define _GNU_SOURCE
//Nevesített konstansok.

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

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

  return (double) titkos_meret / sz;
}
//"Megvizsgálja", hogy mekkora egy átlagos magyar szónak a hossza.

int
tiszta_lehet (const char *titkos, int titkos_meret)
{


  double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}
//Magyar szöveget titkosítunk akkor esélyes, hogy benne lesznek az alábbi ↩
   szavak amennyiben a titkosított szövegönben nincs ilyesmi akkor buktuk a ↩
    törést.

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

  int kulcs_index = 0;

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

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

    }

}

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

  exor (kulcs, kulcs_meret, titkos, titkos_meret);

  return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

  char kulcs[KULCS_MERET];
  char titkos[MAX_TITKOS];
  char *p = titkos;
  int olvasott_bajtok;
  char kod [26] {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o ↩
     ','p','q','r','s','t','u','v','w','x','y','z'};

  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';

  // Végigmegyünk az összes lehetséges kulcson és mindegyiknek az  ↩
     eredményért kiírjuk a standard outra amit kacsacsőrökkel tudunk  ↩
     átirányítani.
  for (int ii = 0; ii <= kod.size(); ++ii)
    for (int ji = 0; ji <= kod.size(); ++ji)
      for (int ki = 0; ki <= kod.size(); ++ki)
    for (int li = 0; li <= kod.size(); ++li)
      for (int mi = 0; mi <= kod.size(); ++mi)
          {
            kulcs[0] = kod[ii];
            kulcs[1] = kod[ji];
            kulcs[2] = kod[ki];
```

```
            kulcs[3] = kod[li];
            kulcs[4] = kod[mi];

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

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

  return 0;
}
```

Az előző kódban megtanultunk titkosítani, de miért titkosítsunk, ha nem tudjuk utána feltörni. A fenti kód egy "nyers erőt" használó törőprogram ami minden lehetséges esetet bepróbál és aztán az eredményt kinyomja a standard outra amit majd terminálból akár át is irányíthatunk. Az átlagos szóhossz függvényünk kiszámolja a bemeneti fájl átlagos szóhozzát. Azt követő függvény pedig megadja, hogy milyen sorrendben kell a szavaknak következniük, hogy a várható eredmény visszajöjjön ezért nem is nagyon alkalmas akármit feltörni, tudni kell mit titkosítottunk, hoszen ha a tiszta lehet fügvény szavai nem szerepelnek akkor semmit nem outputol a program.

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Itt most az a lényeg, hogy megtanítsuk a gépet ezekre a számunkra alap dolgokra mint például és or és exor műveletek. Ezt neurális háló felhasználásával érjük el mégpedig R-be. Tehát a gép tulajdonképpen sok lépés után megtanulja magától használni ezeket a lépéseket.

```
  library(neuralnet)

a1    <- c(0,1,0,1) //hais és igaz állások .
a2    <- c(0,0,1,1)
OR    <- c(0,1,1,1) //az és művelet viszsatérési értékei.

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])
```

Láthatjuk, hogy futtatás után a számítógép egész jó hibatűrési határral meg tudja tanulni a gép az vagy műveletet. Ahol kellett 0-hoz és 1hez közelítő elredményeket kapunk.

[1,] 0.00117009

[2,] 0.99986988

[3,] 0.99912751

[4,] 1.00000000

Ugyan ilyen módszerrel a nagyon jól elsajátítja az és műveletet is

```
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])
```

Azomban amikor az exor művelethez érünk -ami akkor vált igenre amikor a két kapott érték külömböző- akkor már azt tapasztaljuk, hogy a neurális hálónk ezt nem képes megtanulni. Ezért régen sokan elfordultak a használatától.

```
library(neuralnet)

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])
```

Látjuk, hogy sen nem 1-hez se nem 0-hoz közelítő értékeket nem kaptunk.

[1,] 0.4999981

[2,] 0.4999980

[3,] 0.5000008

[4,] 0.5000007

Semmi gond van megoldás. Rejtett rétegeket kell használni.

```
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])
```

Mostmár normális eredményeket kapunk.

[1,] 0.0003545297

[2,] 0.9668830788

[3,] 0.9435458430

[4,] 0.0004843918

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/Perceptron

A lényeg, hogy a programnak meg kell találnia és külömböztetni a piros pixeleket ezek a vonal alatt vannak és a fekete pontok közül melyek a vonal felett vannak. A feladataban 3 fájlra bontottuk a kódot. A mandel.cpp-vel legenerálunk egy manderbolt halmazt. Ezt a következő fejezetben bővebben kifejtjük majd. Magát a programot még nem tudjuk lefuttatni hiszen ha megnézzük akkor a gépünk számára még ismeretlen png++ függvénykönyvtárat használunk. A **sudo apt-get install libpng++-dev** parancsal ezt a problémát tudjuk orvosolni. A program maga csak 2 db fájlból épül fel ezek pedig a main.cpp és a ml.hpp. Fordításokr ezekre ügyelni kell. A végeredményt az ml.hpp-ben lévő perceptron osztály fogja kiszámolni.

5. fejezet

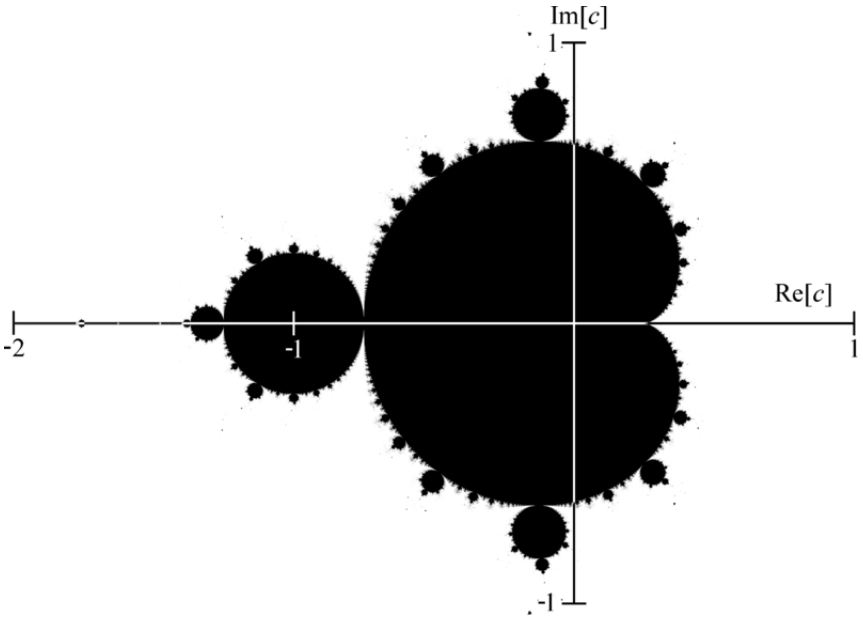
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

A manderbolt halmazt komplex számsíkon értelmezzük. A komplex számok gyakorlatilag számpárok melyek egy valós számrészből és egy képzetes részből állnak amit i-nek nevezünk. Lényeg, hogy a gyök -1-et tudjuk valahol ábrázolni. A komplex számsíkon ki lenet számolni pl az olyan másodfokú eggyenletet ahol pl a gyök alatt minusz lenne. Magát a halmazt 1980-ban Benoit Mandelbrot fedezte fel. Ez a halmaz lényegében azokat a komplex számokat tartalmazza amelyek nem tartanak a végtelenbe. A halmaz kiszámításához szükséges képlet pedig a következő: $z_{n+1}=z_n{}^2+c$ (0<=n) a kiszámítás menete pedig úgy történik, hogy úgy, hogy a választott rácspontok minden egyes pontját (képen látható 800x800) megvizsgáljuk az előbb említett képlettel ráillesztjük és ha ez kivezet a 2 sugarú körből akkor az az elem nem része a halmaznak, ha pedig része akkor kiszínezzük. Ezt sokáig tudjuk folytatni de nem vagyunk képesek végtelensok pontot kiszámolni ezért csak végessok pontot számolunk ki.



Az alábbi kód pedig megvalósiítja a halmaz kiszálítását.

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

#define MERET 600
#define ITER_HAT 32000

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

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

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

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
```

```
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;

                ++iteracio;

            }

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

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

    delta = clock () - delta;
    std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;

}

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

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

    int kepadat[MERET][MERET];

    mandel(kepadat);

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

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

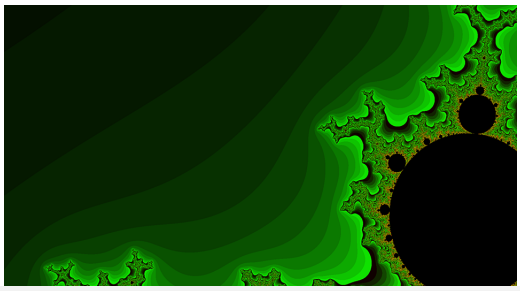
```
                                                    ,
                                                255 -
                                                (255 * kepadat[j][k]) / ITER_HAT ↩
                                                   ));
        }
    }

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

}
```

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

Megoldás videó:

Megoldás forrása:

A processzorunk 1 magját kihasználva számoljuk ki a halmazt. e A programunkat a következő módon tudjuk lefordítani : **g++ mandelpngt.c++ -lpng16 -O3 -o mandelpngt** majd a futtatásnál **./mandelpngt kifile.png 1920 1080 2040 -0.68453684486 -0.065465987864543 0.6549832465 0.98484798846** meg kell adnunk a kimenet fájlt ahová le fogja generálni a halmaz képét a program. A programban használtuk az std komplex osztályt is amit indlude-olnunk kell **#include <complex>**. A program a következő képet generálta. A színezéshez a következő metódust használjuk: minnél később lép ki a körből annál sötétebb a szín. A forráskód a kép alatt található.



```
#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;
```

```
//Itt adunk meg alapértelmezett értékeket de ezt a parancssorból könnyedén ↩
   felülírhatjuk. Egy is kapcsolóval beállítjuk, hogy ha pontosan 9 ↩
   parancssori argumentumot kap akkor felülbírálja az alapértelmezett ↩
   értékeinket. De miért pont 9 ha csak 7 esetet kezelünk le. Ez azért van ↩
   mert az agrv[0] argumentum az maga futtatott fájl neve lesz az agrv[1] ↩
   pedig a kimenet fájl neve lesz amit majd később használunk fel. Az az ↩
   eset is le van kezelve amikor nem pontosan elég argumentumot kap ekkor ↩
   kiírja a teendőket majd kilép.

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

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

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

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

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

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

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

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

              ++iteracio;
            }

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

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

//Kiírjuk a felhasználónak, hogy elkészültünk és mentettünk

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

}
```

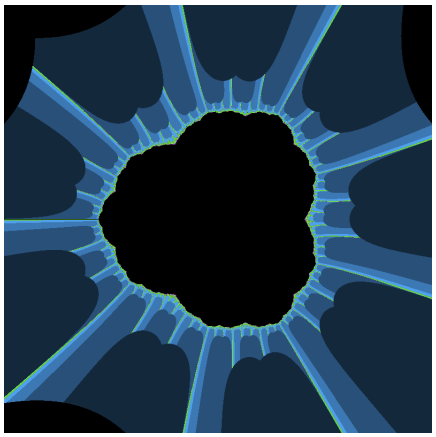
5.3. Biomorfok

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

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

Tisztázzuk, hogy mik a külömbség a két halmaz között. A biomorfok és a mamdeebolt halmaz között mindössze annyi a külömbség, hogy amég a manderbolt halmaz esetében a C az egy változó, azomban a biomorfok esetében pedig ez egyállandó érték. Lényegében a biomorfok Júlia halmazok először egy francia matamatikus dolgozott ezzel Gaston Julia (1893-1978). Magukat a biomorfokat véletlenül találták meg egy programozási bug-ként. A biomorfok nagy népszerűségnek örvendenek a computer grafikában hiszen szép képeket lehet velük csinálni nagyon komplexeket viszonylag egyszerű formulák alkalmazásával.

Az alábbi képet is így készítettük



5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

Lényegében az előző feladat annyi külömbséggel, hogy itt most a videókártya fogja elvégezni a számítási munkákat nem a processzorunk 1 magja fogunk hagyatkozni, hanem az NVIDIA GPU-ban találhato CUDA magokok fogjuk párhuzamosítva elvégezni a számolást. A legfőbb előny pedig az, hogy számottevően gyorsabban fogjuk tudni kiszámolni a halmaznukat. Hiszen az előzőnél egyetlen 1 mag dolgozott itt a munka szétoszlik a CUDA magok között. 50-70 % os gyorsulást lehet ezzel elérni. A végeredmény ugyan az egy szép manderbolt halmaz. A forráskódot lent tekinthetjük meg.

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

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


#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
  // Végigzongorázza a CUDA a szélesség x magasság rácsot:
  // most eppen a j. sor k. oszlopaban vagyunk

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

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

```
  // Hány iterációt csináltunk?
  int iteracio = 0;

  // c = (reC, imC) a rács csomópontjainak
  // megfelelő komplex szám
  reC = a + k * dx;
  imC = d - j * dy;
  // z_0 = 0 = (reZ, imZ)
  reZ = 0.0;
  imZ = 0.0;
  iteracio = 0;
  // z_{n+1} = z_n * z_n + c iterációk
  // számítása, amíg |z_n| < 2 vagy még
  // nem értük el a 255 iterációt, ha
  // viszont elértük, akkor úgy vesszük,
  // hogy a kiinduláci c komplex számra
  // az iteráció konvergens, azaz a c a
  // Mandelbrot halmaz eleme
  while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
      // z_{n+1} = z_n * z_n + c
      ujreZ = reZ * reZ - imZ * imZ + reC;
      ujimZ = 2 * reZ * imZ + imC;
      reZ = ujreZ;
      imZ = ujimZ;

      ++iteracio;

    }
  return iteracio;
}


/*
__global__ void
mandelkernel (int *kepadat)
{

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

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

}
*/

__global__ void
mandelkernel (int *kepadat)
{
```

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

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

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

}

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

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

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

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

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

}

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

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

  if (argc != 2)
    {
      std::cout << "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)
    {
      //sor = j;
      for (int k = 0; k < MERET; ++k)
    {
      kep.set_pixel (k, j,
             png::rgb_pixel (255 -
                     (255 * kepadat[j][k]) / ITER_HAT,
                     255 -
                     (255 * kepadat[j][k]) / ITER_HAT,
                     255 -
                     (255 * kepadat[j][k]) / ITER_HAT));
    }
    }
  kep.write (argv[1]);

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

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

  delta = clock () - delta;
  std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;

}
```

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/mendel%20mozgato%20c%2B%2B

Megoldás videó:

A programunk segítségével képesek vagyunk belenagyítani a halmazba, hiszen ez egy végtelen halmaz. Minden nagyításnál újraszámojuk a halmazt hiszen ez végtelen elemet ratalmaz melyenk végessok elemét számoltuk ki. A feladat magoldásához QTguit fogunk használni amely egy eléggé elterjedt grafikus interfész a C++ nyelvet használók körében. Minnél tovább nagyítjuk annál lassabb lesz a számolás de azt fogjuk észrevenni, hogy egy idő után újjabb és újjabb manderbolt halmazok fognak feltűnni és ez megy a végtelenségig.

Ahoz, hogy hozzá tudjunk kezdeni a fordításhoz fel kell tennünk ezt: **sudo apt-get install libqt4-dev** majd igénybe is vehetjük a **qmake -project** ez csinálni fog egy *.pro fájlt ahol * a mappa neve lesz (ne legyen benne szóköz) ezt a fájlt mág módosítanunk kell. A **INCLUDEPATH += .** sor alá be kell illesztenünk a

következő kis kiegészítésünket: **QT += widgets** mentsük a fájlt. Alábbi parancsot kell beírni a terminálba **qmake *.pro** ahol a * a .pro kiterjesztésű fájl neve (ne legyen szóköz az nem jó). Ezek után egy make parancs szükséges amely elkészíti a végső futtatható állományunkat. Már csak annyi dolgunk van, hogy lefutassuk ./-el.

Bal egérgombot lenyomva tartva illetve az egeret húzva tudunk majd nagyítani és az n bullentyűvel növelhetjük az iterációs határt.

5.6. Mandelbrot nagyító és utazó Java nyelven

link: hhttps://github.com/BenyBalazs/Prog1/tree/master/mandel_javas_zoom

Az előző feladat annyi külömbséggel, hogy itt javában fogunk dolgozni. A kód hasonló lesz átírva javába. A program futtatásához szükségünk lesz javára amelyet az alábbi parancsal telepíthetünk **sudo apt install default-jdk** A fordítás egyszerűen a **javac MandelbrotHalmazNagyító.java** parancsal történik majd a **java MandelbrotHalmazNagyító.java** parancsal futtathatjuk. A programunk van egy bug ami azt eredményezi, hogy új ablakot nyit.

```
/*
 * MandelbrotHalmaz.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt kiszámoló és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class MandelbrotHalmaz extends java.awt.Frame implements Runnable {
    /** A komplex sík vizsgált tartománya [a,b]x[c,d]. */
    protected double a, b, c, d;
    /** A komplex sík vizsgált tartományára feszített
     * háló szélessége és magassága. */
    protected int szélesség, magasság;
    /** A komplex sík vizsgált tartományára feszített hálónak megfelelő kép ↩
       .*/
    protected java.awt.image.BufferedImage kép;
    /** Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
     * (tk. most a nagyítási pontosság) */
    protected int iterációsHatár = 255;
    /** Jelzi, hogy éppen megy-e a szamítás? */
    protected boolean számításFut = false;
    /** Jelzi az ablakban, hogy éppen melyik sort számoljuk. */
    protected int sor = 0;
    /** A pillanatfelvételek számozásához. */
    protected static int pillanatfelvételSzámláló = 0;
```

```
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló
     * <code>MandelbrotHalmaz</code> objektumot.
     *
     * @param      a              a [a,b]x[c,d] tartomány a koordinátája.
     * @param      b              a [a,b]x[c,d] tartomány b koordinátája.
     * @param      c              a [a,b]x[c,d] tartomány c koordinátája.
     * @param      d              a [a,b]x[c,d] tartomány d koordinátája.
     * @param      szélesség      a halmazt tartalmazó tömb szélessége.
     * @param      iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmaz(double a, double b, double c, double d,
            int szélesség, int iterációsHatár) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
        this.szélesség = szélesség;
        this.iterációsHatár = iterációsHatár;
        // a magasság az (b-a) / (d-c) = szélesség / magasság
        // arányból kiszámolva az alábbi lesz:
        this.magasság = (int)(szélesség * ((d-c)/(b-a)));
        // a kép, amire rárajzoljuk majd a halmazt
        kép = new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        // Az ablak bezárásakor kilépünk a programból.
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
            }
        });
        // A billentyűzetről érkező események feldolgozása
        addKeyListener(new java.awt.event.KeyAdapter() {
            // Az 's', 'n' és 'm' gombok lenyomását figyeljük
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
                    pillanatfelvétel();
                // Az 'n' gomb benyomásával pontosabb számítást végzünk.
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    if(számításFut == false) {
                        MandelbrotHalmaz.this.iterációsHatár += 256;
                        // A számítás újra indul:
                        számításFut = true;
                        new Thread(MandelbrotHalmaz.this).start();
                    }
                // Az 'm' gomb benyomásával pontosabb számítást végzünk,
                // de közben sokkal magasabbra vesszük az iterációs
                // határt, mint az 'n' használata esetén
```

```
                } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_M) {
                    if(számításFut == false) {
                        MandelbrotHalmaz.this.iterációsHatár += 10*256;
                        // A számítás újra indul:
                        számításFut = true;
                        new Thread(MandelbrotHalmaz.this).start();
                    }
                }
            }
        });
        // Ablak tulajdonságai
        setTitle("A Mandelbrot halmaz");
        setResizable(false);
        setSize(szélesség, magasság);
        setVisible(true);
        // A számítás indul:
        számításFut = true;
        new Thread(this).start();
    }
    /**
     * A halmaz aktuális állapotának kirajzolása.
     */
    public void paint(java.awt.Graphics g) {
        // A Mandelbrot halmaz kirajzolása
        g.drawImage(kép, 0, 0, this);
        // Ha éppen fut a számítás, akkor egy vörös
        // vonallal jelöljük, hogy melyik sorban tart:
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
    }
    // Ne villogjon a felület (mert a "gyári" update()
    // lemeszelné a vászon felületét).
    public void update(java.awt.Graphics g) {
        paint(g);
    }
    /**
     * Pillanatfelvételek készítése.
     */
    public void pillanatfelvétel() {
        // Az elmentendő kép elkészítése:
        java.awt.image.BufferedImage mentKép =
                new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        java.awt.Graphics g = mentKép.getGraphics();
        g.drawImage(kép, 0, 0, this);
        g.setColor(java.awt.Color.BLUE);
        g.drawString("a=" + a, 10, 15);
        g.drawString("b=" + b, 10, 30);
```

```
        g.drawString("c=" + c, 10, 45);
        g.drawString("d=" + d, 10, 60);
        g.drawString("n=" + iterációsHatár, 10, 75);
        g.dispose();
        // A pillanatfelvétel képfájl nevének képzése:
        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("MandelbrotHalmaz_");
        sb.append(++pillanatfelvételSzámláló);
        sb.append("_");
        // A fájl nevébe belevesszük, hogy melyik tartományban
        // találtuk a halmazt:
        sb.append(a);
        sb.append("_");
        sb.append(b);
        sb.append("_");
        sb.append(c);
        sb.append("_");
        sb.append(d);
        sb.append(".png");
        // png formátumú képet mentünk
        try {
            javax.imageio.ImageIO.write(mentKép, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * A Mandelbrot halmaz számítási algoritmusa.
     * Az algoritmus részletes ismertetését lásd például a
     * [BARNSLEY KÖNYV] (M. Barnsley: Fractals everywhere,
     * Academic Press, Boston, 1986) hivatkozásban vagy
     * ismeretterjesztő szinten a [CSÁSZÁR KÖNYV] hivatkozásban.
     */
     public void run() {
        // A [a,b]x[c,d] tartományon milyen sűrű a
        // megadott szélesség, magasság háló:
        double dx = (b-a)/szélesség;
        double dy = (d-c)/magasság;
        double reC, imC, reZ, imZ, ujreZ, ujimZ;
        int rgb;
        // Hány iterációt csináltunk?
        int iteráció = 0;
        // Végigzongorázzuk a szélesség x magasság hálót:
        for(int j=0; j<magasság; ++j) {
            sor = j;
            for(int k=0; k<szélesség; ++k) {
                // c = (reC, imC) a háló rácspontjainak
                // megfelelő komplex szám
```

```
                reC = a+k*dx;
                imC = d-j*dy;
                // z_0 = 0 = (reZ, imZ)
                reZ = 0;
                imZ = 0;
                iteráció = 0;
                // z_{n+1} = z_n * z_n + c iterációk
                // számítása, amíg |z_n| < 2 vagy még
                // nem értük el a 255 iterációt, ha
                // viszont elértük, akkor úgy vesszük,
                // hogy a kiinduláci c komplex számra
                // az iteráció konvergens, azaz a c a
                // Mandelbrot halmaz eleme
                while(reZ*reZ + imZ*imZ < 4 && iteráció < iterációsHatár) {
                    // z_{n+1} = z_n * z_n + c
                    ujreZ = reZ*reZ - imZ*imZ + reC;
                    ujimZ = 2*reZ*imZ + imC;
                    reZ = ujreZ;
                    imZ = ujimZ;

                    ++iteráció;
                }
                // ha a < 4 feltétel nem teljesült és a
                // iteráció < iterációsHatár sérülésével lépett ki, azaz
                // feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c
                // sorozat konvergens, azaz iteráció = iterációsHatár
                // ekkor az iteráció %= 256 egyenlő 255, mert az esetleges
                // nagyítasok során az iteráció = valahány * 256 + 255
                iteráció %= 256;
                // így a halmaz elemeire 255-255 értéket használjuk,
                // azaz (Red=0,Green=0,Blue=0) fekete színnel:
                rgb = (255-iteráció)|
                        ((255-iteráció) << 8) |
                        ((255-iteráció) << 16);
                // rajzoljuk a képre az éppen vizsgált pontot:
                kép.setRGB(k, j, rgb);
            }
            repaint();
        }
        számításFut = false;
    }
    /**
     * Példányosít egy Mandelbrot halmazt kiszámoló obektumot.
     */
    public static void main(String[] args) {
        // A halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35] tartományában
        // keressük egy 400x400-as hálóval:
        new MandelbrotHalmaz(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/polargen

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

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

int
main ()
{
  PolarGen pg;

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

  return 0;
}
```

Inculdeoljuk az iostreamot és a polargen.h fejlécfájlt amit mi írtunk meg. Létrehozunk egy PolarGen típusú válltozót amit pg-nek nevezünk el. 10x kiiratjuk a polargen doubble következőben található értéket lényegében a program main részéről ennyit.

```
#ifndef POLARGEN__H
#define POLARGEN__H

#include <cstdlib>
```

```
#include <cmath>
#include <ctime>

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

private:
  bool nincsTarolt;
  double tarolt;

};

#endif
```

Itt Deklaráltuk a PolarGen osztályt és változóit amiből van publikus és privát a privátot csak az osztályon belül érhetjük el. A randomszámgeneráláshoz meghívjuk a standard libraryból az srand függvényt melynek alap az aktuális gépi idő lesz. Illetve ledeklaráltuk, hogy a double kovetkezo (); függvényt amit majd a következő cpp-ben fejtünk ki.

```
#include "polargen.h"

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

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

      tarolt = r * v2;
      nincsTarolt = !nincsTarolt;
```

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

Kifejti a double kovetkezo (); függvényt. 2 eset lehetséges ha belelép az if fügvénybe akkor az r * v1 értéket fogja visszatéríteni és a nyncsTarolt bool válltozót az ellenkezőjére állítja magyarul letagadja. Ha nem lépünk bele az if függvénybe akkor pedig visszaadjuk a tarolt válltozóban található értéket és itt is az ellenkezőjére állítjuk a bool válltozónk értékét.

```
    public class PolárGenerátor {
    boolean nincsTárolt = true;
    double tárolt;

    public PolárGenerátor() {
        nincsTárolt = true;
    }

    public double következő() {
        if (nincsTárolt) {
            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2*u1-1;
                v2 = 2*u2-1;
                w = v1*v1+v2*v2;
            } while (w > 1);
            double r = Math.sqrt((-2*Math.log(w)) / w);
            tárolt = r*v2;
            nincsTárolt = !nincsTárolt;
            return r*v1;
        } else {
            nincsTárolt = !nincsTárolt;
            return tárolt;
        }
    }

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

Ugyan azt a polártranszformációs algoritmust írtuk meg csak C++ nyelv helyett Java nyelven. A működési elv ugyan az mint a fenti C++-os porgram esetében. A két forráskódot összehasonlítva észrevehetjük, hogy a javás verzió sokkal rövidebb a C++-nál ez nannak köszönhető, hogy amég C++-ban bonyolult randomszámgenerátort kellet írnunk addig Javában ezt az egész folyamataot tartalmazza a Math.random() függvény.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/LZW%20binfa/sima%20c

Az elején tisztázzuk, hogy mi is az a bineáris fa. Nos ez egy informatikában használt adatszerkezet. Középen áll a gyökér (Root) -egyes elemekre később a Node angol szót is használom- és ennek a gyökérnek maximum 2 ága lehet ezeket nevezzük Bal oldali gyereknek (left child) illetve jobb oldali gyereknek (right child). Ezeknek további két alága lehet és így tovább és így tovább. A gyerekek száma maximum 2 ebből következik, hogy olyan eshetőséggel találkozunk amikor csak 1 gyerek van egy node-nak ebben az esetben a nem létező gyerek az NULL. Ahol véget ér a fa tehát nincs több gyerek azt "levél node"-nak hívjuk (leaf node). Szigorú bineáris fának nevezzük azt az esetet amikor 1 csomópontnak csak 2 vagy 0 gyereke lehet. Adatrendezésre használják a bineáris fákat.

Amit ebben a konkrét esetben használunk az az LZW algoritmus. Az egy tömörítési algoritmus melynek teljes neve Lempel-Ziv-Welch. Ebből a a faépítő szegmense kell nekünk melyet C porgramozási nyelven írtunk meg. A programunk magától ne fog lefordulni gcc fordítóval sírni fog az sqrt-re. GCC-vel így lehet lefordítani **gcc z.c -lm -o z**

```
#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 ("altag=%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);
    }
}
```

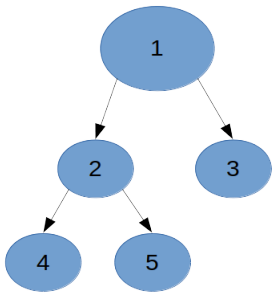
6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/LZW%20binfa/prepost

A bineáris fánkat többféleképpen is bejárhatjuk lehet inorder (ami az eredeti kódba van) és azt jelenti, hogy sorba. Jobbról balra halad végig a fán. Preorder bejárásnál először a gyökeret majd a bal oldali részét aztán a jobb oldali részét nézzük meg. Postorder bejárás esetében pedig először a bal majd a jobb oldali ágat végül a gyökeret nézzük. Erre nézzünk egy példafát a könnyebb megértés kedvéért.



Inorder (Bal, Gyöker, Jobb) : 4 2 5 1 3

Preorder (Gyökér, Bal, Jobb) : 1 2 4 5 3

Postorder (Bal, Jobb, Gyökér) : 4 5 2 3 1

A program módisított kiir függvénye a megfeleő bejárásokhoz igazítva.

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

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

            ++melyseg;                                //Bal oldali ág
            kiir (elem->egyesGyermek(), os);

            kiir (elem->nullasGyermek(), os);
            --melyseg;                                //Jobb oldali ág
        }
    }
```

```
postorder
void kiir (Csomopont* elem, std::ostream& os)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ↩
           leállítása
        if (elem != NULL)
        {
            ++melyseg;                                //Bal oldali ág
            kiir (elem->egyesGyermek(), os);

            kiir (elem->nullasGyermek(), os);
            --melyseg;                                //Jobb oldali ág

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

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/blob/master/LZW%20binfa/z3a7nocomment.cpp

Az első feladatban megismert LZW binfa algoritmust írtuk át C++ nyelvre. Itt a Csomópont osztály gyökér válltozólyát tagként szerepeltetjük a kódban és mindenhol máshol referenciaként hivatkozunk rá. Így ez az elem mindíg benn van a memóriában. Az LZWBinfa osztály védett részében található meg a gyökér válltozó. Láthatjuk azt is, hogy a kódban a csomópont osztály az az LZWBinfa osztály alá van beágyazva ezt azért csináltuk így mert a kódban nem szánukn neki külön szerepet csak a bineáris fánk építőelemének használjuk. Futtatása a következőképp történik: **./*a_program_neve* befile.txt -o kifile.txt** A teljes forráskód a fenti linken tekinthető meg.

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



class LZWBinFa
{
public:
    LZWBinFa (): fa(&gyoker) {}

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

```
    void kiir (void)
    {

        melyseg = 0;
        kiir (&gyoker, std::cout);
    }
    void szabadit (void)
    {
        szabadit (gyoker.egyesGyermek());
        szabadit (gyoker.nullasGyermek());
    }

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

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

private:
    class Csomopont
    {
    public:

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

```
    private:

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

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

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

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

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

};
```

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

    rszoras (&gyoker);

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

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

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

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

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

    if (argc != 4) {
        usage();
        return -1;
    }

    char *inFile = *++argv;

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

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

    unsigned char b;
    LZWBinFa binFa;
```

```
    while (beFile.read ((char *) &b, sizeof (unsigned char))) {
        for (int i = 0; i < 8; ++i)
        {

            int egy_e = b & 0x80;

            if ((egy_e >> 7) == 1)

                binFa << '1';
            else
                binFa << '0';
            b <<= 1;
        }

    }

    kiFile << binFa;

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

    binFa.szabadit ();

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

    return 0;
}
```

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/blob/master/LZW%20binfa/z3a2.cpp

Az előző feladatban megismert programot fogjuk egy kicsit átalakítani, mégpedig úgy, hogy a fent megismert csomópont gyökér válltozó nem tagként hanem mutatóként fog szerepelni. Ezt úgy fogjuk elérni, hogy

```
protected:

    Csomopont gyoker;
...
};
```

helyett

```
protected:

    Csomopont *gyoker;
...
};
```

szerepeljen.

Ez nyilvánvaló hibákhoz vezet melyekre a fordító felhívja a figyelmünket is. Lényegében a könnyebb része ez, hogy a fordító álltal kidobott hibákat kijavítsuk. Minden &gyökeret átírunk szimpla gyökérré. És még

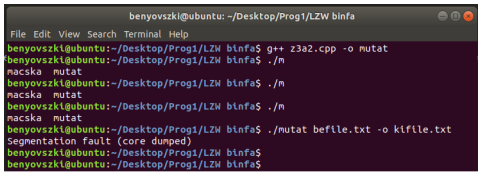
```
void szabadit (void)
    {
        szabadit (gyoker.egyesGyermek());
        szabadit (gyoker.nullasGyermek());
```

Sort kicsréljülk arra, hogy

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

Higyen a gyökér az már nem tagként szerepel ezért ponttal nem lehet rá hivatkozni. a pointer típushoz azt a kis nyilat (->) használjuk mert a mutató mutatóit akarjuk elérni. Az így kapot kód már minden gond nélkül lefordul a **g++** fordítóval. És a fent megemlített parancsal lehet is futtatni.

Meg is kaptuk a szép hibaüzenetet miszerint szegmentásási hiba történt. Ez azért következett be mert nem foglaltunk helyet a memóriában.



Ezért még az LZWBinfa konstruktorát módosítanunk kell a következőképpen.

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

Itt a gyökeret egy új csomópontnak állítjuk be majd a fa válltozót ráállítjuk a gyökérre.

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/blob/master/LZW%20binfa/mozgatoszemantika.cpp

A mozgató szemantika lényege, hogy ne másolgassuk és bontogassul ke többször a fánkat, hanem az, hogy egybe az egészet át tudjuk mozgatni. Ez a mazgatás művelet kevesebb számolási terhet ró a processzorra könnyebb végrehajtani mint mondjuk egy másolást ez azért van mert nem mindíg kelll nekünk feltétlenül megőrizni az eredetit meg a msáolatot sok esetben elég a másolat. Gondoljuka a kivágás beillesztés vagy a másolás beillesztés külömbségére. Ahoz, hogy el tudjuk végezni magát a másolást kell írnunk egy mozgató konstruktort (move constructor) és egy mozgató értékadást (move assignment) mert maga az std::move csak felkészíti az objektumot a mozgatásra nem hajtja azt végre. A *this pedig az új mozgatott fánkat fogja visszaadni.

```
      LZWBinFa (LZWBinFa&& original)
    {
        std::cout<<"Move ctor\n";
        gyoker = nullptr;
        *this = std::move(original);

}
```

```
 LZWBinFa& operator= (LZWBinFa&& original)
    {
        std::cout<<"Move assignment ctor\n";
        std::swap(gyoker, original.gyoker);
        return *this;
}
```

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/Conway/Myrmecologist

Az első nehézséget a megfelelő QT gui verzió beszerzése jelenti.

sudo apt-get install build-essential

sudo apt-get install qtcreator

sudo apt-get install qt5-default

A kis kitekintés után térjünk egyből a lényegre. A hangyakolónia algoritmus mögötti ihletet az Az algoritmus az igazi hangyák viselkedéséről lett mintázva, lényege, hogy megtalálja a legoptimálisabb utat a két pont között, hogy minnél efektívebben és gyorsabban lehessen utazni a kettő között. Ezt a való életbe például egy csomagküldő vállalat tudja igen jól kihasználni. Akiknek a dolgozói hosszú és komplikált utakat kell bejárniuk. Ilyesfajta hangyakolóniás módszerrek sokat tudnak spórolni. A program futásakor észlelhetjük, hogy a hangyákul szolgáló pontok össze vissza mennek, azmoban egy kicsivel később már szabályos pályákat vesznek fel pontok között.**./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s 3 -c 22** parancsal futtatva indítsuk a programot.

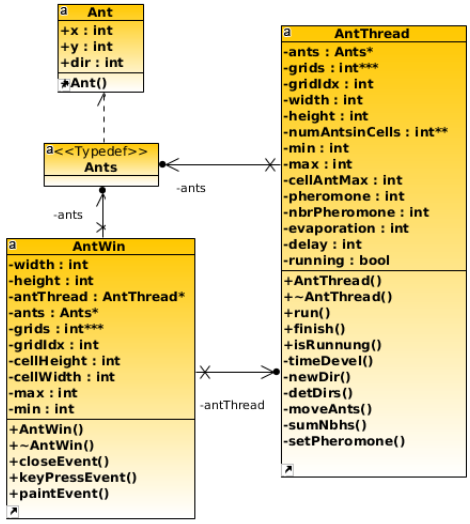
```
    QCommandLineOption szeles_opt ( {"w","szelesseg"}, "Oszlopok (cellakban ↩
       ) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( {"m","magassag"}, "Sorok (cellakban)  ↩
       szama.", "magassag", "150" );
    QCommandLineOption hangyaszam_opt ( {"n","hangyaszam"}, "Hangyak szama. ↩
       ", "hangyaszam", "100" );
    QCommandLineOption sebesseg_opt ( {"t","sebesseg"}, "2 lepes kozotti  ↩
       ido (millisec-ben).", "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" );
```

A sorok közepén lehet látni, hogy a parancssori argumentumok hova karülnek és az elején, hogy meyik mi volt. A végén pedi azt az értéket ami akkor kerül a programba ha a parancssorról nem adunk meg más értéket. Tehát magyarul azok az alapértelmezett értékek.

Ez a parancs így magában eléggé semmitmondó úgyhogy nézzük is meg, hogy mit csinálunk vele. Ekezekn sorba végigmegyünk. A forráskód main részében láthatók.názzük is meg programlistingbe.

Még a feladatunk közé tartozott az is, hogy a forráskód alapján készítsünk osztálydiagrammot mely megmutatja, hogy az osztályok között milyen kapcsolat van.



7.2. Java életjáték

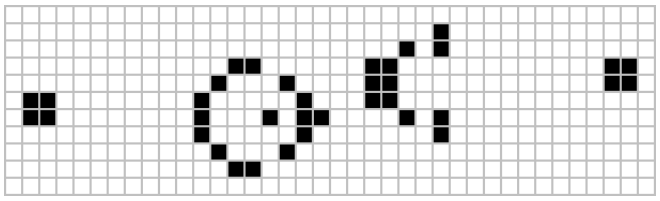
Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/Conway/eletjatek%20java

Az életjáték maga megmagyarázásra kerül a következő pontban itt most a javás külömbségekre és a nevezetes alakzatokra térnénk ki. Ezt az életjátékot John Horton Conway találta ki 1970-ben. Ebben a feladatban java nyelven valósítjuk meg az életjátékot. Eebben az interpretációban tudunk rajzloni a képernyőre. Az életjátékban létrehozhatunk külömböző nevezetes alakzatokat melyek soha nem fognak kihalni és/vagy ismétlődő mozgást végeznek. Ilyen alakzatok lehetnek pl. a végtelen élet ami 4 kocka és a siklo-kilövő

is melyet a lenti ábrán láthatunk(A programunk alapértelnezetten egyilyennel indul). GIF megjelenítésére sajnos nincs lehetőségünk, de az apaphelyzet a lenti kéne látható. Fun fact, hogy ez a kép a hackerek szombóluma.



Az alábbi kód pedig magvalósítja az életjátékot és a kezdőalakzat az a sikólilövő lesz

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

    public Sejtautomata(int szélesség, int magasság) {
        this.szélesség = szélesség;
        this.magasság = magasság;
        rácsok[0] = new boolean[magasság][szélesség];
        rácsok[1] = new boolean[magasság][szélesség];
        rácsIndex = 0;
        rács = rácsok[rácsIndex];
        for(int i=0; i<rács.length; ++i)
            for(int j=0; j<rács[0].length; ++j)
                rács[i][j] = HALOTT;
        siklóKilövő(rács, 5, 60);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
            }
        });

        addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
                    cellaSzélesség /= 2;
                    cellaMagasság /= 2;
                    setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                            Sejtautomata.this.magasság*cellaMagasság);
```

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

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

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

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

    setTitle("Sejtautomata");
    setResizable(false);
    setSize(szélesség*cellaSzélesség,
            magasság*cellaMagasság);
```

```
        setVisible(true);
        new Thread(this).start();
    }

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

        rács[y+ 4][x+ 12] = ÉLŐ;
        rács[y+ 4][x+ 14] = ÉLŐ;

        rács[y+ 5][x+ 11] = ÉLŐ;
        rács[y+ 5][x+ 15] = ÉLŐ;
        rács[y+ 5][x+ 16] = ÉLŐ;
        rács[y+ 5][x+ 25] = ÉLŐ;

        rács[y+ 6][x+ 11] = ÉLŐ;
        rács[y+ 6][x+ 15] = ÉLŐ;
        rács[y+ 6][x+ 16] = ÉLŐ;
        rács[y+ 6][x+ 22] = ÉLŐ;
        rács[y+ 6][x+ 23] = ÉLŐ;
        rács[y+ 6][x+ 24] = ÉLŐ;
        rács[y+ 6][x+ 25] = ÉLŐ;

        rács[y+ 7][x+ 11] = ÉLŐ;
        rács[y+ 7][x+ 15] = ÉLŐ;
        rács[y+ 7][x+ 16] = ÉLŐ;
        rács[y+ 7][x+ 21] = ÉLŐ;
        rács[y+ 7][x+ 22] = ÉLŐ;
        rács[y+ 7][x+ 23] = ÉLŐ;
        rács[y+ 7][x+ 24] = ÉLŐ;

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

        rács[y+ 9][x+ 13] = ÉLŐ;
        rács[y+ 9][x+ 21] = ÉLŐ;
        rács[y+ 9][x+ 22] = ÉLŐ;
        rács[y+ 9][x+ 23] = ÉLŐ;
```

```
        rács[y+ 9][x+ 24] = ÉLŐ;
        rács[y+ 9][x+ 34] = ÉLŐ;
        rács[y+ 9][x+ 35] = ÉLŐ;

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

        rács[y+ 11][x+ 25] = ÉLŐ;

    }

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

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

    public static void main(String[] args) {
        new Sejtautomata(100, 75);
    }
}               g.setColor(java.awt.Color.LIGHT_GRAY);
                g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                        cellaSzélesség, cellaMagasság);
            }
        }

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

```
    public int szomszédokSzáma(boolean [][] rács,
            int sor, int oszlop, boolean állapot) {
        int állapotúSzomszéd = 0;
        for(int i=-1; i<2; ++i)
            for(int j=-1; j<2; ++j)
                if(!((i==0) && (j==0))) {
            int o = oszlop + j;
            if(o < 0)
                o = szélesség-1;
            else if(o >= szélesség)
                o = 0;

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

            if(rács[s][o] == állapot)
                ++állapotúSzomszéd;
                }

        return állapotúSzomszéd;
    }

    public void időFejlődés() {

        boolean [][] rácsElőtte = rácsok[rácsIndex];
        boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];

        for(int i=0; i<rácsElőtte.length; ++i) {
            for(int j=0; j<rácsElőtte[0].length; ++j) {

                int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);

                if(rácsElőtte[i][j] == ÉLŐ) {

                    if(élők==2 || élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
                }  else {

                    if(élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
                }
            }
        }
```

```
        rácsIndex = (rácsIndex+1)%2;
    }

    public void run() {

        while(true) {
            try {
                Thread.sleep(várakozás);
            } catch (InterruptedException e) {}

            időFejlődés();
            repaint();
        }
    }

    public void sikló(boolean [][] rács, int x, int y) {

        rács[y+ 0][x+ 2] = ÉLŐ;
        rács[y+ 1][x+ 1] = ÉLŐ;
        rács[y+ 2][x+ 1] = ÉLŐ;
        rács[y+ 2][x+ 2] = ÉLŐ;
        rács[y+ 2][x+ 3] = ÉLŐ;

    }
   // Alapból berajzolja a sokat emlegetett siklókilövőt.
    public void siklóKilövő(boolean [][] rács, int x, int y) {

        rács[y+ 6][x+ 0] = ÉLŐ;
        rács[y+ 6][x+ 1] = ÉLŐ;
        rács[y+ 7][x+ 0] = ÉLŐ;
        rács[y+ 7][x+ 1] = ÉLŐ;

        rács[y+ 3][x+ 13] = ÉLŐ;

        rács[y+ 4][x+ 12] = ÉLŐ;
        rács[y+ 4][x+ 14] = ÉLŐ;

        rács[y+ 5][x+ 11] = ÉLŐ;
        rács[y+ 5][x+ 15] = ÉLŐ;
        rács[y+ 5][x+ 16] = ÉLŐ;
        rács[y+ 5][x+ 25] = ÉLŐ;

        rács[y+ 6][x+ 11] = ÉLŐ;
        rács[y+ 6][x+ 15] = ÉLŐ;
        rács[y+ 6][x+ 16] = ÉLŐ;
        rács[y+ 6][x+ 22] = ÉLŐ;
        rács[y+ 6][x+ 23] = ÉLŐ;
        rács[y+ 6][x+ 24] = ÉLŐ;
        rács[y+ 6][x+ 25] = ÉLŐ;
```

```
        rács[y+ 7][x+ 11] = ÉLŐ;
        rács[y+ 7][x+ 15] = ÉLŐ;
        rács[y+ 7][x+ 16] = ÉLŐ;
        rács[y+ 7][x+ 21] = ÉLŐ;
        rács[y+ 7][x+ 22] = ÉLŐ;
        rács[y+ 7][x+ 23] = ÉLŐ;
        rács[y+ 7][x+ 24] = ÉLŐ;

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

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

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

        rács[y+ 11][x+ 25] = ÉLŐ;

    }

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

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

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

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/Conway/eletjatek

Folytassuk az életjáték megmagyarázást., A játéknak 4 egyszerű szabálya van és ezek a következők:

1. Egy sejt akkor élheti túl ha 2 vagy 3 szomszédja van.

2. Ha egy üres cellának pontsan 3 szomszédja van akkor szaporodni fog.

3. Amennyiben 3-nál több szomszédja van egy sejtenk akkor az túlnépesedésben pusztu el.

4. Kettőnél kevesebb szomszéd esetében pedig az elszigeteltségben, magányban hal meg a sejt.

Ezek a szabályok egy négyzethálóra vonatkoznak, hiszen ez a sejtjeink "élettere" és mindegyik cellában maximum 1 darab sejt élhet. A Conway féle életjáték hasonlóan viselkedik sok esetben mint az élő szevezethez pl. a szaporodás, kihalás, újászületés terén így a szimulációs játék ellnevezést is megérdemli.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: https://github.com/BenyBalazs/Prog1/tree/master/Conway/BrainB

Ahoz, hogy sikeres legyen a fordítás ahoz kell az open cv melyen az ubuntu repobol a kovetkező parancsal szerezhetünk be. **sudo apt-get install libopencv-dev python3-opencv**. Ezek után a program gondnélkül fordul és fut.

A lényeg az, hogy az egeret rajta kell tartani a megjelenő fekete körökbe. Idő elteltével egyre több fog megjelenni és egyre hevesebb a kör mozgása. A lényeg, hogy minnél tovább tudjuk rajtatartani az egeret a célon. Ezzel mérve a kancentrációt. Hasznossága legfőképpen esportban van legfőképp az ilyne tesztnek értelme pl, hogy mennyire tudjuk a gyorsan válltozó eseményeket követni.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

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

Ezt a feladatot paszoltam.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Ezt a feladatot paszoltam.

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Ez egy ingyenes projekt amelyet a Microsoft hozott kétre és minecraft játékos karakterét programozhatjuk vele, adhatunk neki "küldetéseket". A lényeg, hogy mesterséges intelligenciát írunk minecraftban. A program ingyenesen letölthető a microsoft githubjáról innen. Egy stabil releaset kell letölteni a readme-ben található linkről. A minecraft mappában lévő launchClient.sh-val windows esetében egy .bat fájlt fogunk találni. indul a játék. Problémát okoz ha nem jre8 van a gépünkön.

Utána már csak egy új terminál ablakot kell nyitni amiben lefuttatjuk a parancsainkat melyeket előtte pythonban megírtunk. A python_examples nevű mappában találunk példakódokat melyekkel elkezdhetjük az ismerkedést. Térjünk is át a kódunk elemzésére.

Kezdjük el mozgatni a karakterünket. Ehez a agent_host.sendCommand("valami" [1/-1]) sor felelős a valami helyére kerülhetnek a következők: **move,strafe,pitch,turn,jump,croutch,attack,use**Ezek megfejtése egyszerű angol tudást igényel. A pitch az a kamerát állítja (-)fel meg (+)le a többi szótári fordítás. Nem mindnél lehet -1-et írni a függvénybe iylen pl a jump mert vagy ugrik vagy nem nem tud visszafele ugrani.

Nézzünk egy kis világgenerálást. Itt világgenerálási alapokat és egyéb nyalánkságokat helyezhetünk el. És, hogy ezeket használja is a program ahoz kell ez a sor: **my_mission = MalmoPython.MissionSpec()**

```
missionXML='''<?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>Hello world!</Summary>
              </About>

              <ServerSection>
                <ServerHandlers>
                  <DefaultWorldGenerator //Default vagy Flat generátor  ↩
                     lehet                    generatorString ↩
                     ="3;7,220*1,5*3,2;3;,biome_1"/> //Világgenerálás  ↩
                     alapjai milyen biom legyen
                  <ServerQuitFromTimeUp timeLimitMs="30000"/> //Missziók  ↩
                     idelye.
                  <ServerQuitWhenAnyAgentFinishes/>
                </ServerHandlers>
              </ServerSection>

              <AgentSection mode="Survival"> //Gamemode.
                <Name>MalmoTutorialBot</Name> //Karakterünk neve
                <AgentStart/>
                <AgentHandlers>
                  <ObservationFromFullStats/>
                  <ContinuousMovementCommands turnSpeedDegs="180"/> // ↩
                     Fordulási sebesség
                </AgentHandlers>
              </AgentSection>
            </Mission>'''
```

Akkor lett volna a legegyszerűbb dolgunk ha egy egyszerű sima páylát generáltunk volna mert akkor nem kéne akadályokat kikerülni,de ez a szép a mincraftban, hogy dombos fás és egyéb terepek is léteznek melyeket meg kell tudnia hóditani a programunknak.

Folyamatosan figyeljük a világot és ha van olyan megfigyelt érték akkor átadjuk.

```
if world state.number of observations since last state > 0:
    msg = world_state.observations[-1].text
    observations = json.loads(msg)
    nbr = observations.get("nbr3x3", 0)
    print("Mit latok: ", nbr)

    if "Yaw" in observations:
```

```
        SYaw = observations["Yaw"]
    if "Pitch" in observations:
        SPitch = observations["Pitch"]
    if "XPos" in observations:
        Xkoordinata = observations["XPos"]
    if "ZPos" in observations:
        Zkoordinata = observations["ZPos"]
    if "YPos" in observations:
        Ykoordinata = observations["YPos"]
```

Itt pedig a külömböző esetek, hogy miként kerülje ki a blokkokat a karakter. Merre forduljon.

```
    if SYaw >= 180-22.5 and SYaw <= 180+22.5 :
        elotteidx = 1
        elotteidxj = 2
        elotteidxb = 0

    if SYaw >= 180+22.5 and SYaw <= 270-22.5 :
        elotteidx = 2
        elotteidxj = 5
        elotteidxb = 1

    if SYaw >= 270-22.5 and SYaw <= 270+22.5 :
        elotteidx = 5
        elotteidxj = 8
        elotteidxb = 2

    if SYaw >= 270+22.5 and SYaw <= 360-22.5 :
        elotteidx = 8
        elotteidxj = 7
        elotteidxb = 5

    if SYaw >= 360-22.5 or SYaw  <= 0+22.5 :
        elotteidx = 7
        elotteidxj = 6
        elotteidxb = 8

    if SYaw >= 0+22.5 and SYaw <= 90-22.5 :
        elotteidx = 6
        elotteidxj = 3
        elotteidxb = 7

    if SYaw >= 90-22.5 and SYaw <= 90+22.5 :
        elotteidx = 3
        elotteidxj = 0
        elotteidxb = 6

    if SYaw >= 90+22.5 and SYaw <= 180-22.5 :
        elotteidx = 0
        elotteidxj = 1
```

```
        elotteidxb = 3
```

Az értékek amiket felvehet a Yaw azok lényegébe a szélrózsa értékel az É K NY D nek megvannak a saját értékei fokban. Ezek pedig a következők. É(180) K(-90) NY(90) D(0)

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

Iteratív faktoriális

Ebben az esetben iteratívan írtuk meg a faktoriálist ami azt jelenti, hogy ciklust használtunk a számok összeszorzásához. Majd miután a ciklus lefutott kiírjuk a végeredményt.

```
(defvar zs 1)
(princ "Add meg a faktorialis szamot")
(setq b (read))

(loop for a from 1 to b

    do(setq zs (* a zs))

    )
(write-line "a faktorialis =")
(print zs)
```

Rekurzív faktoriális

Ebben az esetben pedik rekurzívan írtuk meg ez azt jelenti, hogy a fügvény önmagát meghívja. És így számolja ki a faktoriálist. Tehát az elején létrehozunk egy függvényt és benne megadjuk a következőket. Ha a kapott szám kissebb mint kettő akkor

```
(defun faktorialis (b)
    (if (< b 2)
        1
        (* b (faktorialis (- b 1)))))
(print "Adj meg egy számot ")
(setq a (read))
(princ (faktorialis a))/s
```

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

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

;(color-curve)

(define (elem x lista) //Elérjük az x-edik elemet.

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

)

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

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

    (list text-width text-height)
    )
)

// A szöveg kialakításáért felel
```

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

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

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

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

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

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

    (gimp-selection-none image)

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

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

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

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

    ;step 3
```

```
    (gimp-drawable-levels layer HISTOGRAM-VALUE .18 .38 TRUE 1 0 1 TRUE)

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

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

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

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

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

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

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

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

;(script-fu-bhax-chrome-border "Norbert Bátfai" "Sans" 160 1920 1080 400 '
   (255 0 0) "Crown molding" 7)
;(script-fu-bhax-chrome-border "Szenvedés" "Sans" 110 768 576 300 '(255 0
   0) "Crown molding" 6)

//Alapértelmezett értékeket állítunk be a szkriptünkhöz. A User
   változtathat rajta az ablakban.

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

```
    SF-STRING       "Text"      "Norbert Bátfai"
    SF-FONT         "Font"      "Sans"
    SF-ADJUSTMENT   "Font size" '(160 1 1000 1 10 0 1)
    SF-VALUE        "Width"     "1920"
    SF-VALUE        "Height"    "1080"
    SF-VALUE        "New width" "400"
    SF-COLOR        "Color"     '(255 0 0)
    SF-GRADIENT     "Gradient"  "Crown molding"
    SF-VALUE        "Border size"   "7"
)
(script-fu-menu-register "script-fu-bhax-chrome-border" //Ilyen néven kerül ↩
    be a szkriptek közé.
    "<Image>/File/Create/BHAX"
)
```

Ebben a feladatban elkészítettünk a GIMP nevű ingyenes képszerkesztőprogramhoz egy script fájlt ami krómozott betűt készít. Az elkészült scriptet a GIMPBEN az edit preferences scripts helyre kell berakni és utána a file create menüben már látni is fogjuk és használatra kész.

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_hackelese_a_scheme_programozasi_nyelv

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

Ez a kód egy mandalát generál le gimpben. Az előző feladatban megismert felületen állíthatjuk majd be a mandala beállításait. A gimp-layer-resize-to-image-size a réteget újraméretezi úgy, hogy az illeszkedjen a képre.

```
(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
    )
)
```

```
//A szövegméret beállítása

(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)

//Új kép és rétegek létrehozása és megformázása.

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

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

    (gimp-context-set-foreground '(0 255 0))
    (gimp-drawable-fill layer FILL-FOREGROUND)
    (gimp-image-undo-disable image)
```

```
    (gimp-context-set-foreground color)

    (set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ↩
       ))
    (gimp-image-insert-layer image textfs 0 -1)
    (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2))  (/  ↩
       height 2))
    (gimp-layer-resize-to-image-size textfs)

    (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
    (gimp-image-insert-layer image text-layer 0 -1)
    (gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
    (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ↩
       -LAYER)))

    (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
    (gimp-image-insert-layer image text-layer 0 -1)
    (gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
    (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ↩
       -LAYER)))

    (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
    (gimp-image-insert-layer image text-layer 0 -1)
    (gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
    (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ↩
       -LAYER)))

    (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
    (gimp-image-insert-layer image text-layer 0 -1)
    (gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
    (set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ↩
       -LAYER)))

    (plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
    (set! textfs-width (+ (car(gimp-drawable-width textfs)) 100))
    (set! textfs-height (+ (car(gimp-drawable-height textfs)) 100))

    (gimp-layer-resize-to-image-size textfs)

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

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

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

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

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

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

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

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

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

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

    (gimp-display-new image)
    (gimp-image-clean-all image)
    )
)
//Alapértelmezett értékeket állítunk be a szkriptünkhöz.

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

(script-fu-register "script-fu-bhax-mandala"
    "Mandala9"
    "Creates a mandala from a text box."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 9, 2019"
    ""
    SF-STRING       "Text"      "Bátf41 Haxor"
```

```
    SF-STRING       "Text2"     "BHAX"
    SF-FONT         "Font"      "Sans"
    SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
    SF-VALUE        "Width"     "1000"
    SF-VALUE        "Height"    "1000"
    SF-COLOR        "Color"     '(255 0 0)
    SF-GRADIENT     "Gradient"  "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"  //Ilyen néven kerül be a ↩
    szkriptek közé.
    "<Image>/File/Create/BHAX"
)
```

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

3 féle programozási nyelvet külömböztetünk meg. Létezik a gép nyelv az assembly szintű nyelv illetve a magas szintű nyelv. A kurzuson a magas szintű programozási nyelvekkel foglalkozunk. Ezeknek a nyelveknek van szemantikai (tartalmi, értelmezési, jelentésbeli szabályok) és szintaktikai (forráskód összeállítási szabályai) részük. A processzor ezekenk az utasításait közveltlenül nem tudja értelmezni ezért szükségünk van egy fordítóprogramra angolul compiler amely a magas szintű kódból gépi kódot állít elő. Ha bármilyen szintaktikai hibát talál a fordító akkor a programot nem lehet lefordítani, hibát fogunk kapni. A programozási nyelveket 3 osztályba sorolhatjuk. Léteznek Imperatív nyelvek legfőbb jellemzőjük, hogy algoritmikus nyelvek, -algoritmust kódolunk és az működteti a processzort- utasítások sorozatát hajtják végre, változókat használnak. Alcsoportjai az eljárásorientált nyelvek és az objektumorientált nyelvek. Léteznek továbbá delklaratív nyelvek, melyek nem algoritmikusak itt a programozó a problémát adja meg és a programba be van építve annak a megoldása. Nyincs lehetőség magasszintű memóriakezelésre. Alcsoportjai a Funkcionális nyelvek és a Logikai nyelvek.

10.2. Adattípusok

Az adattípusokat 3 dolog határozza meg a **tartomány** -az értékeke amiket az adott típus fölvehet-, a vele végezhető **műveletek** -a tartomány elemein végezhető műveletek listája- és a **reporezentáció** -ezeket, hogyan tároljuk-

Minden programozási nyelvben vannak alap típusok, de eggyes nyelvek lehetővé teszik a saját típusok osztályok készítését is. Ezeknek a programozó állítja be a taroományát a műveleteit és a reprezentációját.

Léteznek egyszerű -például az int- és összetett adattípusok iylen például a tömb amely egy statikus és homogén típus, lehet akár többdimenziós is egy tömb.

Mutató típusnak a lényege, hogy ez egy tárterületre hivatkozik a memóriában,térbeli cím. A legfontosabb művelete az általa megcímzett tartomány elérése.

Nevesített konstansnak alapvetően 3 komponense van: név,típus,érték. Ez az érték előre lefixált pl C-ben a #Define "neve" "értéke" ezen módosítani nem lehet ha valahol szükség van rá a nevével lehet rá hivatkozni és fordításnál aneve helyére az érték kerül be.

A **válltozók** a legfontosabb programozási eszközök van nevük -létrehozásuk után ezzel lehet hivatkozni rájik a programmban- , attribútumuk -int,char,dubble-, címük -memóriának arra a részére hivatkozik ahol a változó van- és értékük. A változóknak többféle képpen oszthatunk ki teret a memóriában **statikusan** -futás előtt eldől mi hova kerül ami aztán ott is marad amég fut a program-, **dinamikusan** -A címek kiosztását az OS végzi -A válltozó akkor kap címterületet amikor aktív utána eltűnik.- és a **programozó is kioszthatja** -itt a válltozókhoz a program rendeli hozzá a címet a futási időben. Lehet abszolút cím ekkor fix helyet biztosítunk neki vagy akár lehet relatív cím amikor egy korábbi elem helyzetéhez képest helyezzük el a vílltozót-. Mindhárom esetben tudnunk kell ezeket törölni és nem szabad túlhivatkozni tehát több válltozó nem kaphatja ugyan azt a címterületet.

A C nelv adattípusai

Aritmetikai típusok ezen beül egészek(int,longint,shortint) char lebegőpontos(float, double, long double) származtatott típusok : tömb, függvény, mutató, struktúra. És ezekre csomó példát hoz a könyv.

10.3. Kifejezések

Részei: Operandusok melyek az értékért felelnek, operátorok ezek a műveleti jelek valamint a sorrendet befolyásoló zárójekel: (). Az operátor lehet prefix, infix, vagy postfix. (operátor előtt között vagy mögött). Egy folyamat kiértékelése során sorrendben végezzük a műveleteket. A sorrend lehet balról-jobbra, jobbról-balra illetve Balról-jobbra a precedencia táblázat figyelembevételével. A műveletek elvégzése előtt meg kell határozni az operandusok értékét. A meghatározás sorrendje nyelvfüggé C-ben tetszőleges (Implementációfüggő). Kiértékelésnél fontosak a speciális kifejezések ilyen sestekben pl. ÉS Vagy sokszor nem is kell végigmenni az egészen az érték már azelőtt eldőlt.

Kifejezések a C-ben.

A C egy kifejezésorientált nyelv. A mutatók előjel nélküli egésznek tekinthetők valamitn a tömb neve mutató típusú.

10.4. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Egy új programozási nyelv elsajátiításához a kezdő lépés mindíg a hello world. A program elejére mindíg be kell írni a függvénykönyvtárakat

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

ez pl kell a kiiratásokhoz és a beolvasásokhoz. Megismertük a main függvényt ami mindíg le fog futni na matter what. Azt, hogy a program egyes részeit { } zárójelekkel kell elválasztani és, hogy a pfrintf miként is működik.

ciklusok válltozók

Az 1.2 ben megismerkedhetünk a válltozókkal. Egy programba többféle változót is elhelyezhetünk. Változótípusaink a c nyelvben az int,char,double,long ezek sorban egésszámot, karaktereket, tizedestörteketés

hosszúegészeket tárolnak. Egy ciklussal is megismerkedhetünk ami a while ciklus alábbi módon lehet paraméterezni while (i < j) lényeg, hogy i-től megy j-ig közte lehet <, >, =, <=, >=, <! ,>! a felkiáltójel a nem egyenlő.

Több ciklust is használhatunk a ciklusok közé tartozik a for és a do while ciklus is for ciklushnaá megadjatjuk a fejében, hogy mennyitől meddig és mit csináljon ;-kel elválasztva.

Lehetőségünk van előre megírni helyettesítő szövegeket a programba #define magdineni öreg ez minden helyen ahol a programunkba szerepel a magdineni szó ki fogja cserélni öregre ugyan ez működik számokkal is.Ez segít elkerülni a mágikus konstans helyzetet nem kell eggyesével minden sorba ahol használtuk a válltozót átirogatni.

A könyv egy jó módszert mutat C-ben egy karakter beolvasásához és kiírásához. *válltozó* = getchar() melyet aztán a putchar(*válltozó*) irat ki a stnadard kimenetre ami álltalában a terminál.

Megismerkedünk a tömbbel. Egy tömbben rengetek adatot el tudunk tárolni így például nek kell a programunkban minden egyes válltozóna külön int vagy char területet nyitni hanem az azonos típusú válltozókat tárolhatjuk egy tömbben is pl int cipomeret[3] {45,46,32};

A programokban az ismétlődő kódcsipeteken nem érdemes mindíg újra leírni újrahasznisítás javallott. Ezeket nevezzük fügvényeknek amiket majd kedvünk szerint hivogatunk példa kedvéjért írjunk egy maximum függvényt.

```
int max (int a, int b){
    if (a>b)
        return a;
    else
        return b;
}
```

Több dolog is tisztázásra vár. Mi is az az int az elején? Nos az a visszatérési érték ha meghívjuk akkor a max(3,12) helyett ő a nagyobbat fogja beírni jelen esetben az a sor 12-re fog válltani. Ugyan ezen az elven visszaadhat semmit: void törted: double vagy karaktert char.

10.5. Típusok, operátorok és kifejezések

[KERNIGHANRITCHIE]

Változók

Változónevek akkor jók ha tövidek és jól reprezentálják a változó nevét a programban. Vannak kulcsszavak amiket nem használhatunk ezeket a C nyelv fenntartja magának ezek az inf, float, else, if stb.

A C nyelvben néhány alapvető adattípus van ezek az int float double és char ezeknek a long illetve short valamint unsigned verzióik. Használat előtt minden válltozót Deklarálni kell pl "típus" "név" = "érték" és ;-vel le kell zárni. Ha az int-be törtszám kerül akkor azt a program levágja érdemes ilyen sestben float vagy double használata.

Külömböző aritmetikai poerátpraink vannak a C nyelvben

```
+:összeadás
-:kivonás
*:szorzás
```

```
/:osztás
%:modulo azaz a két szám osztásakor keletkező maradék.
```

A műveletek kiértékelési sorrendjét ()-el lehet befolyásolni.

Logikai operátoraink is vannak iylen a

```
>=:nagyobb egyenlő
<=:kissebb egyenlp
==:egyenlő?
!=:nem egyenlő
&&:és
||:vagy
!"valami":negácó értékmegfordítás
```

Ahoz, hogy műveleteket tudjunk végezni külömböző típusokkal ahoz előbb közös alapra kell őket hozni ezt a program megteszi az int és a float között az intet automatikusan float-ra válltja illetva ha a charban szám van akkor azt intként is kezelheti. De pl a float indexként való használata nem lehetséges.

A C nyelvben van 2 gyakran használt operátor a ++"válltozó" és ugyan ez -- alezek 1et adnak hozzá és 1et vonnak le. Ezek prefixként és postfixként is használlhatók és a végeredmény ugyan az lesz. Ezeket ciklusoknál használjuk leggyakrabban.

A C nyelv rendelkezik bitmanipulációs operátorokkal. Ezek a következők:

```
& bitenkénti ÉS
| bitenkénti megengedő VAGY
^ bitenkénti kizáró VAGY
<< BitShift balra
>> BitShift jobbra
~ egyes komplemens
```

Értékadó operátorok is léteznek legegyszerűbb a =(legyen eggyenlő) vannak a programozó dolgát könnyítők is ilyen a i += 2 amit azt jelenti, hogy az i legyen eggyenlő i+2-vel magyarul adj hozzá 2-t ez érvényes szorzásra is. Ezek a kifejezések kompaktá és könnyen olvashatóvá teszik a kódot.

10.6. Vezérlési szerkezetek

Minden utasítás végén ; kell tenni ez az utasításlezáró jel C-ben. Az összefüggő utasításokat {}-ba kell tenni. Lehetőség van elágaztatásra az if(){} else{} használatával. Többirányú verzió a switch utasítás ezen belül eseteket case-eket kell megadni. Vannak ciklusaink, az ismétlődő műveletek végrehajtására while ciklus és a for ciklus a leggyakoribb. Ezek elöltesztelőek tehát előbb kerül végrehajtásra a kritérium kiértékelése mint az alatta lévő kód ezér lehet 1x se fut le. ellentétben a do while ciklus ami 1x biztos lefut mert hátultesztelő. Végtelen ciklusból a break utasítással lehet kiugrani

10.7. Programozás

Több válltozás is van a C++ és a C nyelv között. Pl kapunk egy boolean tipusú igaz hamis válltozót bool néven ami true vagy false lehet. A c++ már képes szöveget válltozóként tárolni nem mint a C ahol ezt egy

karaktereknek a tömbjével kellett megtenni. itt string szöveg = "Pistak kalácsot evett". A C++ nyelveben bárhol lehet válltozótdeklarálni azol utasítás is állhat. C nyelvben egy függvényt a nevével azonosítunk ami azt jelenti, hogy nem lehet 2 azonos nevű függvény. C++ egy függvényt a neve és az argumentumlistája alapján különíti el ezért akár lehet 2 ugyanolyan nevű függvény is amelynek más az argumentumlistája. Erre egy példa:

```
void macskajaj (int karom, string tappancs)
{
    .....
}

void macskajaj ()
{
    .....
}
```

Fontos megjegyzés, hogy a viszsatérési érték nem jelent külömbésget tehát ha az egyik void a másik int de ugyan az a neve és az argumentumai akkor az hibát eredményez.

10.8. Objektumorientáltság alapelvei.

A C++ is egy objektumorientált nyelv objektumokat készíthetünk specializálhatunk. Az utasítások egységbe zárása hasonlóan történik mint a C- ben tehát {}-el és minden utasítás végén 1 ; áll. Objektumok létrehozásakor használhatunk konstruktort ez az inicializálásnál fontos. Az objektumaink által lefoglalt terület felszabadításért a destruktorok felelnek melyke ~ jellel kezdődnek.

10.9. Objektumorientáltság alapelvei.

Itt is van lehetőség nevesített konstans deklarálására. Mutatóva jelezhetjük, hogy az érték vagy maga a mutatót nem lehet meválltoztatni. Osztályok tagválltozói is lehetnek a konstansok. Létezik konstansról nem konstansra autómatikus konverzió azomban visszafelé nem létezik. Érdemes a hosszú függvényeket elválasztani a main ből. Így sokszor lehet meghívni és egyszerűbb módosítani. Az inline függvény deklarációjában nem szükséges az inline szó használata és nem is szokták használni. Az inline függvények többszörös deklarációja nem okoz linkelési hibát. A fordító majd eldönti, hogy inline alkalmazza e a függvényt vagy a megszokott módon.

III. rész

Második felvonás

Bátf41 Haxor Stream
A feladatokkal kapcsolatos élő adásokat sugároz a https://www.twitch.tv/nbatfai csatorna, melynek permanens archívuma a https://www.youtube.com/c/nbatfai csatornán található.

11. fejezet

Helló, Arroway!

11.1. OO szemlélet

```
public class polargen {

    boolean nincsTarolt = true;
    double tarolt;

    public polargen() {
        nincsTarolt = true;
    }

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

```
        polargen g = new polargen();
        for (int i = 0; i < 10; ++i){
            System.out.println(g.kovetkezo());
        }
    }
}
```

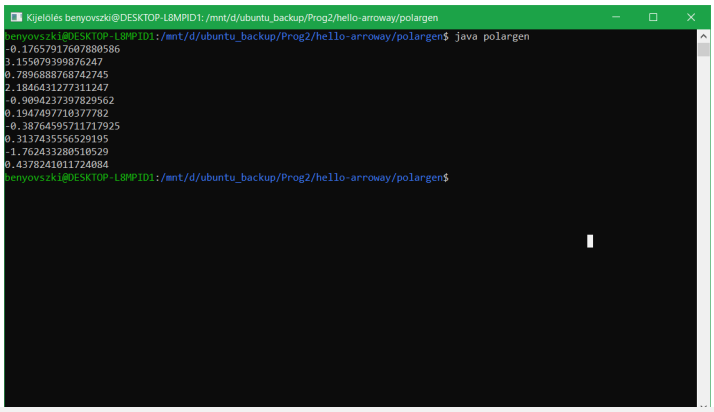
Még mielőtt összevetnénk a mi kódunkat a Java sdk-ban találhatóval azelőtt ejtsünk egy-két szót magáról a kódról és az objektum orientált programozásról.

Az objektum orientáltásg azért fontos mert így sokkal bonyolultabb problémákat le tud ítni a programozó sokkal egyszerűbben, olvashatóbban. 3 alapelve van : egységbe zárás, adatrajtés és öröklés.

Ebben a programkódban az egységbe zárás elvét tudjuk szemléltetni, hiszen az osztályunk public és semmelyik sem örököl. Tehát itt most minden változót és függvényt a polargen classunk tartalmazza.

A kódunk pseudo random számokat generál és írja ki a standard outra. A számolást azt a kovetkezo() függvény végzi, amit meg is hívunk a példányra a maintben. A függvény kiszámok 2 random számot az eggyiket eltárolja, feljegyzi, hogy van egy tárolt értékünk és a másik számot pedig visszatéríti ami ki is ír a kimenetre. A függvény újbóli meghívása esetén pedig feljegyezzük, hogy már nincs tárolt érték valamint az előzőelg kiszámolt értéket visszaadja ami szintén kiírásra kerül. Ez azért jó mert így a processzorunk kevesebbszer számolja végig a képletet.

A porgram futás közben.




```
    synchronized public double nextGaussian() {
        // See Knuth, ACP, Section 3.4.1 Algorithm C.
        if (haveNextNextGaussian) {
            haveNextNextGaussian = false;
            return nextNextGaussian;
        } else {
            double v1, v2, s;
            do {
                v1 = 2 * nextDouble() - 1; // between -1 and 1
                v2 = 2 * nextDouble() - 1; // between -1 and 1
                s = v1 * v1 + v2 * v2;
            } while (s >= 1 || s == 0);
            double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
            nextNextGaussian = v2 * multiplier;
            haveNextNextGaussian = true;
```

```
            return v1 * multiplier;
        }
    }
```

Most nézzük meg, hogy mi a külömbség a mi kódunk és az SDK-ban található között. Az 1. szembetűnő külömbség a class nevében van hiszen itt synchronized public double-t használ. Ez azért jó mert ha többszálas munkakörnyezetben előfordulhat, hogy több szál is megpróbál ugyan arra az adatra ráfrissíteni ezt védhetjük ki a synchronized kulcsszó használatával ezzel a kulcsszóval megjelölt függvényben egyszerre csak egy szál fut. A másik lényeges külömbség a strictmath és a math között található. A kettő között az a külömbség, hogy a strictmath megköveteli, hogy minden sestben ugyan az legyen a végeredmény pl. más rendszeren lefutattva is. A sima math függvénynél hasonlót de nem feltétlenül mindíg ugyan azt téríti vissza és a sima math enged platofrmspecifikus implementációkat mint pl. a x86 floating point használata.

11.2. Homokozó

Tanulságok, tapasztalatok, magyarázat...

11.3. „Gagyi”

```
class Gagyi
  {

    public static void main(String[] args)
      {
        Integer x = -127;
        Integer t = -127;
        System.out.println(x);
        System.out.println(t);
        while (x <= t && x >= t && t != x);
      }
  }
```

Ebben a példában a java sdk-ban található Integer Wrapper class-t fogjuk használni. Ez több mozgásteret enged a sima inthez képest ami csak a számok bineáris értékét tárolja. Integer-re osztály révén hívhatunk meg függvényeket pl. megfordíthatjuk a reverse() fügvénnyel. Nézzük meg, hogy a fenti Integer deklarálásunkat a fordító, hogyan olvassa. Az integer x = -300-at a következőképpen: Integer x = Integer.valueOf(-300). Alapértelmezetten a javában 127 és -127 között gyorsítótárazva vannak a számok hiszen ezeket gyakran használják, ebből kövezkezik ha a megadott értékekk között hozunk létre változókat azoknak az értéke ugyanarra a gyorsítótárazott objektumra fog mutatni ezért az összehasonlításoknál a várt eredményt fogjuk megkapni. Azomban ha ezen értékek fölé vagy alá megyünk akkor már új objektumot hoz létre a java, aminek a memóriacíme is más lesz. Ez eredményezi azt, hogy ha a fent említett értékekk közül veszünk 2 ugyan olyan számot akkor nem lépünk be a ciklusba viszont ha túlmegyünk az értékhatáron akkor már belépünk mert nem az Integer értékét hasonlítja össze a Java. Erre természeresen erre a problémára van megoldás az .intValue() függvénnyel ki lehet küszöbölni és akkor már az értékek lesznek összehasonlítva.

Az Integer gyorsítótára így néz ki.

```
private static class IntegerCache {
        static final int low = -128;
        static final int high;
        static final Integer cache[];

        static {
            // high value may be configured by property
            int h = 127;
            String integerCacheHighPropValue =
                sun.misc.VM.getSavedProperty("java.lang.Integer. ↩
                   IntegerCache.high");
            if (integerCacheHighPropValue != null) {
                try {
                    int i = parseInt(integerCacheHighPropValue);
                    i = Math.max(i, 127);
                    // Maximum array size is Integer.MAX_VALUE
                    h = Math.min(i, Integer.MAX_VALUE - (-low) -1);
                } catch( NumberFormatException nfe) {
                    // If the property cannot be parsed into an int, ignore ↩
                        it.
                }
            }
            high = h;

            cache = new Integer[(high - low) + 1];
            int j = low;
            for(int k = 0; k < cache.length; k++)
                cache[k] = new Integer(j++);

            // range [-128, 127] must be interned (JLS7 5.1.7)
            assert IntegerCache.high >= 127;
        }

        private IntegerCache() {}
    }
```

11.4. Yoda

Először is nézzük meg, hogy mi fán terem a Yoda conditions. A lényeg, hogy a kifejezések a megszokotnál fordított sorrendben kerülnek leírásra tehát pl a változó a jobb oldalon és a konstans a bal oldalon. Ez az elnevezés a StarWars filmből ered hiszen itt az angol verzióban Yoda fordítva beszél. Ezze nézzünk is egy példát.

```
          int szam = 2;
          if (3 > szam);
```

így néz ki a Yoda condition a gyakorlatban. A Yoda condition használtaával ki lehet védeni tipikus porgramozási hibákat mint pl. amikor véletlenül az egyenlő-e? operátpr helyett a legyen egyenlő! operátort használjuk a feltételekben

```
int szam = 2;
if (3 = szam); //Ez fordítási hibát eredményez mert nem lehet a 3 ↩
    számot egyenlővé tenni semmivel.
```

Ezek után nézzük meg, hogy melyik kód is lép ki NullPointerException-el ha nem követjük Yoda stílusát.

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

Ez a kód fordítási hibához fog vezetni hiszen ez hasonló a matamatikában a nullával való osztáshoz. Viszont Yoda-ban ez a kód lefordul és le is fut és a végeredmény várt hamis lesz.

Sokan azt tanácsolják, hogy ha lehet ne használjuk ezt hiszen a rosszul == helyetti = jelre a legtöbb fordító felhívja a figyelmet. Yoda conditions-t hasznélunk akkor könnyen előfordulhat, hogy rosz helyre bekerül egy nullpointer ami később nem várt viselkedéshez vezethet.

11.5. Kódolás from scratch

A programunknak az a lényege, hogy a példányosítástól kapott érték + 1 től kezdi el számolni a PI tizedesjegyeit úgy, hogy a programnak semmit tudomása nincs arról, hogy mitk az előtte lévő tizedesjegyek. Az alábbi kódban pl a kiszámolt 1. 6 jegy lesz pontos.

```
    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();
    }
```

A PiBBP függvényben számoljuk ki magát a képletet. A StrictMath.floor() floor függvény a kapott számtól lefelé visszaadja a számhoz legközelebb álló egészet pl. a 2.9-ből pl 2-t ad vissza egész számok esetén nem történik semmi. Csinálunk egy stringbuffert amibe majd szépen lassan fogjuk beletölteni a kiszámolt jegyeinket. Létrehozunk egy tömböt ami majd a hexa karaktereink lesznek. A while ciklusban pedig megtörténik maga az az átváltás és a hozzáfűzás ha az adott szám kissebb mint 10 akkor egyszerűen hozzáadjuk a bufferhez ha pedig nagyobb akkor meg kivonunk belőle 10-et és a tömbben a megfelefő helyen álló betűt fűzzük hozzá a bufferhez a .append() főggvénnyel. Végén pedig áttöltjük a buffert magába a string változóba.

```
    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;
            }
```

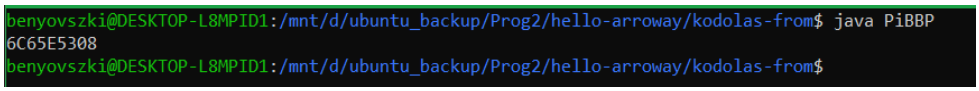
```
            t = t/2;

            if(t < 1)
                break;

            r = (r*r) % k;

        }

        return r;
    }
```

A public double d16Sj() függvény végzi a könyv 4. oldalán található S1 S2 S3 S4 számok kiszámtását amihez segítségül fogja hjvni az alatta található függvényt is ami bineáris hatványozást végez.

```
    public String toString() {

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

Az utolsó függvény visszaadja majd magát a kiiratni kívánt PI hexa karaktereket a bufferből. A mainben egy kiiratásba ágyazott példányosítás történik és mivel 1m-t írtunk be ezért a kiiratás 1m+1-től fog majd indulni.

A program futás közben



12. fejezet

Helló, Arroway!

12.1. Liskov helyettesítés sértése

Ebben a feladatban a madarak segítségével meg fogjuk sérteni a Liskov elvet. De a kódunk le fordul és le is fut akkor tulajdonkppen itt mi a hiba? Az, hogy amint látjuk a forráskódból a Pingvin osztály örökli a repülést is hiszen azt beleépítettük a madár osztályba és így a programunk röptetni fogja a madarat ami rossz hiszen a valóságban a Pingvin egy röpképtelen madár. És mi nem várnánk el a Pingvintől ,hogy erre képes legyen. Az ilyen hibák hibás működéshez szoktak vezetni amiket az okát nehéz megtalálni. Ezt legegyszerűbben úgy tudjuk megoldani, hogy elkülönítjök a repülést a Madár osztálytól. Lesznek a sima madaraink és a repülő madaraink és így elkülönül a repülés és így az fgv-re már nem lehet meghívni a reptető függvényt.

A C++ Változat

```
#include <iostream>

using namespace std;

class Madar {
public:
  virtual void repul() { cout << "Ezaz repulok!" << endl; };
};

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

class Sas : public Madar
{};

class Pingvin : public Madar
{};
```

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

     Sas sas;
     program.fgv ( sas );

     Pingvin pingvin;
     program.fgv ( pingvin );

}
```

Az áttervezett verzió.

```


class Madar {

};

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

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

class Sas : public RepuloMadar
{};

class Pingvin : public Madar
{};

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

     Sas sas;
     program.fgv ( sas );

     Pingvin pingvin;
```

```
     program.fgv ( pingvin );

}
```

A Javás megolás ami sért.

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

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

class Sas extends Madar
{};

class Pingvin extends Madar
{};

class lsiskovsert{

public static void main ( String[] args )
{
     Program liskov = new Program();
     Madar madarr = new Madar();
     liskov.fgv ( madarr );

     Sas sass = new Sas();
     liskov.fgv ( sass );

     Pingvin pingvinn = new Pingvin();
     liskov.fgv ( pingvinn );

}}
```

12.2. Szülő-gyerek

Ebben a feladatban az öröklés folyamatát mégpedig egy igen egyszerű java és c++ kóddal. A C++ kóddal kezdjük.

```
#include <iostream>

using namespace std;
```

```
class Szulo
{
public:
  void szulo_vagyok() {
    cout << "Szia én vagyok a szulő" << endl;
  }

};

class Gyerek : public Szulo {

public:
  void gyerek_vagyok() { cout << "Szia én vagyok a gyerek" << endl; }

};

int main() {

  Szulo apa;
  apa.szulo_vagyok();
  // apa.gyerek_vagyok(); Ha ezt a programsor nem lennek kommentelve akkor ↩
     nem fordulna le a program.

  Gyerek pista;
  pista.szulo_vagyok();
  pista.gyerek_vagyok();
}
```

Akkor szokutunk származtatott osztályokat használni ha van egy tágabb csoportunk ezek legyenek modjuk az állatok ezen belül lehetnek modjuk az emlősök. Az ős osztály szokta tartalamzni az általánosabb leírást és a gyereke a specifikusabb dolgokat. A C++-ban : utáni osztály a szülő osztály angolul base class valamint az előtte lévő a származtatott osztály a gyerek angolul a Derived class. Hozzunk létre egy osztályt stílusosan szülő néven amiben létrehoztunk egy public függvényt ami ki fog írni a kimenetre. Ezek után készítünk egy Gyerek osztályt ami örökölni fogja a szülő funkcióit. : operátorral történik az örökítés itt most public öröklődést használtunk, de lehet protected valamint private öröklődés is. A main függvényben pedig plédányosítunk és meghívjuk az egyedekre az függvényeket. Láthatjuk, hogy a gyerek képes használni a saját függvényeit valamint az őséjét is. Viszont az ős nem tudja használni a gyerekét ami logikus hiszen a valóságban sem tudják használni az emlősök az emer tulajdonságait viszont fordítva igen.

```
benyovszki@DESKTOP-L8MPID1:/mnt/d/ubuntu_backup/Prog2/hello-liskov/szulo-gyerek$ ./szulo
A szulore meghivva
Szia en vagyok a szulo

A szulore meghivva
Szia en vagyok a szulo
Szia en vagyok a gyerek
benyovszki@DESKTOP-L8MPID1:/mnt/d/ubuntu_backup/Prog2/hello-liskov/szulo-gyerek$
```

Viszont ha a fent említett kommentes részt is beel akarnánk venni azt tapasztaljuk, hogy a program nem fordul le.

```
benyovszki@DESKTOP-L8MPID1:/mnt/d/!Prog2/Prog2/hello-liskov/szulo-gyerek$ g++ szulo-gyerek.cpp -o szulo
szulo-gyerek.cpp: In function ‘int main()’:
szulo-gyerek.cpp:27:6: error: ‘class Szulo’ has no member named ‘gyerek_vagyok’; did you mean ‘szulo_vagyok’?
  apa.gyerek_vagyok(); // Ha ezt a programsort kikommentelnnk akkor a program nem fordulna.
      ^~~~~~~~~~~~~~
      szulo_vagyok
```

Most nézzük meg ezt a példát javába átültetve. Javában öröklődést az extends kulcsszóval lehet elérni. Itt is ugyan úgy viselkednek a függvényhívások tehát az ős csak a saját függvényeit tudja kezelni viszont a gyerek képes mindkettő kezelésére.

```
class Szulo {
    public void szulo_vagyok() {
    System.out.println("En vagyok a szulo hehehe");
  }
}

class Gyerek extends Szulo {
  public void gyerek_vagyok(){System.out.println("En vagyok a gyerek hehehe ↩
     ");}
  public static void main(String[] args) {

    Szulo apa = new Szulo();
    Gyerek pista = new Gyerek();

    apa.szulo_vagyok();
    // apa.gyerek_vagyok(); // Itt is ugyanúgy reagál rá a fordító.
    pista.szulo_vagyok();
    pista.gyerek_vagyok();

  }
}
```

12.3. Anti OO

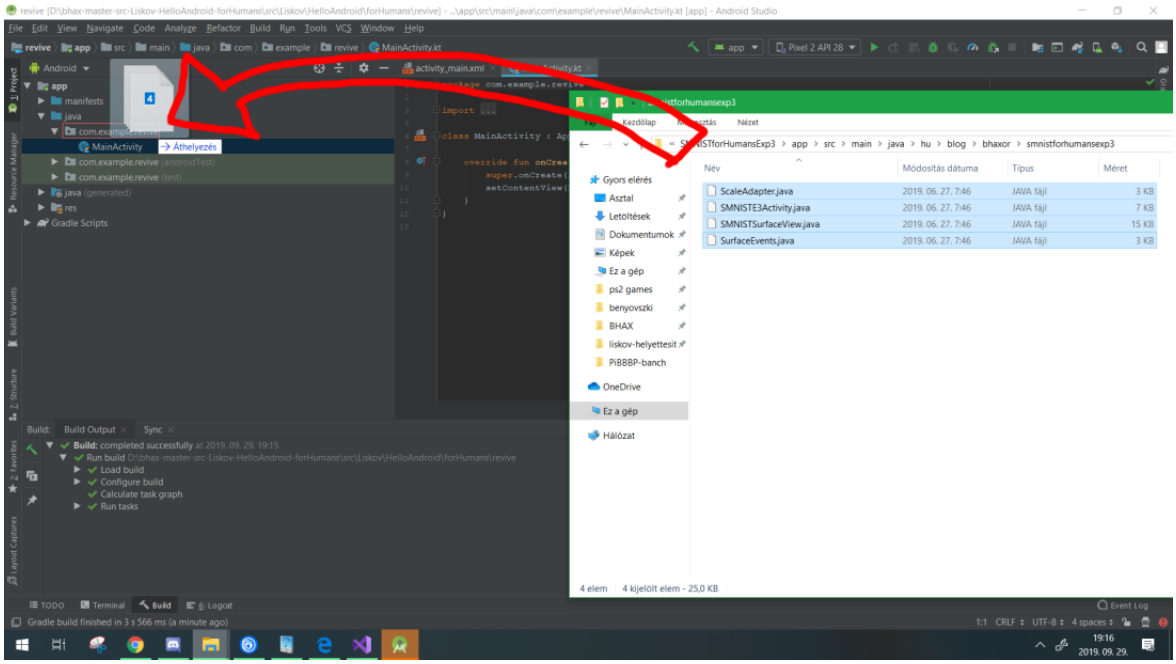
Egy táblázatban hasonlítjuk össze a futási időket.

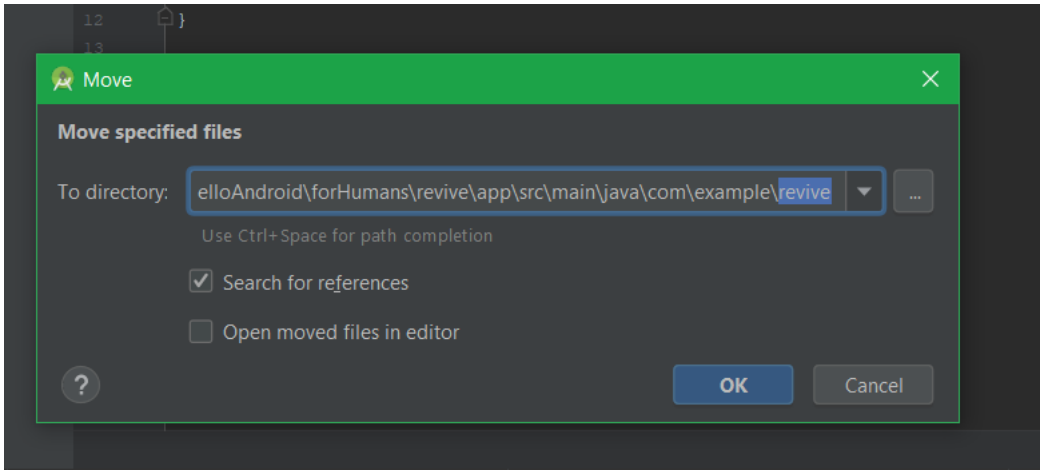
A táblázatból kiolvasható, hogy a legjobban a java nyelv teljesített ez valószínüleg az alapból bekapcsolt gyorsításoknak köszönhető amiket a nyelv tartalmaz. Ezért gyorsabban fut le mint az optimalizálatlan C valamint C# kód.

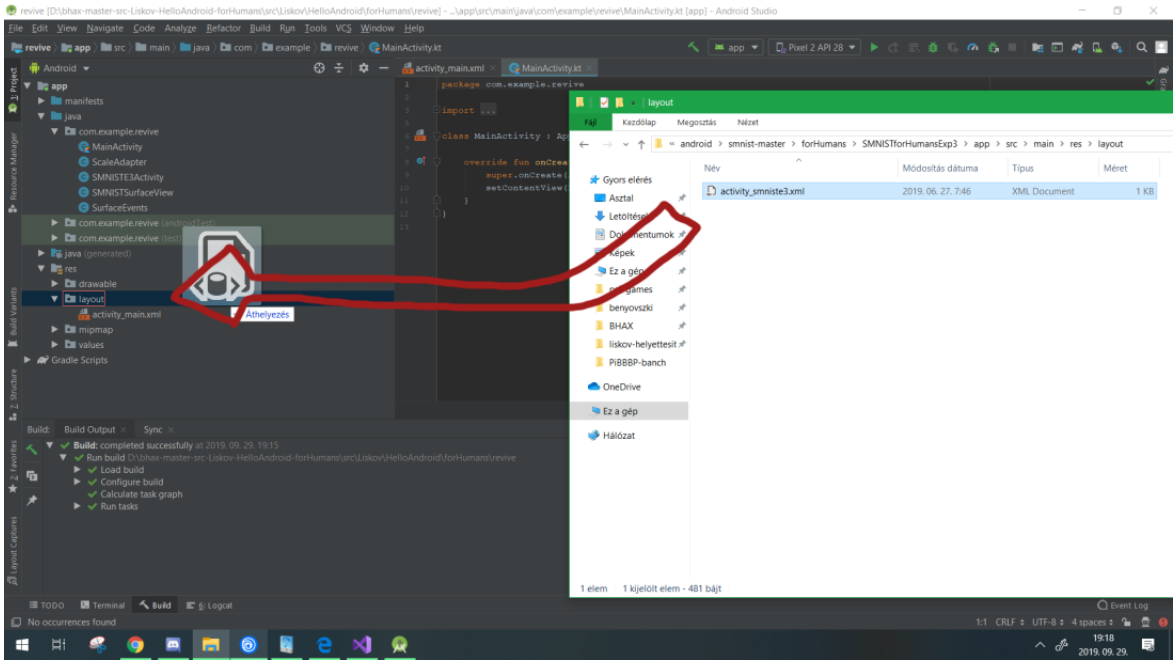
	C	C++	C#	JAVA
10^6	1.796875		2,056352	1.64
10^7	21.062500		23,05691	19.501
10^8	248.109375		263,0927	225.36

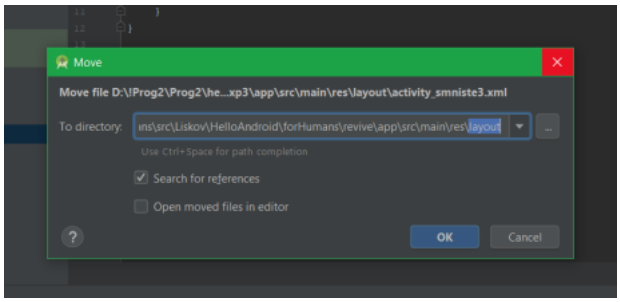
12.4. Hello, Android!

Ebben a feladatban az alkalmazás felélesztéséhez android stúdiót fogunk használni. Az alap project beimportálása és futtatása nem volt lehetséges ezért egy teljesen új projectbe fogjuk belerakni a fájlokat méghozzá egy új empty activity-be. A fájlok importálásának folyamatát képekben mutatom meg.

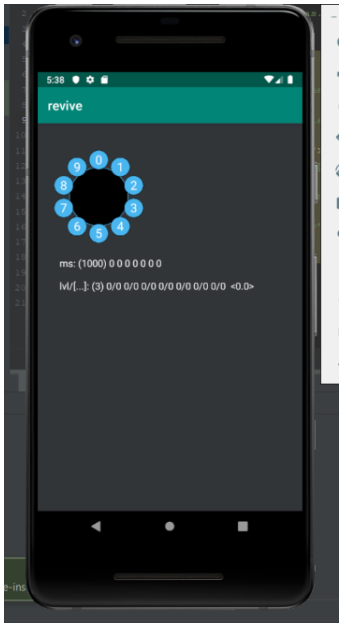








A már ott lévő alapból legenerált fájlokat törölhetjük. Miután ezzel elkészültünk még apró módosításokra van szükség az activity_smniste3.xml-en valamint az AndroidManifest.xml-en. Utóbbinak a 12. sorát át kell írni a MainActivity-t SMNISTE3Activity-re. Az activity_smniste3.xml ben pedig a 7. sort kell módosítani, hogy átlinkeljük a régi fájlokat az új projecthez. És azt játjuk, hogy már működik is és a kis virtuális telefonunkon működik is a program.



Ezek után már csak kissebb módosításokat kell elvégezni a programban legyenek mondjuk ezek a feladat leírásában szereplő színek. Ezek az SMNISTSurfaceView-ban találhatóak.

```
int[] bgColor =
        {
                Color.rgb(49, 54, 59),
                Color.rgb(35, 38, 41)
        };
```

A 74-78. sorokba található kódrészlet a váltakozó háttérszínért felelős itt a színek rgbkódját megadva változtathatunk. A többi színt a 358-370. sorban találjuk. Itt át kell írnunk a pontok után szereplő színneveket másra. És ez lett a végeredmény. A módosítható értékek nevét kiírtam a képre.

12.5. Ciklomatikus komplexitás

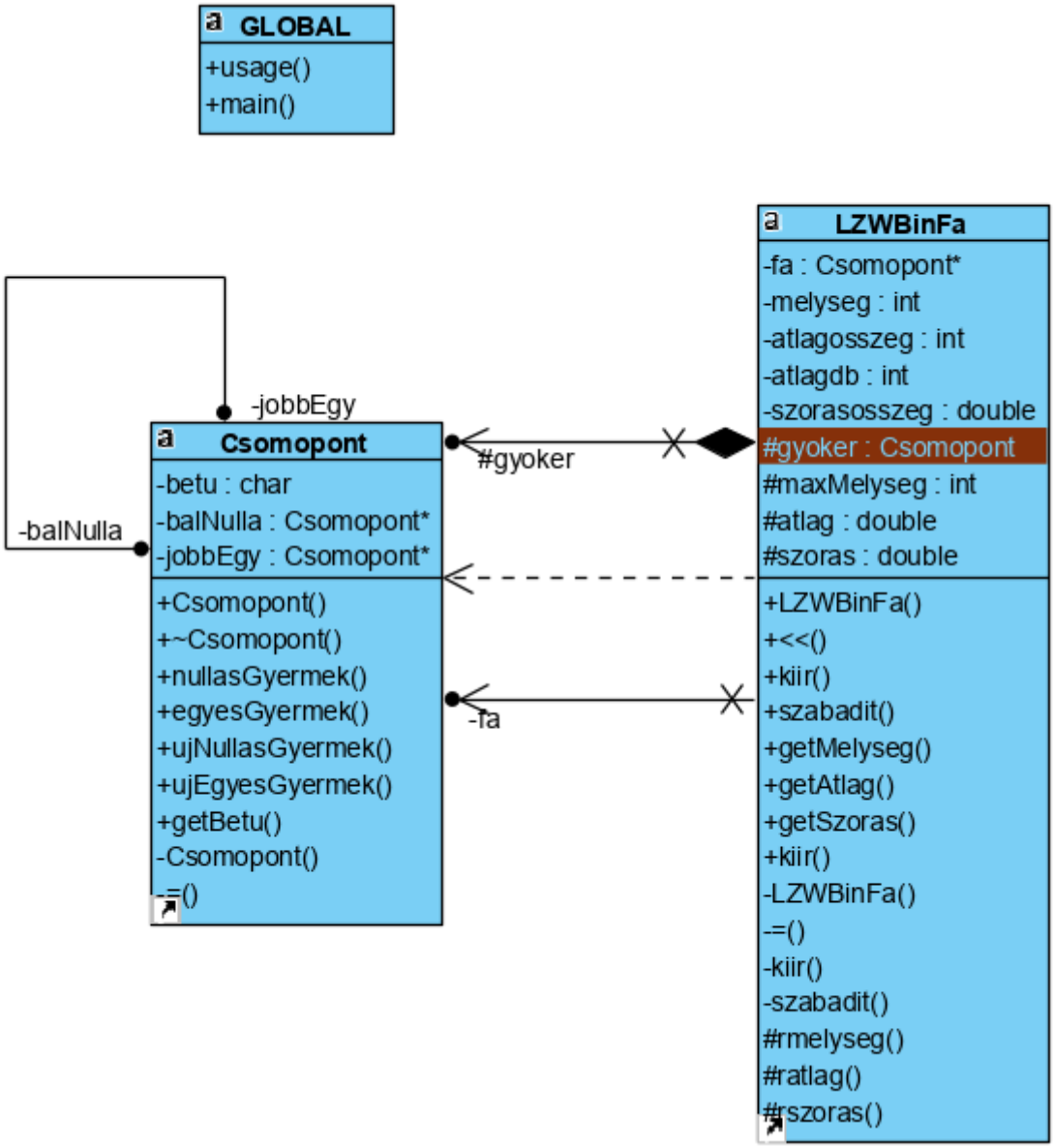
A Ciklomatikus komplexitás azt adjam eg a kódunkról, hogy mennyire egyszerően, tömören lett megírva minnél alacsonyabb az értéke annál jobb. Egy jó programmodulnál ez az érték kissebb mint 10. Ha ez az érték meglahadja a 10-et akkor el kell gondolkoznunk a kódunk egyszerüsítésén hiszen egy szeretágazó, bonyolult kódot nehéz értelmezni, módosítani valamint karbantartani. Ha ez az érték nagyon magas akkor azon is el kell gondolkodnunk, hogy újratervezzük/újraírjuk ez az egészet. Rövid kódoknál a számolást akár mi is elvégezhetjük, de egy hosszú programnál érdemes erre külön programot használni. Kézi számítás esetén mindíg 1 től indulunk. Minden metódusban található futással kapcsolatos elem. Minden return-nél ami nem a metódus legutolsó része. Elágaztatásoknál, loopoknál operátoroknál és kivételeknél 1-et adunk hozzá. Mi egy online eszkötz fogunk alkalmazni mégpedig a http://www.lizard.ws/#try-t ide illesztjük be a javás binfa forráskódját és a következő eredményt kaptuk.

Function Name	NLOC	Complexity	Token #	Parameter #
LZWBinFa::LZWBinFa	3	1	9	
LZWBinFa::egyBitFeldolg	22	4	104	
LZWBinFa::kiir	4	1	26	
LZWBinFa::kiir	4	1	22	
LZWBinFa::Csomopont::Csomopont	5	1	21	
LZWBinFa::Csomopont::nullasGyermek	3	1	8	
LZWBinFa::Csomopont::egyesGyermek	3	1	8	
LZWBinFa::Csomopont::ujNullasGyermek	3	1	11	
LZWBinFa::Csomopont::ujEgyesGyermek	3	1	11	
LZWBinFa::Csomopont::getBetu	3	1	8	
LZWBinFa::kiir	15	3	107	
LZWBinFa::getMelyseg	5	1	21	
LZWBinFa::getAtlag	6	1	32	
LZWBinFa::getSzoras	12	2	68	
LZWBinFa::rmelyseg	11	3	51	
LZWBinFa::ratlag	12	4	66	
LZWBinFa::rszoras	12	4	78	
LZWBinFa::usage	3	1	14	
LZWBinFa::main	64	14	401	

13. fejezet

Helló, Mandelbrot!

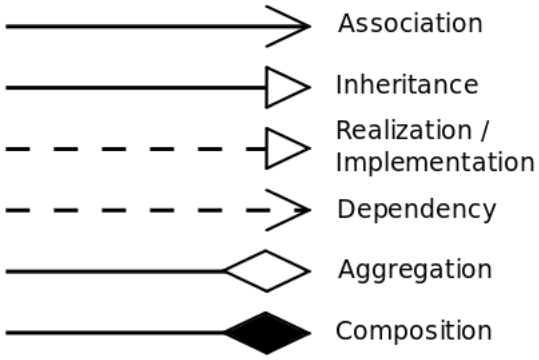
13.1. Reverse engineering UML osztálydiagram



Az osztályok ban lévő tagok láthatóségét a nevük mellett lévő **- # +** jel mutatja meg nekünk melyek sorba a következőket jelentik. Private, protecete valamint public.

A diagrammunkból látszik, hogy a 2 global függvényünk nem függ semmitől, hiszen nem indulnak belőle nyilak és felé sem mennek.

Először vegyük szemügyre a csomópont osztályt melynek a bal oldalán láthatunk egy nyilat ami a csomópontból indul és ugyan oda érkezik vissza. Ez azt jeleni, hogy összeköttetés van sima association ami mivel mindkét végén pont van mind a két oldalról navigable (?bejárható?).

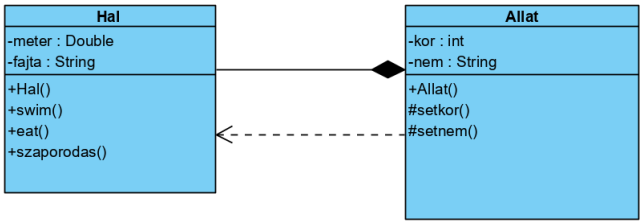


Kompozíciós kapcsolat van a csomópont és az LZW között ez azt jelenti, hogy a csomópont osztály nem létezhet az LZW nélkül. Láthatunk felül egy olyan nyilat is aminek az a neve, hogy Aggregation ebben az esetben viszont létezhetne függetlenül a tárolótól (LZW osztály).

Most az LZW felöli oldalt nézzük meg. Látható egy sima eggyirányú -mert csak az eggyik végén van pont- kapcsolat a #gyükér. Egy függőséget jelző nyil ami azt jelent, hogy a két osztály között szemantikai kapcsolat van ez azt jelenti ebben az esetben, hogy ha a Csomópontban változás történik az kihatással lesz az LZW-re is. Valamint láthatunk még egy eggyirányú kapcsolatot amit a fa tagnak a kapcsolata.

13.2. Forward engineering UML osztálydiagram

Forward engineering-et érdemes alaklmazni amikor a kódot tervezzük. Gondolattérkép szerűen felépítjük a kódot és aztán már csak egy gombnyomásra Visual Paradigm segítségével készíthetünk az osztálydiagrammból ami a képen látható akármilyen nyelven egy kódot.



13.3. Egy esettan

Feladatot tutorálta: Olah Sándor Lajos

A könyv végigvezet minket egy program megírásán. Könyvtárak formájában és az lesz a lényeg, hogy később ne keljen átírni a forráskódot nagyon. A megírt program egy informatika cég termékeit fogja tárolni alkatrészenként (megjelenítők, merevlemezek) és összetett termékekként (számítógép). Az alap osztály amit minden örökölni fog az a Product osztály ez tartalmazza a nevet a termék korát és ki is tudja számolni az árát. Az á függvényt az alosztályokban felüldefiniáljuk a megfelelő termék tulajdonságaira alapozva pl. a merevlemezt leárazzuk 30 majd 90 nap után. A programnak tudnia kell olvasni fájlból és azt eltárolnia adatbázisban. Erre írtunk egy void függvényt a Program osztályba. Soronként fogjuk beolvasni az adatokat és

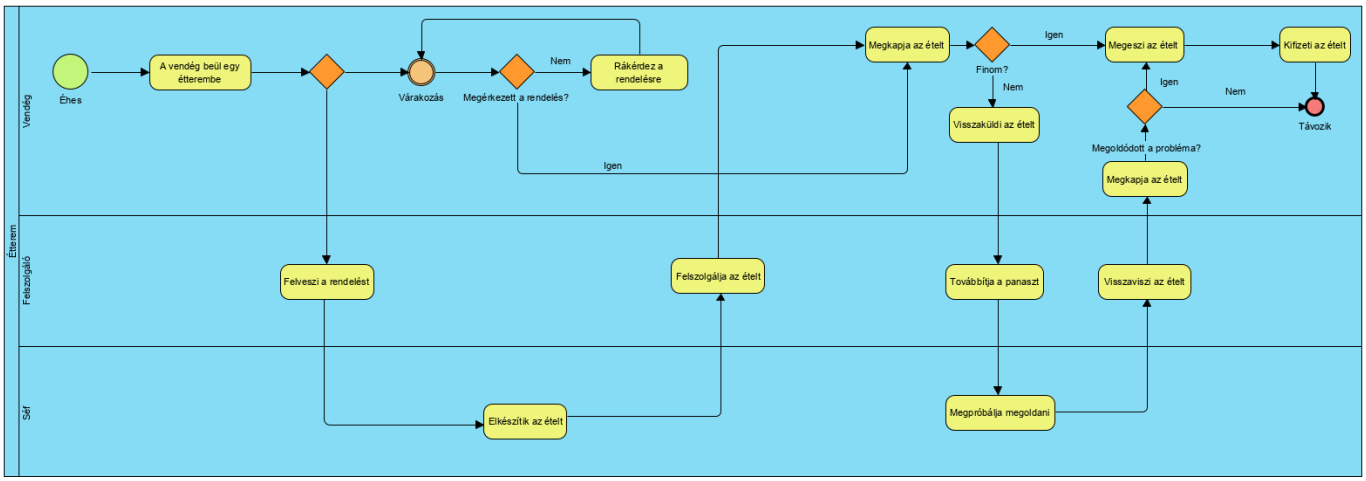
az 1. karakter lesz a termékazonosító. Az áron kívül mindent tud tárolni -az árat a termék korából számoljuk ki- ezért az árat majd a gyerek osztályokban felüldefiniálhatjuk. A nehezebb feladat az összetett termékek tárolása ezt az osztályt is a Product osztályból származtatjuk és azokat az alkatrészeket amikből felépülnek egy külön vektorban tároljuk el és ehez a vektorhoz írnuk függvényeket amik hozzáadnak alkatrészeket. Ha ezzel elkészültünk akkor már csak az adatok kiírásáról kell gondoskodnunk. Ezt a Product osztályban található virtual void Printparams függvény csinálja. Ezt terméknek megfelelően felül lehet definiálni. A termékeket nyilván kell tartanunk ezért be kell vezetnünk egy új osztályt ami ezt végzi. A példányosításhos létrehozunk egy ProductFactory osztályt és a termékek kezelését az ebben található ReadAndCreateProduct függvényre bízzuk.

A program futás közben.

A program forrása.

```
benyovszki@DESKTOP-L8MPID1:/mnt/d/!Prog2/Prog2/hello-mandelbrot/jo-esttan$ ./esettan
Test1: create inventory and printing it to the screen.
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20191007, Age: 0, Current price: 30000, InchWidth: 13, InchHeight: 12
1.: Type: HardDisk, Name: WD, Initial price: 25000, Date of acquisition: 20191007, 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: 6579, Current price: 24000, InchWidth: 12, InchHeight: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 4754, Current price: 28000, InchWidth: 10, InchHeight: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age: 4754, Current price: 70000
Items:
 0. Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20011001, Age: 6579, Current price: 24000, InchWidth: 12, InchHeight: 13
 1. Type: HardDisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 4754, Current price: 28000, SpeedRPM: 7000
3.: Type: HardDisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20050228, Age: 5333, Current price: 20000, SpeedRPM: 7000

The content of the inventory has been written to computerproducts_out.txt

Done.
```

13.4. BPMN



Business Process Model and Notation lényege, hogy érthető folyamatábrát készítsen. Lehetőségünk van a folyamatok elágazásait kitárgyalni benne. Mi most egy éttermi rendelésen a folyamatábráját készítettük el. Leolvashatjuk róla, hogy milyen lépések vannak amíg eljutunk a kezdőállapotból a végállapotig. A zöld kér a kezdőállapot, a lekerekített szélű téglalap a tevékenység a rombusz az elágaztatás a dupla körvonalas kör az a várakoztatás a rózsaszín kör pedig a végállapot.

13.5. TeX UML

A feladat megoldásához MetaUML csomagot választottuk amit innen lehet letölteni https://github.com/ogheorghie
A leírásban található pdf alapján könnyen lehet vele diagrammokat készíteni.

```
#include <boost/filesystem.hpp>
#include <iostream>
#include <vector>
#include <cstdlib>
#include <experimental/filesystem>
using namespace std;

int main(int ac, char** av)
{
    string extension;
    vector<string> files;
    int count = 0;
    boost::filesystem::recursive_directory_iterator iterator (string("/mnt/ ↩
       d/src"));
    while(iterator != boost::filesystem::recursive_directory_iterator())
    {
      string extension = boost::filesystem::extension(iterator->path(). ↩
         filename());

        if (boost::filesystem::is_regular_file(iterator->path()) &&  ↩
           extension == ".java")
        {
          files.push_back(iterator->path().filename().string());
          count++;
        }

      ++iterator;

    }

    int size = files.size();
    cout << size << endl;

ofstream myfile;

    myfile.open ("kimenet.txt");

        for(int i=0 ; i<size ; i++)
        {
            myfile << files[i]<<";"<<endl;
        }

    myfile.close();
```

```
    return 0;

}
```

Egy osztályt így lehet leírni abban a csomagban először a válltozókat majd a metódusokat írjuk le. A láthatósági jeleket egyszerűen a név elé lehet írni. Ha egy egységünk nem rendelkezik metódusokkal vagy változókkal akkor a hiányzó tag(ok) helyére () üres zárójelet kell rakni.

Az így megírt osztályokat már csak csomagokba kell rendeznünk. PL. így **Package.Q("sampleclient")(A, B, C, D, E);** Majd ami a legnagyobb kihívást jelentette meg kell határoznunk az osztálydiagrammon belül a tagok elhelyezkedését. Erre ebben a csomagban 2 lehetőségünk van pl. közvetlenül megadni -ezt a manual relatív elhelyzésnek nevezi-, hogy egymástól milyen irányban helyezkedjenek el. **J.w = H.e + (100, 0);** (a w és az e angol égtájak) ezt jó esetben csak 2 elemre kell alkalmazni utána átálhatunk a felhasználóbarátabb verzióra **topToBottom.midx(50)(L, K, M);** ezzel pl. függőlegesen tudjuk elrendezni középre igazítva elemenként. 50-es térközzzel. Arra is van lehetőség, hogy a láthatóságjelzőket kikapcsoljuk **Class_noVisibilityMarkers.L;**. Végül kirajzoltatjuk az az osztálydiagrammot **drawObjects(S, T, V);**. Most a linkeket kell kialakítanukn -a draw után működik- ez is egyszerűen megy **link(inheritance)(N.e -- K.w);** mek kell adnunk az összeköttetés nevét és azt, hogy mik között jöjjön létre és, hogy melyik oldalakat kösse össze(angol égtájak kezdőbetűje). A végeredményről nem tudtunk megfelelő képernyőmentést készíteni, de itt megtekinthető. És a teljes kód kb. 300 sor azt itt lehet megnézni. A végén **mptopdf *a_bemeneti_fájl_neve*** parancsal lehet belőle pdf-et készíteni.

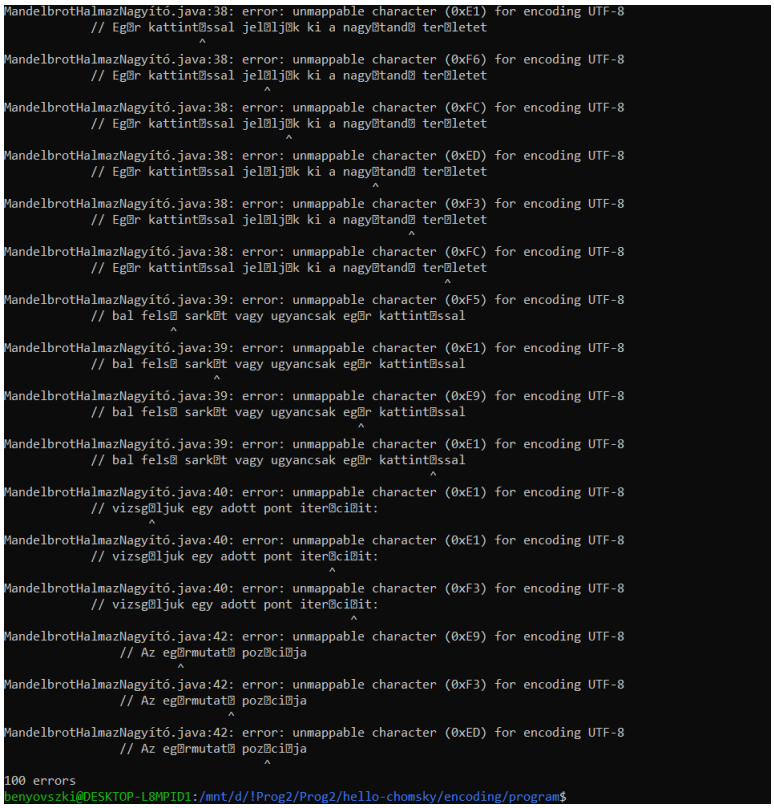
14. fejezet

Helló, Chomsky!

14.1. Encoding

Láthatjuk ha megnyitjuk a kódunkat egy szerkesztővel akkor láthatjuk, hogy alapból nem ismeri fel a magyar ékezetes karaktereket ezért alapból nem is fog lefordulni. Éppen ezért fordítani sem tudjuk hiszen alapból nem fogja felismerni a karaktereket mert UTF-8-ban nincs lekódolva. Erre használjuk a -encoding kapcsolót amivel megadhatjuk a nekünk passzoló karakterkészletet ami jelen esetben az ISO-8859-1 lesz. Tehát ez lesz a helyes fordítási parancs **javac -encoding "ISO-8859-1" MandelbrotIterációk.java MandelbrotHalmazNagyító.java** amivel hiba nélkül le fog fordulni valamint futni.

```
    }
    // Vonszolva kijel�l�nk egy ter�letet...
    // Ha felengedj�k, akkor a kijel�lt ter�let
    // �jrasz�m�t�sa indul:
    public void mouseReleased(java.awt.event.MouseEvent m) {
        if(m.getButton() == java.awt.event.MouseEvent.BUTTON1 ) {
            double dx = (MandelbrotHalmazNagy�t�.this.b
                    - MandelbrotHalmazNagy�t�.this.a)
                    /MandelbrotHalmazNagy�t�.this.sz�less�g;
            double dy = (MandelbrotHalmazNagy�t�.this.d
                    - MandelbrotHalmazNagy�t�.this.c)
                    /MandelbrotHalmazNagy�t�.this.magass�g;
            // Az �j Mandelbrot nagy�t� objektum elk�sz�t�se:
            new MandelbrotHalmazNagy�t�(
                    MandelbrotHalmazNagy�t�.this.a+x*dx,
                    MandelbrotHalmazNagy�t�.this.a+x*dx+mx*dx,
                    MandelbrotHalmazNagy�t�.this.d-y*dy-my*dy,
                    MandelbrotHalmazNagy�t�.this.d-y*dy,
                    600,
                    MandelbrotHalmazNagy�t�.this.iter�ci�sHat�r);
        }
    }
```



14.2. OOCWC lexer

14.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Először is szükségünk lesz a <GL/glut.h> könyvtárra amit ubuntun a következő parancsal szerezhetünk be: **sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev**. A program 1.rendű logikai nyelvben megfogalmazható mondatokat vizualizálja. Az irányítás kezdetben nem volt kézreálló, hiszen az ellentétes irányokba fordult a kocka a gomb lenyomásakor ezért ezen módosítottam. A módisított irányítás kódja.

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

```
        glutPostRedisplay();
}
```

Most folytassuk a színek elkülönítésével. A program 66. sora erre módosult: glColor3f (200.818f, 5.900f, 130.824f); A 188. sor ere glColor3f (30.309f, 30.820f, 30.150f); valamint a 220. sor erre: glColor3f (50.804f, 50.820f, 50.150f);

14.4. Teljes képernyős java program

A teljes képernyős porgram megvalósításához írni kell egy képernyőkezelőt ami most nekünk a screen.java lesz.

```
import java.awt.*;
//import java.awt.image.BufferStrategy;
//import java.awt.image.BufferedImage;
//import java.lang.reflect.InvocationTargetException;
import javax.swing.JFrame;

public class screen {
  private GraphicsDevice video_card; //Videókártya létrehozása

  public screen()
  {
    GraphicsEnvironment env = GraphicsEnvironment. ↩
       getLocalGraphicsEnvironment();
    video_card = env.getDefaultScreenDevice(); //elkapjuk a videókártyát
  }
```

A programunk megírásához szükségünk lesz az Abstract Window Toolkit-re (awt) amiből mindent bekérünk. Példányosítjuk a GraphicsDevice osztályt ami megmondja, hogy az adott környezetben mik érhetőek el. Utána a képernyőnk tulajdonságait lekérjük az env-be majd a GraphicsDevice objektumunknak megadjuk az alapértelmezett grafikai eszköz adatait.

```
  public void setFullScreen(DisplayMode dm, JFrame window)

  {

      window.setUndecorated(true); //nem lesz titlebar meg scrollebar
      window.setResizable(false); //nem lehet átméretezni
      video_card.setFullScreenWindow(window);

      if( dm !=null && video_card.isDisplayChangeSupported()) //van  ↩
         monitorbeállításunk és azokat lehet is állítani
        {
```

```
        try {video_card.setDisplayMode(dm);} catch(Exception ex) {}
        }
  }


  public Window getFullScreenWindow() //A tényleges függvény ami majd ↩
     teljes képernyőt fog nekünk állítani.
  {
    return video_card.getFullScreenWindow();
  }

  public void restoreScreen() { //A függvény ami levesz minket a teljes ↩
     képernyőről.
    Window w = video_card.getFullScreenWindow();
    if(w != null) {
      w.dispose();
    }
    video_card.setFullScreenWindow(null);
  }
}
```

```
import java.awt.*;
// import java.awt.event.*; //később az egérhez kell majd a ↩
   továbbfejlesztéshez.
import javax.swing.*;

public class fulscreen extends JFrame {

  public static void main (String[] args)
  {
    DisplayMode dm = new DisplayMode(800,600,16, DisplayMode. ↩
       REFRESH_RATE_UNKNOWN); //Megadjuk a képernyő adatain szélesség ↩
       magasság, színmélység és ő magának lekéri, hogyhány Hz-n frissül a ↩
       képernyő.
    fulscreen f = new fulscreen();
    f.run (dm);
  }


  public void run (DisplayMode dm) {

    getContentPane().setBackground(Color.PINK); //Háttérszín
    getContentPane().setForeground(Color.WHITE); //Előtérszín
    getContentPane().setFont(new Font("Arial", Font.PLAIN, 500)); //Bet ↩
       űtípus beállítása

    screen s = new screen(); //Képernyőosztály példányosítása
    try {
      s.setFullScreen(dm, this); //Ezt beállítjuk teljes képernyőre a dm- ↩
```

```
          ben lévő beállításokkal.
        try {
          Thread.sleep(5000);
        }catch (Exception ex) {}
    }finally {
      s.restoreScreen(); //Végül kilépünk a teljes képernyőből.
    }
    }

    public void paint(Graphics g) { //Ez kiir egy feliratot a képernyőre
      super.paint(g);
      // g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,  ↩
         RenderingHints.VALUE_TEXT_ANTIALIAS_ON); // anti aliasing-elni  ↩
         próbáltam a szöveget de nem igen ment.
      g.drawString("Felirat", 200, 200);
    }
}
```

14.5. l334d1c45

Javában lekészítettük a l33t chipper osztályt.

```
import java.io.BufferedReader; // Kell a buffer létrehozásához
import java.io.IOException; // Kivételkezeléshez
import java.io.InputStreamReader; // Beolvasás
import java.util.HashMap; //karaktertérkép létrehozásához
import java.util.regex.Matcher; //Összepárosító
import java.util.regex.Pattern; //Karakterminta

class L33tConvertor {

    private String toLeetCode(String str) {
        Pattern pattern = Pattern.compile("[^a-zA-Z0-9]");
        StringBuilder result = new StringBuilder();
        HashMap<Character, String> map = new HashMap<Character, String>();
```

Leetcode ra átalakító függvény először létrehozzuk a mintát ami egy pattern tipusú osztály amibe deklarál-juk, hogy a kis nagybetűk és a számok is benne lesznek a-Z ls 0-9 ig. A stringbuilder az nagyon hasonlít a srtingbuferre a külömbség csak annyi, hogy ezt 1 szál használja csak ebbe fogjuk tárolni a végeredményt. Végül létrehozzuk a karaktertérképet amit lentebb fejtünk ki ez megadja, hogy mit mire fog cserélni a program.

```
        map.put('A', "@");
        map.put('B', "Ăź");
        map.put('C', "Â©");
```

```
        map.put('D', "Ä`");
        map.put('E', "â,\ensuremath{\lnot}");
        map.put('F', "Ć'");
        map.put('G', "6");
        map.put('H', "#");
        map.put('I', "!");
        map.put('J', "Âż");
        map.put('K', "X");
        map.put('L', "ÂŁ");
        map.put('M', "M");
        map.put('N', "r");
        map.put('O', "0");
        map.put('P', "p");
        map.put('Q', "0");
        map.put('R', "Â®");
        map.put('S', "$");
        map.put('T', "7");
        map.put('U', "Â$\mathrm{\mu}$");
        map.put('V', "v");
        map.put('W', "w");
        map.put('X', "%");
        map.put('Y', "ÂĄ");
        map.put('Z', "z");
        map.put('0', "O");
        map.put('1', "I");
        map.put('2',"2");
        map.put('3',"2");
        map.put('4',"2");
        map.put('5',"2");
        map.put('6',"2");
        map.put('7',"2");
        map.put('8',"2");
        map.put('9',"2");

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

Fentebb láthatjuk a helyettesítő karaktereinket. Nézzük most a forciklust hiszen ez az osztályunk lelke. 0 tól a bekért string hosszáig megyünk és minden lépésnél növeljük az i-t 1-el. A szövegünkön karakterenként

végigmegyünk és egy nagy csalást követünk el, hiszen mindet NAGYBETŰRE állítuk. Ezután létrehozunk egy összehasonlítót ami kikeresi a bemenet párját. Most, hogy ne kapjunk minden ismeretlen karakternél null-t ezért megvizsgáljuk. Ha ismeretlen karakter ami nincs rajta a patternen akkor visszadobjuk ha pedig rajta van a mintánkon akkor a map.get visszaadja a csere értékét. És a végén visszaadjuk az eredményt stringé konvertálva.

```
    }

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

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

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

Végül nézzük a main függvényt. Példányosítjuk a Converter osztályunkat majd létrehozunk egy bufferreadert -karaktereket olvas be a standard bemenetről- egy Stringet majd végül a kilépéshez szolgáló int válltozót. A bekért szavunkat átadjuk a leetWorld stringnek. Létrehozunk egy új Stringet amit egyenlővé teszünk az objektumunkra meghívott függvény végeredményével aminek paraméterül megadtuk a bekért szöveget. Az eredményt kiirjuk a standard kimenetre. Végül megkérdezzük a usert, hogy szeretne-e még egy szöveget l33tesíteni.

14.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció

14.7. Perceptron osztály

15. fejezet

Helló, Stroustrup!

15.1. JDK osztályok

```
#include <boost/filesystem.hpp>
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

int main(int ac, char** av)
{
    string extension;
    vector<string> files;
    boost::filesystem::recursive_directory_iterator iterator (string("/mnt/ ↩
       d/src"));
    while(iterator != boost::filesystem::recursive_directory_iterator())
    {
      string extension = boost::filesystem::extension(iterator->path(). ↩
         filename());

        if (boost::filesystem::is_regular_file(iterator->path()) &&  ↩
           extension == ".java")
        {
          files.push_back(iterator->path().filename().string());
        }
      ++iterator;
    }
```

Ebben a feladatban boostot használva kellett kiiratni a jdk osztályokat. Ezek én a windwos-os jdk 13 osztályokat választottam. Az elején includeoljuk is be a boost filesystemet. Létrehozunk egy Stringet ez majd később hasznos lesz. Az osztályok nevét egy vektorba fogjuk beletölteni. Megadjuk a az elérési utat. A while ciklusunk azt csinálja, hogy végigmegy az összes mappán rekurzívan. A string válltozónkban eltároljuk a fájlok kiterjesztését tehát ebben a stringben csupa .java lesz. Ezzel megkatuk a .java fájlokat. Itt adódott egy probléma miszerint az src tartalmazott egy mappát ami szintén .java volt és aprogramunk

ezt is beleszámolta. Ezt kiküszöbölendő nem elég megnéznünk azt, hogy .java a fájl azt is néznünk kell, hogy eggyáltalán fájl e amit nézünk ezt hivatott végrehajtani az is_regular_file függvény a boostban. Az így kapott értékeket beletöltjük a vektorunkba.

```
  <programlisting language="C++">
    int size = files.size();
    cout << size << endl;

ofstream myfile;

    myfile.open ("kimenet.txt");

        for(int i=0 ; i<size ; i++)
         {
             myfile << files[i]<<";"<<endl;
         }

    myfile.close();

    return 0;

}
```

A vektorunk pontosan annyi elemet fog tartalmazni amennyi osztály van szóval elég csak azt kiiratni valamint, hogy fancy-k legyünk az összes osztály nevét kimentjük egy fájlba.

15.2. Másoló-mozgató szemantika

15.3. Változó argumentumszámú ctor

```
#include <iostream>
#include <cstdarg>
#include <map>
#include <iterator>
#include <cmath>
#include <random>
#include <limits>
#include <fstream>

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

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

    va_list vap;

    va_start ( vap, nof );

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

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

    va_end ( vap );

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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


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

    units[0] = image;

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

#ifdef CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );

#else

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

```
          {
            units[i][j] = 0.0;

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

            units[i][j] = sigmoid ( units[i][j] );

          }

#endif

      }

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

    return image;

  }

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

    learning ( image, y );
  }

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

    units[0] = image;

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

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

    int i {n_layers-1};

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

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

      }

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

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

            double sum = 0.0;

            for ( int l = 0; l < n_units[i+1]; ++l )
              {
                sum += 0.19*weights[i][l][j]*backs[i][l];
              }

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

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

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

    delete [] backs;

  }

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

```
        delete [] weights[i-1];
      }

    delete [] weights;

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

    delete [] units;
    delete [] n_units;

  }

  void save ( std::fstream & out )
  {
    out << " "
        << n_layers;

    for ( int i {0}; i < n_layers; ++i )
      out << " " << n_units[i];

    for ( int i {1}; i < n_layers; ++i )
      {
        for ( int j {0}; j < n_units[i]; ++j )
          {
            for ( int k {0}; k < n_units[i-1]; ++k )
              {
                out << " "
                    << weights[i-1][j][k];

              }
          }
      }

  }

private:
  Perceptron ( const Perceptron & );
  Perceptron & operator= ( const Perceptron & );

  int n_layers;
  int* n_units;
  double **units;
  double ***weights;

};
```

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

int main (int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width() * png_image.get_height();
    Perceptron* p = new Perceptron (3, size, 256, size);

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

Most láthatjuk a konstruktorban, hogy tényleg nem fogunk a képünkből 1 értéketvisszakapni hanem az átadott képet fogjuk visszakapni hiszen ezt adtuk meg. A konstruktort így kell értelmezni: mennyi argumentum lesz és utána felsorolva az argumentumok: bemeneti kép mérete 256-os érték -amire le lesz szűkítve a kép- valamint a kimenet mérete.

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

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

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

    png_image.write("output.png");
    delete p;
    delete [] image;
}
```

Végigmegyünk a kép szélességén és magasságán és az image tömbbe beletesszük a megfelelő red értéket. Létrehozunk egy új image-et. A következő for ciklussal felülírjuk a blue értékeket. Végül kimentjük az új képet és felszabadítjuk a memóriát. A double* egy tömböt térít vissza mert a hpp-ben átírtuk. Ennek segítségével tutunk belenyúlni a képbe. A header file ban a konstruktoron módosítottunk, hogy változó argumentumszámú legyen valamint a double () operátorát átalakítottuk, hogy egy tömbel térjen vissza. Ez a tömb nem más lesz mint a units tömb.

15.4. Összefoglaló extends Hibásan implementált RSA törése

Az RSA lényege, hogy fogunk egy matamatikai eljátást amit az eggyik irányba egyszerű végrehajtani, ami jelen esetben 2 db nagy prím szám szorzása. Ezt összeszorozni egyszerű viszont a szorzatból visszafejteni az eredeti számokat nem, és jelenleg sincs erre megoldás ha elég nagy számot használunk. Az RSA titkosítás 2 kulcsból ákk egy publikusból és egy titkosból. A publikus kulcs a szorzat amivel bárki kódolhat szabadon de a privát kulcsot a két hatalmas prímszámot titokban tartjuk és ezzel dekódoljuk az üzeneteket. RSA val egy küldés úgy működik, hogy a küldő megkapja a publikus kulcsot ezzel titkosítja az üzenetét amit elküld és ezt a másik oldalon a fogadó fél a privát kulcsal már rögtön tudja is dekódolni. Ez a módszer sokszor lassú ezért ritkábban használják a felhasználók adatainak a titkosítására. Erre vannak gyorsabb megoldások is. Elég a meséből nézzük a kódot(kat). A kódolás java segítségével oldottuk meg.

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.*;


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

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

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

Használjuk a BigInteger java osztály hiszen itt hatalmas zsámokkal fogunk dolgozni amit a sima integer nem tud feldolgozni mert forfítási hibát kaptam. Valamint a SecureRandom osztályt is használatba vettün amit egy titkosításhoz kellő erősségű random számot generál nekünk. Létre is hozzuk a változóinkat. A p és a q 2 benga nagy random valószínüleg prímszám amit a BigInteger.probablePrime() függvénnnyel hoztunk létre a függvény 1. paramétere a bithosszúság a másik meg random bitek amiket arra használ, hogy prímeket válasszon. A végén visszaad egy valószínüleg prím számot. A phi az a két szám-1 összeszorozva.

```
      modulus    = p.multiply(q);
      publicKey  = new BigInteger("65537");
      privateKey = publicKey.modInverse(phi);
   }
```

Megtörténik a 2 nagy prímszám szorzása ezt fogjuk modulusként használni. Megadunk egy konkrét publikus kulcsot amit bárki használhat. A privát kulcs az nekünk fontos ezt is kiszámoljuk a mod.Inverse. Ez fogja a publikus kulcsot és a reciprokán végrehajtja a modulot a phi értékével.

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

```
    }
```

Ez a függvény végzi a titkosítást fontos, hogy byte tömmböt fogunk titkosítani. Létrehozunk egy BigInteger amit vissza fogunk téríteni de még előtte egy kicsit átalakítjuk. Tehát mi kost a bytokat fogjuk a publikus kulcsal maghatványozni végül modulo műveletet is végrehatjuk vele a modulus értékével.

```
   public static void main(String[] args) {

    int N = Integer.parseInt(args[0]);
    RSA key = new RSA(N);
```

Az N értéket parancssorról kajuk meg és ezzel az értékkel példányosítjuk az RSA osztályt.

```
    String s = "Yetbedanyfortravellingassistanceindulgenceunpleasing";
    byte[] bytes = s.getBytes();
    BigInteger message = new BigInteger(bytes);
    List<BigInteger> result = new ArrayList<BigInteger>();
    byte[] atmenet = new byte[1];
    for(int i = 0; i < bytes.length; i++)
    {
        atmenet[0] = bytes[i];
        result.add(key.encrypt(atmenet));
    }
    System.out.println(result);
   }
}
```

A String s lesz a titkosítandó szöveg amit bytokká alakítunk a getBytes() függvénnyel. Ezt beletöltjük egy BigInteger-be. Most jön az érdekes rész, hiszen mi elrontottuk az RSA kódolónkat, hogy ne egyben titkosítson mindent hanem minden egyes szót titkosítson külön külön. Ehez létre kell hoznunk egy átmeneti tömböt amit a függvényünk el tud fogadni. tehát végigmegyünk a kódolandó szavunkon és mindegyik elemére külön meghívjuk a kódoló függvényünket és ezt az eredmény ArrayList-be töltjük bele. Ez lényegében olyan mint C++-ba a vektor. A legvégén kiírjuk az eredményt. A program kimenetét mostmár át is tudjuk adni a törőnknek. Nézzük a törőt.

```
class KulcsPar{

    private String values;
    private char key = '_';
    private int freq = 0;

    public KulcsPar(String str, char k){
        this.values = str;
        this.key = k;
    }

    public KulcsPar(String str){
        this.values = str;
    }

    public void setValue(String str){
```

```
        this.values = str;
    }

    public void setKey(char k){
        this.key = k;
    }

    public String getValue(){
        return this.values;
    }

    public char getKey(){
        return this.key;
    }

    public void incFreq(){
        freq += 1;
    }

    public int getFreq(){
        return freq;
    }

}
```

A kódolt karaktereket osztályba fogjuk tölteni. Ez fog tárolni kódolt értéket, a betű előfordulását illetve a valószínűség alapján feltételezett karaktert. Ehez tartozik 1 konstruktor illetve eggyértelmű függvények amiknek a nevük a megáért beszél. Most nézzük a program main részét. lényegében set és get metódusok. Az egész egy try catch bolkba van mert a java csak így ened fájlból olvasni.

```
import java.io.*;

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

            String line[] = new String[10000];

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

            inputStream.close();
```

Fájlból beolvassunk az előző program kimenetét csinálunk egy jó nagy tömböt amibe valószínüleg bel fog férni minden és amég van bejövő adata addig pumpáljuk is bele valamint növeljük a számlálót, hogy tudjuk is, hogy mennyit olvastunk be. Ha végeztünk akkor bezárjuk a fájlt.

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

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

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

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

Létrehozun egy tömböt ami KulcsPár osztály példányait fogja tartalmazni. Ennek a 0. elemét rögtön be is állítjuk ai egy új KulcsPár lesz és a konstruktort meghívtuk a a line tömb 0 elemére. For ciklussal végigmegyünk a beolvasott sorokon. A forciklusba ágyazunk még egy forciklust ami addig megy amég el nem éri a db válltozó értékét. Ha a kp tömb adott eleme megeggyezik a beolvasot sor aktuális elemével akkor az adott érték gyakoriságát növeljük, valamint feljegyezzük, hogy ilyen van,majd kilépünk a ciklusból. Ha nics talált érték akkor készítünk egy új példányt és növeljük a darabszámot.

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

Ez a dupla ciklus a sorbarendezi a példányokat a gyakoriság értékük apalján.

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

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

while((k = f.read()) != -1) {
    if((char)k != '\n') {
        key[kdb] = (char)k;
```

```
                    //System.out.println(key[kdb]);
                    kdb++;
                }
            }

            f.close();
```

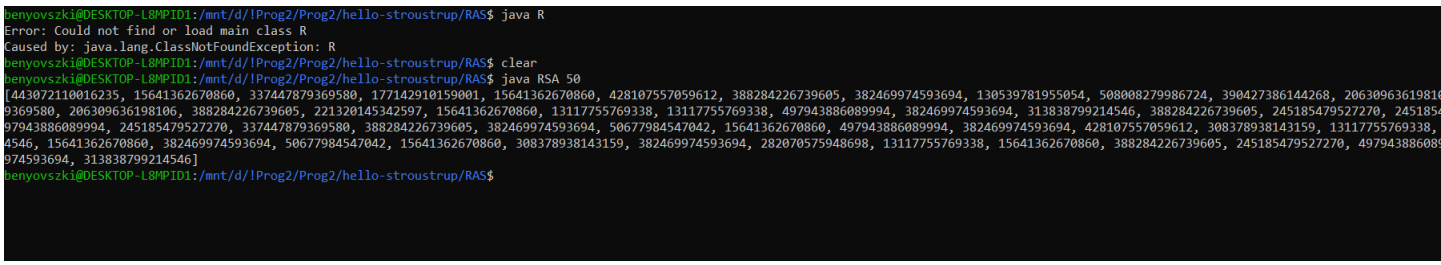
Most beolvassuk a karaktereinket amihez egy új filereaderre lesz szükségünk. A bemeneti adatokat is egy tömbbe fogjuk tárolni valamint csak akkor ha az a sor nem egy üressor valamint növeljük a karakterek darabszámát.

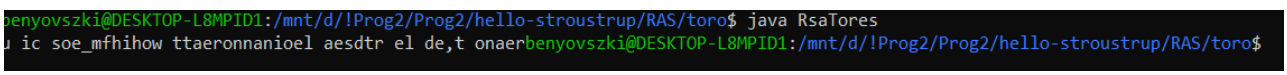
```
            for(int i = 0; i < kdb && kp[i] != null; i++) {
                kp[i].setKey(key[i]);
            }
```

Szépen feltötjük a az osztályaink key változóit a darabszámig. Ha null értéket találunk akkor egyből megállunk.

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

        } catch(IOException e) {
        }

    }

}
```

A legvégén pedig kiiirjuk a megfelelő key értékeket és így elméletileg vissza is kapjuk a szöveget. Ez elméletben szép és jó de valójában meg se közelítjük az eredeti söveget. Ehez egy alap angol abc betűgyakoriságot használtunk ami eltérhet a szövegétől. Ha ismerjük a szöveg pontos betűgyakoriságát akkor több az esélyünk de a végeredmény akkor sem lesz kielégítő.





16. fejezet

Helló, Gödel!

16.1. Gengszterek

Lambda kifejezéseket akkor érdemes használni ha inline függvényt szeretnénk definiálni amit aztán sehol máshol nem használunk fel csak ebben az 1 esetben. Ilyenkor nekünk még visszatérítési értéket sem kell megadni mert azt a fordító mjad elintézi. Itt rendezni fogunk a sort segítségével ami végég fog menni a gengstereken az elejétől a végéig és az aktuális x és y elemet fogja összemérni a randőrével és addig fog rajta dolgozni amég ki nem alakítja a sorrendet. Úgy fogja megcsinálni, hogy a legközelebbre eső gangster kerüljön a lista elejére és a legmesszebb lévő a végére.

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

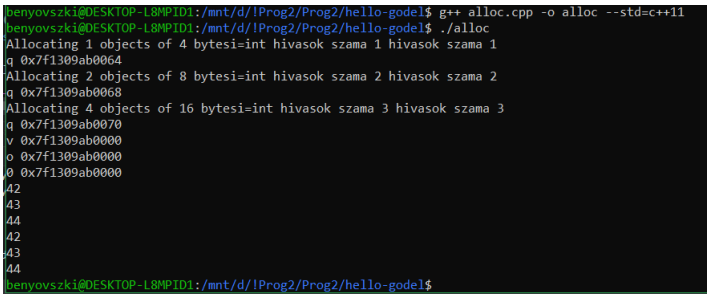
16.2. C++11 Custom Allocator

A c++ alaból rendelkezi allocatorral ami a legtöbb esetben kielégítő. Costum allocatrot akkor érdemes használni ha nagyon optimalizáli szeretnénk a kódunkat. Tehát itt most mi fogjuk megírni azt, hogy az

```
 template<typename T> struct CustomAlloc {
using size_type = size_t;
using value_type = T;
using pointer = T *;
using const_pointer = const T*;
using reference = T &;
using const_reference = const T &;
using difference_type = ptrdiff_t;

Arena& arena; CustomAlloc(Arena& arena) :arena(arena) {} int nn{ 0 };
pointer allocate(size_type n) {
```

```
    ++arena.nnn;
    ++nn;
    int s;
    char* p = abi::__cxa_demangle(typeid (T).name(), 0, 0, &s);
    std::cout << "Allocating " << n << " objects of " << n * sizeof(T) << " ↩
        bytes" << typeid (T).name() << "=" << p << " hivasok szama " << nn  ↩
       << " hivasok szama " << arena.nnn << std::endl;
    free(p);
    char* q = arena.q;
    std::cout << "q " << static_cast <const void*> (q) << std::endl;
    arena.q += n * sizeof(T); return reinterpret_cast<T*>(q);
  }
  void deallocate (pointer p, size_type n){
                            delete[] reinterpret_cast<char *>(p);
                            std::cout << "Deallocating"
                                      << n << " objects of "
                                      << n*sizeof (T)
                                      << " bytes. "
                                      << typeid (T).name() << "=" << p
                                      << std::endl;

                          }
};
```

Az elején a template arra szolgál, hogy átvegye a tipust pl. a vektornak <int>,<string>. Utána áll a struktúra neve, hogy meg tudjuk majd hívni. A memória lefoglalását az allocate végzi ami minden egyes lefutásnál az előzőnél 2* akkora helyet foglal le 1 2 4 8 és így továb. A q mindíg a szabad memóriacím elejére fog mutatni. Ha ezeket lefoglalata az teljesen jó csak ezek leloglalva is maradnak, void deallocate fogja ezeket felszabadítani, hogy ne maradjon meg mind csak az éppen aktuális foglalás. Az allocátorba nyomonkövető szövegeket is beépítettünk. A lefoglalt memória a dealloc nélkül le is marad foglalva tehát folyamatosan többet fog lefoglaln, ha viszont megírjuk a deallocatort akkor csak az aktuálisan használt memória marad lefoglalva a többi felszabadul.



16.3. STL map érték szerinti rendezése

Ebben a feladatba az volt a lényeg, hogy a programunk a map-ot ne a kulcsa alapján rendezze, hanem az érték mező szerint. Ehez létrehozunk egy parit vektort amibe előbb mindent átpakolunk, majd ennek az elemeit fogjuk rendezni csökkenő sorrendben lambda kifejezéssel, végül betölteni az ordered vektorba. És a végén visszatérítjük az ordered vektort.

```
        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;
}
```

16.4. Alternatív Tabella rendezése

Azért van szükség java.lang Interface Comparable<T> mert a sort függvény ezt használja amikor rendez, ezeket az értékeket rendezésnél lehet kulcsként használni anélkül, hogy egy külön összehasonlítót adnánk meg. Nézzük meg a kódban, hogy erre miért is van szükség.

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList(csapatok ↩
   );
java.util.Collections.sort(rendezettCsapatok);
java.util.Collections.reverse(rendezettCsapatok);
java.util.Iterator iterv = rendezettCsapatok.iterator();
...
...
class Csapat implements Comparable<Csapat> {

protected String nev;
protected double ertek;
```

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

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

Tehát azért volt szükség implementálnia az osztálynak a comparable interfészt, hogy egyszerűbb legyen megírni a compareTo függvényt ami így össze tudja hasonlítani a csapat osztályok értékeit. És ha ezekből az objektumokból készítenénk egy listát vagy tömböt akkor a sort függvény a compare to alapján sorba is rendezi nekünk. Láthatjuk fent, hogy a rendezettCsapatok Listára meghívjuk a sortot.

17. fejezet

Helló,!

17.1. FUTURE tevékenység editor

A feladatot tutorálta:Szegedi Csaba

A program felélesztéséhez szükségünk lesz egy 8-as JDK-ra hiszen a programban használt java fx csak itt létezik a többi JDK-ban nem. Erre van egy jó program az sdkmen, ennek segítségével át is tudjuk válltani az sdk-t és már mehet is a program. Ha sikerült lefuttatni akkor észrevehetjük, hogy a porgramban sok hiba van a sok közül az egyik az, hogy nem képes több altevékenységet létrehozni ezt kiküszöbölendő. Az altevékenységet létrehozó kódot ciklusba kell helyeznünk.

Eredeti kód:

```
public TextFieldTreeCell(javafx.scene.control.TextArea propsEdit) {
           this.propsEdit = propsEdit;
           javafx.scene.control.MenuItem subaMenuItem = new javafx.scene. ↩
              control.MenuItem("Új Altevékenység"););
           addMenu.getItems().add(subaMenuItem);
           subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> {
               java.io.File file = getTreeItem().getValue();

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

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

                   System.err.println("Cannot create " + f.getPath());

               }
           });
```

A módosított kód.

```
        public TextFieldTreeCell(javafx.scene.control.TextArea propsEdit) {
            this.propsEdit = propsEdit;
            javafx.scene.control.MenuItem subaMenuItem = new javafx.scene. ↩
               control.MenuItem("Új Altevékenység");
            addMenu.getItems().add(subaMenuItem);
            subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> {
                java.io.File file = getTreeItem().getValue();

                int i = 1;
                while (true) {
                    java.io.File f = new java.io.File(file.getPath() +  ↩
                       System.getProperty("file.separator") + "Új  ↩
                       Altevékenység" + i);

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

                    }
                }
            });
```

Most ha sikeresen létrehoztuk az altevékenységet akkor a break utasítás-al kilépünk a végtelen ciklusunkból, amennyiben nem sikerült a létrehozás, akkor error üzenetet küldünk és újra elkezdődik az altevékenyég létrehozása akkor hibaüzenetet küldünk és növeljük az i-t.

17.2. OOCWC Boost ASIO hálózatkezelése

Itt ez a fájl beolvasással foglalkozik. Az sscanf formázott adatokat olvas be a bemenetről. A scan f 4 külömböző adatot vár. A while ciklusra azért van szükségünk mert addig szeretnénk beolvasni amég van megfelelő adatunk ezt a sscanf visszatérési értékével ellenőrizzük ami a sikeresen beolvasott adatok száma lesz. Az n az a végén megkapja int ben, hogy összesen mennyit olvastunk be szóval ő nem számít bele a beolvasott adatok mennyiségébe. Ennek segítségével azt is meg tudjuk nézni, hogy összesen mennyit olvasunk be hiszen ezt a számot mindíg hozzáadjuk az nn-hez. Minden futásnál a data+nn-edik helyen lévő adatot fogjuk beolvasni így nem fogjuk többször ugyan azt a sort olvasni.

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

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

17.3. SamuCam

A program parancssori argumentumként kapott IP-címről nyit meg kamerát. A program elindításához szüskég lesz a lbpcascade_frontalface.xml-re. Ezt a program könyvtárába kell helyezni. Láthatjuk a programmból, hogy a gyökérben fogja keresni a fájlt std::string faceXML = "lbpcascade_frontalface.xml";.

```
     #include "SamuCam.h"

SamuCam::SamuCam ( std::string videoStream, int width = 176, int height =  ↩
   144 )

  : videoStream ( videoStream ), width ( width ), height ( height )

{
  openVideoStream();
}

SamuCam::~SamuCam ()

{}

void SamuCam::openVideoStream()

{

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

}
```

Megnyitjuk a webkameránkat az OpenVC segítségével és beállítjuk. A videoCapture.open (videoStream) függvény megnyitja a kamerát. Amennyiben mi nem IP-ről szeretnénk kamerát nyitni hanem az eszközünkről a zárójelben lévő értéket módosítanukn kell 0-ra. Utána a .set() fügvényekkel beállítjuk a szélességet és a magasságot valamint, hogy mennyi FPS-el fusson.

```
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;
```

Példányosítunk egy CascadeClassifiert ez az OpenCV-ben objektumok felismerésére alkalmas. Itt látható, hogy a program megnyitja az xml fájlt. Ezt megadjuk a classifier-nek, hogy lehetővé váljon az arcfelismerés. Utána egy rövid hibakezelés következik, ha nem sikerül a .load függvénynek betöltenie az xml-t. Valamint még példányosítjuk a mat.ot ami egy n dimenziós mátrixosztály ebbe fogjuk tárolni a kép adatait.

```
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,  ↩
               0, cv::Size ( 60, 60 ) );

            if ( faces.size() > 0 )

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

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

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

              }

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

            emit  webcamChanged ( webcam );

          }

        QThread::msleep ( 80 );

      }

    if ( ! videoCapture.isOpened() )

      {
        openVideoStream();
      }
  }
}
```

Végül elemezzük a program lelkét a while ciklust ami addig fog menni amíg meg van nyitva a webcam. A cv::resize()-al átméretezzük és interpoláljuk a cv::INTER_CUBIC-segítségével ami pixletérképet készít. A frame-et rjuk felül hiszen az a függvény bemenete és kimenete is. Ezután a frame-et a cv::cvtColor segítségével szürkére állítjuk és ennek az adatait a frissen létrehozott cv::Mat grayFrame;-be töltjük. A cv::equalizeHist kieggyenlíti a hisztogrammot. .detectMultiScale()-t a fej megkereséséhez használjuk a függvény különböző méretű téglalapokkal tér vissza. Ha találtunk fejet akkor annak az adatát eltároljuk az onlyFace-be. A képet átszinezzük pirosra majd az emit-el kiküldjük. Ezt a jelet majd a SamiBrain szépen feldolgozza. Ezután még egy képet készítünk ami megint elküld egy jelet ami jelezni fogja a SamuBrain-nek, hogy jó lenne frissíteni a webcamképet. Mindezek után rövid várakozás következik és kezdődik elölről.

17.4. BrainB

QT-ban a slot signalt küklömböző objektumok közötti kommunikációra használhatjuk. Ezek a függvények akkor fognak meghívódni ha bekövetkezik egy bizonyos esemény amiről kód küld egy emit-et. A megfelelő emit-et fogja befogni a connet .

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

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

A fogadó helyet megjelöljük és a SIGNAL() függvényben megadjuk, hogy milyen paramétereket várunk a heroesChanged jeltől, ezt melyik objektumnak továbbítsuk és a SLOT za, hogy mit továbbítunk jelen azt, hogy fusson le az updateHeroes()-a kapott paraméterekkel. A másik connect is hasonlóan működik. Ott a befejező időt fogja megkapni int-ben. Az endAndStat leállítja a programot és kiírja, hogy az alábbi könyvtárban találod a végeredményt.

18. fejezet

Helló, Berners-Lee!

18.1. A C++ és A java könyv összehasinlítása (Objektumorientáltság)

Mind a két programozási nyelv Objektumorientált, ez azt jelenti, hogy remek egységekbe tudunk zárni adatokat támogatják az öröklést és az adatrejtést is. Ez azt jelenti, hogy az adatok egy csoportjára (pl.bank vagy emberek) megtalálhatóak az osztályokban az összes szükséges változó és függvény. Az Objektumok célja a programozó életének a megkönnyítése, azomban vigyázni kell mert az objektumorentáltság egyúttal a program sebességéből is visszavesz. Ezeket az osztályokat C++ nylven a **class** szóval készíthetjük el. A java osztályok nélkül nem is képes létezni ezért annak a legkissebb egységei is maguk az osztályok. Ezeket az osztályokat lehet példányosítani mind a két nyelven azaz létrehozni belőlük egy válltozót pl. a **class Kecske-ből** létrehozzuk **Kecske Giza** objektumát. Az így létrehozott egyedek saját maguknak csinálnak egy változatot osztály válltozóiból. A függvények csak egyszer kerülnek bele a memóriába. A bennük lévő válltozókat **.-al vagy -> -al** módosíthatjuk és a hozzájuk rendelt függvényeket is ezzel érhetjük el. Például így elérhetjük **Giza.kor** válltozóját ami legyen egy int és módosíthatjuk is azt. Ugyan így meghívhatunk egy függvényt is **Giza.etet(20)** ami azt jelenti, hogy végrehajtja a függvényt és ami benne van. A hibák elkerülése végett érdemes a válltozókat private-ra állítani. C++ -ban úgy érjük el, hogy az osztálydeklarálást követően private: szó után felsoroljuk őket. Ez azt jelenti, hogy csak az adott osztály függvényei férnek hozzá a válltozókhoz ebben az esetben írnunk kell függvényeket melyekkel módosíthatóak valamint lekérhetőek a válltozók értékei. Ez a módszer azért jó mert akkor csak a megadott függvények segítségével érhetőek el a válltozók és nem lesz hiba pl. más érték kerül bele mint amit várnánk. Mind a 2 nyelvben ha nem állítjuk be public-ra a válltorzókat akkor private-lesz. Ez C++-ban public: untáni felsorolást jelent javában pedig bele kell írni a class nevébe tehát **class public Kecske**. A classon belüli válltozóknak adhatunk meg kezdőértéket, ha ezt nem tesszük meg akkor a kezdőérték c++-ban random lesz Javában pedig 0. Ezek hibázoz is vezethetnek később a program futás közben. Kezdőértéket többféleképpen is beállíthatunk: osztályon belül, példányosítás után a módosító függvény meghívásával (sok sor és feleselges lépések) vagy készíthetünk egyedi konstruktort is ami beállítja a példány értéketit. A konstruktor függvény akkor kapunk ha a class nevéből csinálunk a classon belül egy függvényt. pl**Kecske(int kor, string nem){this->kor = kor; this->nem = nem;}**. Ha ezt a függvényt elkészítettük akkor a következőképp példányosíthatunk: Kecske Giza(12, fiu);

18.2. A python nyelv bemutatása

A python rengeteg eszközön elérhető programozási nyelv. A python kódok futtatásához nincs szükség fordításra ezeket interpreter végzi. Általában nyelv azért is egyszerű mert itt nem kell minden sor után ; -rakni hiszen minden utasítás a sor végéig tart. A tagolásokat itt enerekkel és tabokkal lehet elvégezni. Természetesen itt is vannak előre lefogalalt szavak. Ebben a nyelvben is vannak változók természetesen. Itt nem kell nekünk előre megondani, hogy mi is lesz az adott válltozó, azt majd az interpreter kitalálja abból, hogy mit rendeltünk adott esetben a válltozóhoz (számot betűt stb.). Érdekes adattípusokkal találkozhatunk a python nyelvben. Ilyenek pl. a listák. Ezekben bármennyi elemet felsorolhatunk és nem kell azonosaknak lennie ez azt jelenti, hogy lehetnek benne stringek, számok, tizedestörtek stb. Vannak a Tuple-ök amik lényegében rendezett lezárt listát alkotnak. Ezek foxek amég a listához hozzá lehet fűzni és elemeket válltoztatni addig itt ez nem lehetséges minden elem fix csak csinálni és törölni lehet őket. Nos, hogy mi éppen melyiket csináltuk az attól függ, hogy milyen zárójelet használtunk. () sima zárójellel a Tuple-öket lehet létrehozni. Tehát egy példa goat = (21 'male' 2) hatunk is dolgokat a listát [] kapcsoszárójellel hozzuk létre. Ezek érdekes lehetőségeket biztosítanak. Szótárakat is létrehozhatunk itt minden elemhez hozzárendelhetünk saját nevet amivel hivatkozhatunk rá tehát nem kell mondjuk kecske[2] vel hivatkozni hanem mondjuk létrehozzuk a kecske szótárat kecske = {"weight":21, "gender" : "male" , "age" : 3} és akkor itt hivatkozhatunk az értékekre a revükkel ezeket is lehet válltoztatni. A pythonban találhatunk rengeteg beépített függvényt is ami megkönnyíti a gyors és eredményes munkát. Függvények terén a már megszokottak itt is jelen vannak if while for stb. Itt figyelni kell, hogy nem {}-el jelöljük, hogy meddig tart hanem behúzásokkal amiket vagy tab vagy szóközzel rakhatunk le. Fontos, hogy egységes legyen a tagolás tehát minden egybetartozó rész ugyan annyira legyen behúzva. Tehát egy for ciklust a következőképp írhatunk be

```
for x in goat :
      print x
```

A pythonban osztályokat is könnyedén létrehozhatunk a class kulcsszóval valamint függvényeket a def kulcszóval működési elvük szinte ugyan az mint a óz eddig tanult programozási nyelveké csak a függvény paramétereknél pl nem kell megadni, hogy milyen lesz mert azt majd a kapott adatokbók ki fogja találni. A classoknál lehetőség van az öröklésre is. A könyv nem igazán tér ki az osztályokra igazán mélyen. Hibakezelésre is van lehetőségünk mégpedik aa try: valamint az expect: parancsok használatával. A pythonban is működik a már eddig használt kivételkezelés A python nyelvhez rendkívül sok modult lehet letölteni ami szintén megkönnyíti a fejlesztést. A könyv ezután példákat mutat a fent említett dolgokra.

IV. rész

Irodalomjegyzék

18.3. Általános

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

18.4. C

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

18.5. C++

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

18.6. Lisp

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

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

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