# Univerzális programozás

## Írd meg a saját programozás tankönyvedet!



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

**COLLABORATORS**

| | *TITLE* :<br><br>Univerzális programozás | |
|---|---|---|

| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
|---|---|---|---|
| WRITTEN BY | Bátfai, Norbert | 2019. december 2. | |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|---|---|---|---|
| 0.0.1 | 2019-02-12 | Az iniciális dokumentum szerkezetének kialakítása. | nbatfai |
| 0.0.2 | 2019-02-14 | Inciális feladatlisták összeállítása. | nbatfai |
| 0.0.3 | 2019-02-16 | Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába. | nbatfai |
| 0.0.4 | 2019-02-19 | Aktualizálás, javítások. | nbatfai |
| 0.0.5 | 2019-02-27 | Hello Turing! | mariannranyhoczki |
| 0.0.6 | 2019-03-06 | Hello Chomsky! | mariannranyhoczki |
| 0.0.7 | 2019-03-13 | Hello Caesar! | mariannranyhoczki |
| 0.0.8 | 2019-03-20 | Hello Mandelbrot! | mariannranyhoczki |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| 0.0.9 | 2019-03-27 | Hello Welch! | mariannranyhoczki |
| 0.0.10 | 2019-04-03 | Hello Conway! | mariannranyhoczki |

# 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?

## 1.2. Milyen doksikat olvassak el?

- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [KERNIGHANRITCHIE]
- [BMECPP]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.

## 1.3. Milyen filmeket nézzek meg?

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

# II. rész

# Tematikus feladatok

**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!

Egy mag 100 százalékban:

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

  return 0;
}
```

Azt,hogy a megírt ciklus valóban 1 magot használ 100 százalékon egy másik terminálban top paranccsal ellenőrizhetjük. A folyamatot a Ctrl+C billentyűkombinációval lőhetjük le legegyszerűbben.

Egy mag 0 százalékban:

```
#include <unistd.h>
int
main ()
{
  for (;;)
    sleep(10000);

  return 0;
}
```

Ha azt szeretnénk hogy egy végtelen ciklus ne használja a magot akkor "elaltathatjuk" a sleep függvénnyel mely után a zárójelben megadott szám az altatás idejét jelzi másodpercekben. Másik terminálban a top paranccsal ellenőrizhetjük hogy valóban nem használ-e egy magot sem.

Minden mag 100 százalékban.

```
#include <omp.h>
int
main ()
{
#pragma omp parallel
{
  for (;;);
}
  return 0;
}
```

Ha azt szeretnénk hogy a program több szálat használjon használhatjuk az OpenMp könyvtárat de ekkor a fordítás a -fopenmp kapcsolóval történik. Ilyenkor a programot a parancs szerint párhuzamosan , egyszerre több szálon futtaja. A helyes működést a rendszerfigyelőben ellenőrizhetjük és ha jól működik a program láthatjuk hogy több CPU-t is használ.

## 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:

```
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

Mit for kiírni erre a `T1000(T1000)` futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

## 2.3. Változók értékének felcserélése

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

```
#include<stdio.h>

int main()
{
int a=7, b=19;
  printf("a=%d b=%d\n",a,b);

  a=(a-b);
  b=(a+b);
  a=(b-a);

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

return 0;

}
```

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

Két változó értékének felcserélését segédváltozó nélkül is könnyen megtehetjük, csupán egy kis matematika szükséges hozzá. Először az 'a' változót egyenlővé teszem 'a-b'-vel, majd a 'b' változót 'a+b'-vel, végül pedig az 'a' változót 'b-a'-val. Így az értékek felcserélődnek és a printf segítségével kiirathatjuk őket a terminálba. A % jel után pedig d betűt írunk, ezzel jelezve hogy a kiírt számok decimális, előjeles egészek lesznek.

## 2.4. Labdapattogás

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

```
Program T1000

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

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

```
    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, "O" );

        refresh ();
        usleep ( 100000 );

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

        if ( x>=mx-1 ) { // 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;
}
```

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

Ennél a vizuális feladatnál a labdánkat egy karakterrel ,a 0-val jelöljük. A feladat az hogy ha a képernyő széléhez ér a labda akkor visszapattanjon, ezért használjuk az ncurses könyvtárat,amely tartalmazza a getmaxyx-et amellyel a terminál kiterjedéseit adjuk meg.Az if feltételekkel adjuk meg a labdának azt ,hogyha elérte a terminál kiterjedését akkor pattanjon vissza.

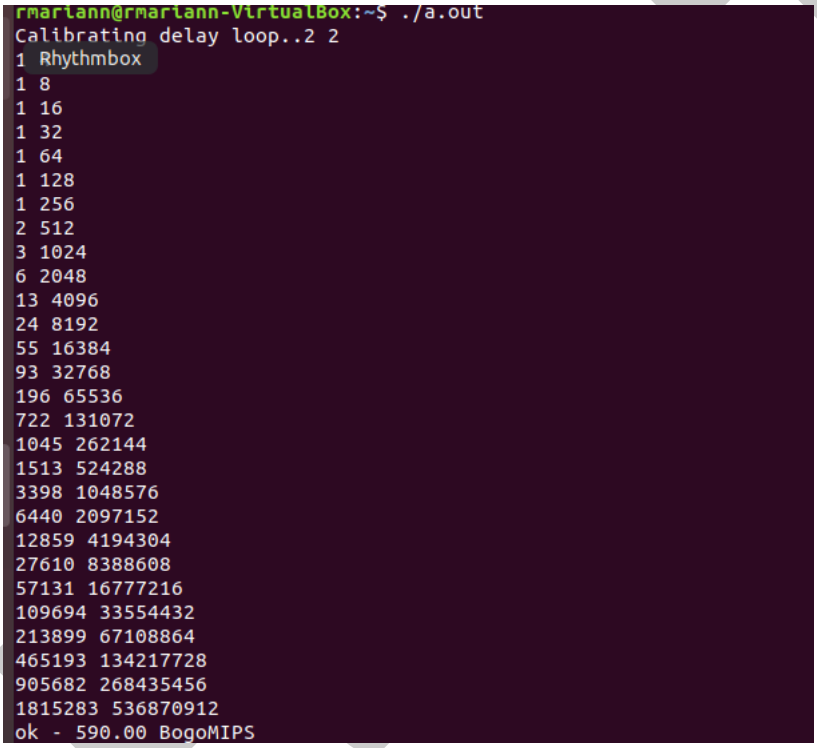
## 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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!

```
Program T100

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

void
delay (unsigned long long int loops)
{
unsigned long long int i;
for (i = 0; i < loops; i++);
}

int
main (void)
{
unsigned long long int loops_per_sec = 1;
unsigned long long int ticks;

printf ("Calibrating delay loop..");
fflush (stdout);

while ((loops_per_sec <<= 1))
{
ticks = clock ();
delay (loops_per_sec);
ticks = clock () - ticks;

printf ("%llu %llu\n", ticks, loops_per_sec);

if (ticks >= CLOCKS_PER_SEC)
{
loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
(loops_per_sec / 5000) % 100);

return 0;
}
}

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

A MIPS a Million Instructions Per Secon rövidítése. Ez a számítógép számítási sebességének a mértékegysége. A kernelnek szüksége van egy időzítő ciklusra, amit a processzor sebessége alapján határoz meg.Ezért a kernel megméri hogy egy bizonyos ciklus mennyi idő alatt fut le a számítógépen. A Bogo az angol bugus szóból ered amelynek jelentése hamis és azért adták neki ezt a nevet mert csak egy megközelítő értéket ad de nem pontos. A program futásakor az értéket a loops_per_sec változóba teszi.

Nekem a következő eredmény jött ki




## 2.6. Helló, Google!

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

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

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

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

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

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

  return sqrt(osszeg);
}

int
main (void)
{

  double L[4][4] = {
    {0.0, 0.0, 1.0 / 3.0, 0.0},
    {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
    {0.0, 1.0 / 2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0 / 3.0, 0.0}
  };

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

  int i, j;

  for (;;)
    {

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

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

    }

  kiir (PR, 4);

  return 0;
}
```

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

## 2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

A Brun-tétel szerint az ikerprímek (azaz az olyan prímek melyeknek különbsége 2 és egymás után következnek) reciprokának az összege konvergál egy értékhez, melyet Brun-konstansnak nevezünk.

A következő program/szimulácuó R nyelven íródott.

```
sumTwinPrimes <- 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 = sumTwinPrimes)
plot(x,y,type="b")
```

A prímszámokkal való számítás miatt meghívjuk a matlab csomagot. Egy függvény létrehozásával kezdünk, majd a primes(x) fügvénnyel a príszámokat adjuk meg. A diff vektor tartalmazza azoknak a prímszámoknak a különbségét amelyek egymás után következnek. Az idx vektor az ikerprímeket tartalmazza. A t1primes és a t2 primesben vektorokban tároljuk a 2 ikerprímet. A returnben sum fügvénnyel visszaadjuk az eredményt. A plottal megrajzolva látható, hogy a számok ténylegesen egy értékhez tartanak.

## 2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

A Monty Hall probléma arról szól hogy ha van 3 ajtóm és az egyik mögött nyeremény van akkor ha választunk egyet és a választásunk után kinyitnak egy másik ajtót amiben nem található nyeremény akkor

ugyanannyi esélyünk marad e nyerni az elsőre választott ajtóval mintha megváltoztatnánk a választásunkat. A számítógépes szimuláció alapján számít, hogy másik ajtót választunk, ugyanis ekkor megduplázódik az esélyünk a nyerésre.

Az első ajtó kinyitásakor 1/3 az esélye, hogy a kinyitott ajtó a nyertes és 2/3, hogy nem az. Ezután egy üres ajtó kinyitásakor ha nem változtatunk a döntésünkön, akkor 1/3-ad marad az esélye a nyerésnek,de ha változtatunk akkor a szimuláció alapján az esélyünk 2/3-adra növekszik.

Ennek bizonyítását láthatjuk R nyelven megfogalmazva, egy televíziós műsorra levetítve:

```
kiserletek_szama=1000000
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)
```

Először meg kell adnunk az elvégezendő kísérletek számát, melyeket a for ciklusban vizsgalunk majd. Ezután megadjuk az ajtó számát melyet a játékos választott, majd for ciklussal vizsgáljuk ,hogy a játékos eltalálta-e elsőre a jó ajtót, ez akkor történik meg ha a kisérlet tömb első eleme megegyezik a játékos tömb első elemével. Ha a játékosnak ez sikerült, akkor a mibol vektorba beletesszük azokat az értékekekt amelyeket a játékos nem választott. Ha nem sikerült eltalálni a helyes ajtót akkor a mibol vektorba szintén 2 érték kerül, az egyik az ajtó a nyereménnyel a másik pedig az az ajtó ,ami nem nyer ,de nem is választotta a játékos.

A nemvaltoztatesnyer vektor tárolja majd az eseteket amikor első próbálkozásra sikerült a nyereményt választani a játékosnak, azaz ha a kísérlet tömb eleme megegyezik a játékos elemével. A valtoztatesnyer

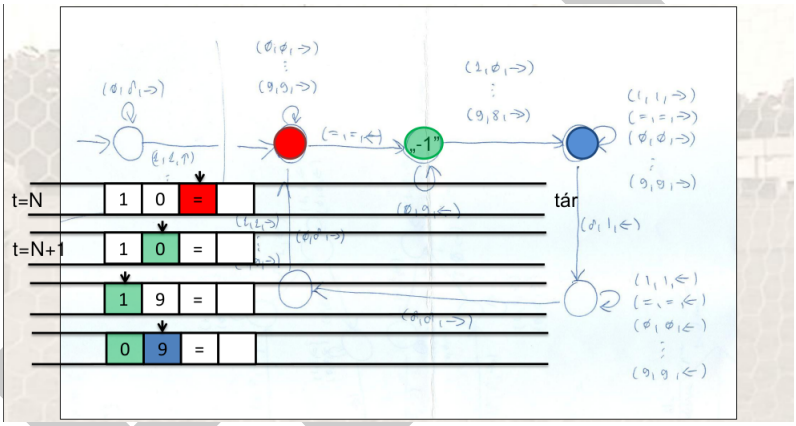
vektorba pedig megkapjuk azokat az eseteket amikor a játékos változtatással nyert. Eztuán a lenght-el kiiratjuk az esetek számait.

# 3. fejezet

# Helló, Chomsky!

## 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

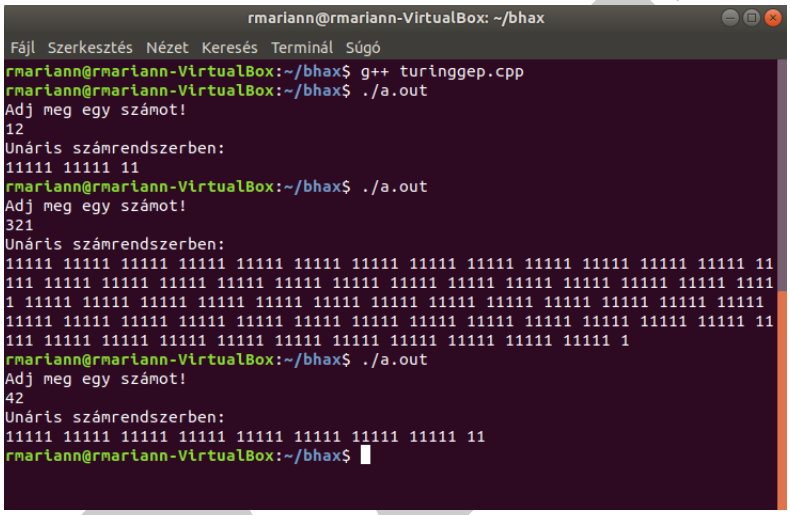



A feladatom egy decimális ,azaz tízes számrendszerből egy unáris, azaz egyes számrendszerbe váltó gép írása. Az unáris a legegysezreűbb számrendszer amely a természetes számok ábrázolására szolgál. A célja hogy egy szám beírása után annyi vonal jelenjen meg amennyi maga a szám értéke. A következő Turing gépben a vonalak helyett 1-esek fognak megjelenni. Működési elve hogy a számok beolvasása után ha talál egyenlőségjelet akkor az előtte lévő számból kivon 1-et, amíg az 0 nem lesz. Minden kivont 1-es után kiír egy vonalat (esetünkben egy 1-est). A végső sorozatban az egyesek számának pontosan meg kell egyezni a a megadott szám értékével.Az itt látható programban minden 5 darab 1-es után szóköz jelenik meg de ez nem számít egy kivont értéknek.

```
#include <iostream>
using namespace std;
int main() {
    int szam;
    int szamlalo = 0;

    cout << "Adj meg egy számot!\n";
    cin >> szam;
    cout << "Unáris számrendszerben:\n";
```

```
    for(int i = 0; i < szam; ++i) {
        cout << "1";
        ++szamlalo;
        if(szamlalo%5 == 0)
            cout << " ";
    }
    cout << '\n';
    return 0;
}
```

A következő képen a kód tesztelése látható:




## 3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen

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

Első előadás feladat passzolása

## 3.3. Hivatkozási nyelv

A C nyelvnek rengeteg szabványa van,de leggyakrabban a C18-as szabványt használjuk. Mivel több verzió van ezért van hogy az egyik verzióban megírt program nem biztos hogy kompatibilis lesz a másik verzió fordítójával.Például a C89-es szabványban a for ciklus fejében deklarált változót a fordító nem fordítja le, valamint a // kommentjelet sem ismeri ezért a /**/ komment jelölést kell használni és a printf függvény itt sprintként ismert.

```
#include <stdio.h>
#include <math.h>
int main ()
{
   char str[80];
```

```
    sprintf(str, "A Pi értéke: %f", M_PI);
    puts(str);
    return(0);
}   //Ez a program kiírja a standard kimenetre a pi ↩
    értékét
```

## 3.4. Saját lexikális elemző

Lexikális elemzőre főként akkor van szükségünk hogyha nagyon adattömeggel dolgozunk. Egy elemzővel könyebben kinyerhetők statisztikai adatok,szavakat tudunk osztályozni és például grafikus ábrázolásra is alkalmas.

```
import nltk
from nltk.tokenize import word_tokenize
text = open('elemzo.txt').read()
tokens = word_tokenize(text)
digit_count = len(set(word for word in tokens if word.isdigit() ↩
   ))
print('The number of digits in the text: ')
print(digit_count)
```

Miután az elem.txt tartalmát odaadtuk a text változónak szavakra bontjuk a stringet és megkeressük azokat a stringeket amik csak számból állnak.A set függvény miatt azokat a megegyező (duplikált) stringeket csak egynek veszi. Ezután a len() függvénnyel megkapjuk a számból álló stringek számát melyet beleírunk a digit_count változóba.A végén pedig printf-el kiíratjuk az eredményt.

## 3.5. l33t.l

Egy új írási mód terjedt el a 80-as években annak érdekében hogy bizonyos fájlokra kulcsszavas keresés során nehezebb legyen rátalálni.Ekkor az ABC betűit olyan karakterekre cserélték amellyel a szöveg olvasható maradt,de nehezebben megtalálható. Például az A betűt 4-esre cserélték vagy a C-t nyitó zárójelre. Ezt az írásmódot, leetspeak-nek nevezték el.

A következő egy példa az 1337-es kódolóra.

```
import nltk
text = open('text.txt').read()
text = text.lower()
text = text.replace('a', '4')
text = text.replace('b', '8')
text = text.replace('c', '(')
text = text.replace('e', '3')
text = text.replace('g', '6')
```

```
            text = text.replace('i', '1')
            text = text.replace('l', '|')
            text = text.replace('o', '0')
            text = text.replace('s', '5')
            text = text.replace('t', '7')
            text = text.replace('x', '8')
            text = text.replace('z', '2')
            print(text)
```

Az előző program pythonban íródott .Először importáljuk az nltk könyvtárat.A kódunk a kapott inputban kicseréli a betűket a megadott karakterekre, majd ezeket elmenti és a printf segítségével megjeleníti az outouton. Erre példa: "informatika -> 1nf0rm471k4"

## 3.6. A források olvasása

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

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

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

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

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

Egy for ciklus, melyben a változó értékét 4-ig növelem. A ++i-vel a változó értéke mindig eggyel nő.

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

Egy for ciklus, melyben a változó értékét 4-ig növelem. A i++-al a változó értéke mindig eggyel nő. Ha egy kifejezésben csak egy változó szerepel, akkor használhatjuk a ++i-t és az i++-ot is,de kifejezés kiértékelésénél már számít melyiket használjuk.

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

Egy for ciklus 0-tól 4-ig,de a végrehajtás sorrendje hibás, mert értékadás történik az i változó növelésénél.

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

For ciklus, mely n-szer fut le és a d pointer értékét odaadja az s pointernek, majd növeli a változók értékét

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

A printf-el az outputra kiírunk két fügvényvisszatérési értéket. De ez nem ajánlott mert nem tudhatjuk melyik fog előbb lefutni és melyik érték melyik függvényhez tartozik.

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

Kiíratjunk az outputra decimális formában az f(a) függvény visszatérési értékét és az a változót.

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

Kiíratjunk az outputra decimális formában az f(a) függvény visszatérési értékének referenciáját és az a változót.

## 3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})))$
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})\wedge(SSy \ ↩
   text{ prím})))$
$(\exists y \forall x (x \text{ prím}) \supset (x<y)) $
$(\exists y \forall x (y<x) \supset \neg (x \text{ prím}))$
```

I. Minden x-re létezik olyan y, amely nagyobb nála és egy prímszám.

II. Minden x-re létezik olyan y, amely nagyobb nála, és y is és y+2 is egy prímszám.

III. Minden x-re létezik olyan y , hogy x prím, ha x kisebb y-nál.

IV. Minden x-re létezik olyan y, amely kisebb nála ha x nem prím.

## 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 típusú változó.
- ```
  int *b = &a;
  ```

  Egész típusú változóra mutató pointer.
- ```
  int &r = a;
  ```

  Egy egész típusú referencia, mely megkapja a értékét.
- ```
  int c[5];
  ```

  Egy 5 elemű tömb mely egészeket tartalmaz.
- ```
  int (&tr)[5] = c;
  ```

  Egy ötelemű tömb referenciája mely a c tömb címére hivatkozik.
- ```
  int *d[5];
  ```

  5 elemű tömb, mely pointereket tartalmaz.
- ```
  int *h ();
  ```

  Függvény, melynek a visszatérérsi érteke egy pointer.
- ```
  int *(*l) ();
  ```

  Egy pointer mutat egy mutató pointert visszaadó függvényre.
- ```
  int (*v (int c)) (int a, int b)
  ```

  Függvény, melynek ha megadunk két értéket, visszaad egy egész típusú változót, majd visszatér a változóra mutató pointerrel.
- ```
  int (*(*z) (int)) (int, int);
  ```

Mivel az előző kódok referenciák ezért a g++ paranccsal fordíthatóak.

# 4. fejezet

# Helló, Caesar!

## 4.1. double ** háromszögmátrix

Írj egy olyan `malloc` és `free` párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

Megoldás videó: https://youtu.be/1MRTuKwRsB0, https://youtu.be/RKbX5-EWpzA.

A háromszögmátrixok olyan mátrixok melyek ugyan négyzet alakúak de főátlójuk alatt vagy felett csak 0 érték található. Jelen esetben alsó háromszögmátrixról van szó, melynek fátlója FELETT vannak a nullértékek.

```
#include <stdio.h>
#include <stdlib.h>
int
main ()
{
    int nr = 5;
    double **tm;
    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }
    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL ↩
           )
        {
            return -1;
        }
    }
    for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
            tm[i][j] = i * (i + 1) / 2 + j;
    for (int i = 0; i < nr; ++i)
    {
```

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

Az nr változó megadja a mátrix magasságát, majd a malloccal megadott méretű területet foglalunk. Ezt úgy tesszük hogy a double* méretét megszorozzuk az nr változóval és az így kapott számú bitnyi helyet foglalunk(jelen esetben 5*8=40).Ezután a for ciklusban a mutatók értékét mindig 1- el növeljük, majd értékeket adunk a változóknak.Végül a printf segítségével kiíratjuk őket majd a free-vel felszabadítjuk a területeket.

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Ebben a feladatban a kizáró vagyos (XOR) titkosítás van bemutatva.A kizáró vagy művelet 1 értéket ad, ha a két bit különböző és 0-t ha megegyező. Például:

A xor titkosítás nem a legbiztonságosabb titkosítási mód, de kevésbé fontos adatok titkosítására tökéletesen megfelel.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define MAX_KULCS 100
#define BUFFER_MERET 256
int
main (int argc, char **argv)
{
  char kulcs[MAX_KULCS];
  char buffer[BUFFER_MERET];
  int kulcs_index = 0;
  int olvasott_bajtok = 0;
  int kulcs_meret = strlen (argv[1]);
  strncpy (kulcs, argv[1], MAX_KULCS);
  while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
    {
```

```
      for (int i = 0; i < olvasott_bajtok; ++i)
  {
    buffer[i] = buffer[i] ^ kulcs[kulcs_index];
    kulcs_index = (kulcs_index + 1) % kulcs_meret;
  }
      write (1, buffer, olvasott_bajtok);
    }
}
```

A program elején meg kell adnunk a kulcs és a buffer méretét.VAlamint létrehozunk egy kulcs változót, amellyel nyomon tudjuk követni hogy a titkosításnál melyik karakternél járunk és létrehozunk egy másik változót is amely megadja hogy hány bájtot olvasunk be az inputról. Ezután a while ciklusban karakterenként megtörténik a titkosítás. Ha a folyamatban a kulcs végéhez értünk kezdődik előről mindaddig amíg a szövegünk titkosítva nem lesz. , majd kiirathatjuk. Az algoritmusnak az a feladata hogy összehasonlítsa az előre megadott kulcsban lévő értékeket a bitsorozatban lévőkével.Ha az érték különböző akkor 0-t kapunk, ha pedig ugyanaz akkor 1-et, mely bitsorozatból a xor művelet használata után a titkosítatlan adatokat kapjuk meg.

aAz előző algoritmus fordításkor a c99 szabványt kell használnunk:

```
            gcc exor.c -o e -std=c99
        x
```

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

```
public class ExorTitkosító {

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

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;
        while((olvasottBájtok =
                bejövőCsatorna.read(buffer)) != -1) {

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

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

```
            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

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

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Mivel a Java egy objektumorientált nyelv ezért itt classokkal dolgozuk. A String[] args-el átadhatóak a programnak a parancssori argumentumok.A try-ban tároljuk a ülönböző utasításokat és ha hibát talál azt jelzi.A catch "elkapja" a hibát és egy hibaüzenetet ad vissza. A try -ban helyet foglalunk az ExorTitkosito függvénynek amely eztán megkapja a kimenetet a bemenetet és a kulcsot is. Létrehozunk egy kulcs és egy buffer tömböt majd a kulcs tömbbe beolvassuk magát a kulcsot a getBytes segítségével. A while ciklus addig tart amíg már nem tud többet beolvasni vagy addig ameddig el nem éri a karaktersorozat a buffer méretét (ezt előzőleg 256 bájtban adtuk meg) .A for ciklusban megszoroztuk a kulccsal a buffer elemeit, majd maradékos osztást végzünk a % operátorral.A kulcs hosszúságát elérve az érték 0 lesz.

## 4.4. C EXOR törő

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

Megoldás videó:

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

```
#include <stdlib.h>
double atlagos_szohossz(const char *titkos, int titkos_meret) {
    int sz = 0;
    for(int i = 0; i < titkos_meret; ++i)
        if(titkos[i] == ' ')
            ++sz;
    return(double) titkos_meret / sz;
}
int tiszta_lehet(const char *titkos, int titkos_meret) {
    // a tiszta szöveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket
    double szohossz = atlagos_szohossz (titkos, titkos_meret);
    return szohossz > 6.0 && szohossz < 9.0
           && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
           && strcasestr (titkos, "az") && strcasestr (titkos, "ha");
}
void exor(const char kulcs[], int kulcs_meret, char titkos[],
        int titkos_meret, char *buffer) {
    int kulcs_index = 0;
    for (int i = 0; i < titkos_meret; ++i) {
        buffer[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}
void exor_tores(const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret) {
    char *buffer;
    if((buffer = (char *)malloc(sizeof(char)*titkos_meret)) == NULL) {
        printf("Memoria (buffer) faliora\n");
        exit(-1);
    }
    exor (kulcs, kulcs_meret, titkos, titkos_meret, buffer);
    if(tiszta_lehet (buffer, titkos_meret)) {
        printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
               kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],
               kulcs[6],kulcs[7], buffer);
    }
    free(buffer);
}
int main(void) {
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;
    // titkos fajt berantasa
    while((olvasott_bajtok =
               read(0, (void *) p,
                     (p - titkos + OLVASAS_BUFFER <
                      MAX_TITKOS) ? OLVASAS_BUFFER :
```

```
                        titkos + MAX_TITKOS - p)))
        p += olvasott_bajtok;
    // maradek hely nullazasa a titkos bufferben
    for(int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\0';
    // osszes kulcs eloallitasa
    #pragma omp parallel for private(kulcs)
    for(int ii = '0'; ii <= '9'; ++ii)
        for(int ji = '0'; ji <= '9'; ++ji)
            for(int ki = '0'; ki <= '9'; ++ki)
                for(int li = '0'; li <= '9'; ++li)
                    for(int mi = '0'; mi <= '9'; ++mi)
                        for(int ni = '0'; ni <= '9'; ++ni)
                            for(int oi = '0'; oi <= '9'; ++oi)
                                for(int pi = '0'; pi <= '9'; ++pi) {
                                    kulcs[0] = ii;
                                    kulcs[1] = ji;
                                    kulcs[2] = ki;
                                    kulcs[3] = li;
                                    kulcs[4] = mi;
                                    kulcs[5] = ni;
                                    kulcs[6] = oi;
                                    kulcs[7] = pi;
                                    exor_tores (kulcs, KULCS_MERET, titkos,
                                    p - titkos);
                                }
    return 0;
}
```

Az Exor törő elején definiáljuk a kulcs méretét és a titkosított szöveg maximális méretét.Megadjuk az átlagos szóhosszot majd a for ciklus végrehajtásával az sz változóhoz mindig hozzáadunk 1-et.Ha a tiszta_lehet függvényben azt kapjuk hogy a kódunk tiszta akkor a következő exor függvényel a tiszta kódot vissza is kapjuk. Az exor_tores függvény a szöveg tisztaságától függően 1-et vagy 0-át ad vissza. A main függvényben 2 tömböt deklarálunk kulcs és titkor néven. A két több mérete a korábban megadott értékeket veszi fel.A while ciklusban olvassuk a bájtokat mindaddig még a bemenet végére nem érünk vagy a Buffer el nem éri a maximális méretét. Ezután for ciklusokkal előállítjuk az összes lehetséges kulcsot alkalmazva a xor műveletet.Találat esetén a kulcs és a feltört szöveg kiíratásra kerül

## 4.5. Neurális OR, AND és EXOR kapu

R

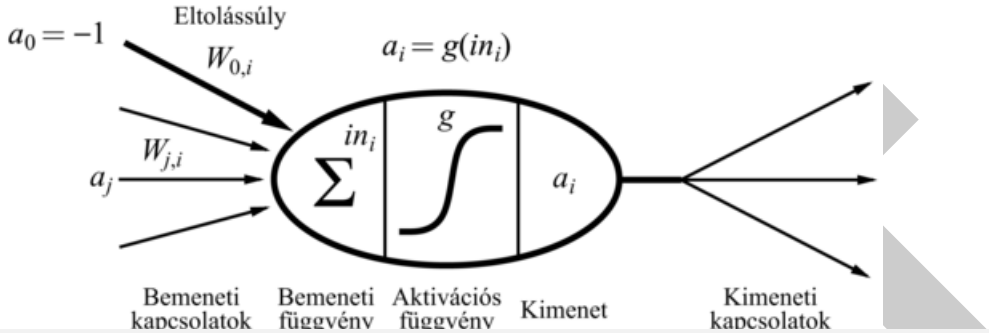
Tutor:Szűcs Gergő

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

A neurális hálóknak manapság egyre nagyobb szerepe van az informatikában ,hiszen a merterséges intelligencia erre épül.A neurális háló képes tanulni és különböző adatokat tulajdonságokat megjegyezni mellyekkel a bejövő adatokat értékeli és reagál rájuk. Az alábbi ábrán a neuron egyszerű matematikai modellje

látható. "Az mondható, hogy a neuron akkor "tüzel", amikor a bemenetek súlyozott összege meghalad egy küszöböt. A kimeneti értéket az aktivációs függvény fogja megadni."

A következő képen a neuron egyszerű matematikai modellje látható. Forrás: link




```
library(neuralnet)
a1    <- c(0,1,0,1)
a2    <- c(0,0,1,1)
OR    <- c(0,1,1,1)
or.data <- data.frame(a1, a2, OR)
nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE,  ↩
   stepmax = 1e+07, threshold = 0.000001)
plot(nn.or)
compute(nn.or, or.data[,1:2])
a1    <- c(0,1,0,1)
a2    <- c(0,0,1,1)
OR    <- c(0,1,1,1)
AND   <- c(0,0,0,1)
orand.data <- data.frame(a1, a2, OR, AND)
nn.orand <- neuralnet(OR+AND~a1+a2, orand.data, hidden=0, linear.output= ↩
   FALSE, stepmax = 1e+07, threshold = 0.000001)
plot(nn.orand)
compute(nn.orand, orand.data[,1:2])
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)
exor.data <- data.frame(a1, a2, EXOR)
nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=0, linear.output=FALSE,  ↩
   stepmax = 1e+07, threshold = 0.000001)
plot(nn.exor)
compute(nn.exor, exor.data[,1:2])
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)
exor.data <- data.frame(a1, a2, EXOR)
nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ↩
   output=FALSE, stepmax = 1e+07, threshold = 0.000001)
plot(nn.exor)
compute(nn.exor, exor.data[,1:2])
```

R nyelvben dolgozunk, így szükségünk lesz a neuralnet csomagra. Az a1-be és az a2-be értékeket teszünk majd az OR-ban meghatározzuk hogy logikai vagy esetén milyen értéket kell visszakapnunk, azaz itt tanít-

juk a programot ezután már magát fogja. A neuralnet egy súlyozást számol minden bemenethez.Ezután a súlyokat beállítja, melyeknek a helyességét a compute paranccsal ellenőrzünk.

AND használata esetén majdnem ugyanez a feladatom,de ott más lesz a logikai művelet igazságértéke.

EXOR használata esetén rejtett neuronokkal kell dolgoznunk.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Tutor:Szűcs Gergő

```
#include <iostream>
#include "mlp.hpp"
#include "png++/png.hpp"
int main (int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width()*png_image.get_height();

    Perceptron* p = new Perceptron(3, size, 256, 1);
    double* image = new double[size];

    for(int i {0}; i<png_image.get_width(); ++i)
         for(int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;
    double value = (*p) (image);
    std::cout << value << std::endl;
    delete p;
    delete [] image;
}
```

Az előbb ,megadott program inputja egy kép,melyben pixelenként vizsgálja a piros szín méretét.AZ adatok neurális hálóba kerülnek. Ezeket az adatokat véletlenszerű számokkal súlyozzuk.Ezeknek az értékeknek a vizsgálatakor kapunk egy számot -1 és 1 között súlyozástól függően.

# 5. fejezet

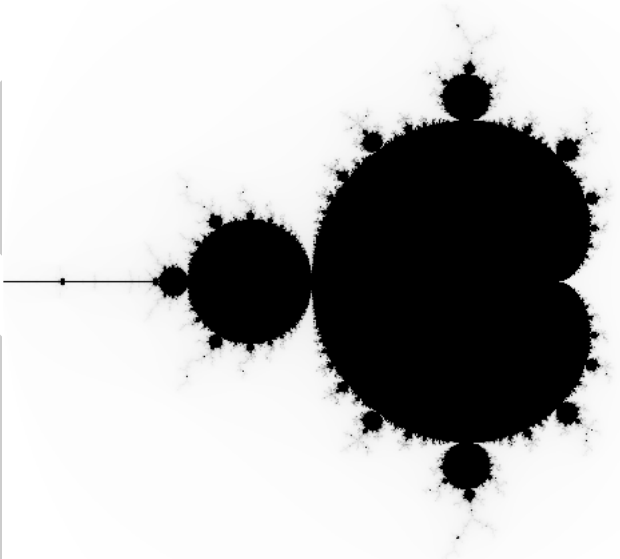
# Helló, Mandelbrot!

## 5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

A Mandelbrot halmaz a komplex számoknak egy meghatározott részhalmaza. Mandelbrot halmaz ábrázolásánál a képernyő megfelel a síknak a pixelek pedig az oda tartozó komplex számoknak.



```
#include <stdio.h>
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>
#define MERET 600
#define ITER_HAT 32000
void
mandel (int kepadat[MERET][MERET]) {
```

```
    clock_t delta = clock ();
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

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

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

        for (int k = 0; k < szelesseg; ++k)
        {
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;
                ++iteracio;
            }
            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)
    {
        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;
}
}
```

A kódunk elején definiáljuk a kép méretét és az iterációs határt.Egy megadott lépésközzel végig megyünk a rácspontokon, majd a C és a Z komplex számokat változókban tároljuk. Mivel komplex számokkal dolgozunk ezért létrejön egy koordinátarendszer és minden számnak 2 koordinátája lesz (x és y).A while ciklusban szűmoljuk a halmaz következő elemeit mindaddig, amíg a z négyzete kisebb, mint 4. Az iterációs határ elérése után az iteráció konvergens ,Ezért a c a mandelbrot halmaz eleme. Már csak meg kell adnunk a pixelek színét és ki is rajzolhatjuk az ábrát.

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

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

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

```
#include <iostream>
#include "png++/png.hpp"
#include <complex>
int
main ( int argc, char *argv[] )
{
  int szelesseg = 1920;
  int magassag = 1080;
  int iteraciosHatar = 255;
  double a = -1.9;
  double b = 0.7;
  double c = -1.3;
  double d = 1.3;
  if ( argc == 9 )
    {
      szelesseg = atoi ( argv[2] );
```

```
      magassag =  atoi ( argv[3] );
      iteraciosHatar =  atoi ( argv[4] );
      a = atof ( argv[5] );
      b = atof ( argv[6] );
      c = atof ( argv[7] );
      d = atof ( argv[8] );
    }
  else
    {
      std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↩
         " << std::endl;
      return -1;
    }
  png::image < png::rgb_pixel > kep ( szelesseg, magassag );
  double dx = ( b - a ) / szelesseg;
  double dy = ( d - c ) / magassag;
  double reC, imC, reZ, imZ;
  int iteracio = 0;
  std::cout << "Szamitas\n";
  // j megy a sorokon
  for ( int j = 0; j < magassag; ++j )
    {
      // k megy az oszlopokon
      for ( int k = 0; k < szelesseg; ++k )
        {
          // c = (reC, imC) a halo racspontjainak
          // megfelelo komplex szam
          reC = a + k * dx;
          imC = d - j * dy;
          std::complex<double> c ( reC, imC );
          std::complex<double> z_n ( 0, 0 );
          iteracio = 0;
          while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
            {
              z_n = z_n * z_n + c;
              ++iteracio;
            }
          kep.set_pixel ( k, j,
                          png::rgb_pixel ( iteracio%255, (iteracio*iteracio ↩
                             )%255, 0 ) );
        }
      int szazalek = ( double ) j / ( double ) magassag * 100.0;
      std::cout << "\r" << szazalek << "%" << std::flush;
    }
  kep.write ( argv[1] );
  std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

Mint az előző programban , itt is meg kell adnunk a kép adatait, az iterációs határt és az adatokat amelyek a számításhoz szükségesek.Ebben a kódban 2 for ciklussal járjuk be a rácspontokat, majd a kép pixeleit

beállítjuk. Figyeljünk arra hogy ne azokat az értékeket adjuk meg mint az előző feladatban, hiszen így egy színesebb képet kaphatunk.A while ciklus megmutatja hogy mennyire távolodott el egymástól a z_n és a z_0.Ha a while ciklusból való kilépés az iterációs határ elérése miatt történik, akkor az iteráció konvergens ,Ezért a c a mandelbrot halmaz eleme.

## 5.3. Biomorfok

Első gyakorlat passzolási lehetőség

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

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

"A CUDA jelentése Compute Unified Device Architecture, vagyis egy új hardver és szoftverarchitektúra amelynek célja, hogy ellássa és kezelje a GPU-n történő számításokat anélkül,hogy azokat le kellene képezni valamelyik grafikus API-ra." Forrás : https://dea.lib.unideb.hu/dea/bitstream/handle/2437/4529/Szakdolgozat.pdf;jsessionid=4DE14EBBC2F036722B7A6CA9355DD3?sequence=1

A CUDA tartalmaz egy rácsot ahol minden sor és oszlop 60 blokkot tartalmaz, 1-1 blokk pedig 100 szálat így egyszerre 60x60x100-at fog számolni. Ha például egy 600x600 pixeles képet szeretnél kiszámolni akkor minden pixelnek megfelel egy szál. A CPU-nál viszont csak egy szálunk volt

```
#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;
  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;
}
```

A fordítást más paranccsal kell elvégeznünk:

```
nvcc mandelpngc_60x60_100.cu -lpng16 -03 -o mandelpngc
```

Hogy a programot futtatni tudjuk szükségünk lesz nvidia kártyára és akár 50-70x-es gyorsulás is elérhető.

## 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 videó: Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal.

A következő kód QT GUI-t használ.Ahhoz hogy fordítani tudjunk létr kell hoznunk egy mappát és belehelyezni mind a 4 fájlt amelyre szükségünk lesz. A qmake -project parancs futtatásával létrejönegy *.pro fájl. FOntos hogy a parancsot helyes mappában futtassuk.Írjuk a fájlba a következő sort: QT += widgets.HA ez megtörtént akkor a qmake*.pro parancs futtatásával létrejön egy Makefile , ami után már létrehozható egy bináris fájl a make paranccsal.

```
// frakablak.cpp
#include "frakablak.h"
FrakAblak::FrakAblak(double a, double b, double c, double d,
                     int szelesseg, int iteraciosHatar, QWidget *parent)
                         : QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");
    szamitasFut = true;
    x = y = mx = my = 0;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    magassag = (int)(szelesseg * ((d-c)/(b-a)));
    setFixedSize(QSize(szelesseg, magassag));
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag,  ↩
       iteraciosHatar, this);
    mandelbrot->start();
}
FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}
void FrakAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);
    }
    qpainter.end();
}
void FrakAblak::mousePressEvent(QMouseEvent* event) {
    // A nagyítandó kijelölt területet bal felsõ sarka:
```

```
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;
    update();
}
void FrakAblak::mouseMoveEvent(QMouseEvent* event) {
    // A nagyítandó kijelölt terület szélessége és magassága:
    mx = event->x() - x;
    my = mx; // négyzet alakú
    update();
}
void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {
    if(szamitasFut)
        return;
    szamitasFut = true;
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double a = this->a+x*dx;
    double b = this->a+x*dx+mx*dx;
    double c = this->d-y*dy-my*dy;
    double d = this->d-y*dy;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ↩
       iteraciosHatar, this);
    mandelbrot->start();
    update();
}
void FrakAblak::keyPressEvent(QKeyEvent *event)
{
    if(szamitasFut)
        return;
    if (event->key() == Qt::Key_N)
        iteraciosHatar *= 2;
    szamitasFut = true;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ↩
       iteraciosHatar, this);
    mandelbrot->start();
}
void FrakAblak::vissza(int magassag, int *sor, int meret)
{
    for(int i=0; i<meret; ++i) {
        QRgb szin = qRgb(0, 255-sor[i], 0);
        fraktal->setPixel(i, magassag, szin);
    }
```

```
    update();
}
void FrakAblak::vissza(void)
{
    szamitasFut = false;
    x = y = mx = my = 0;
}
```

```
// frakablak.h
#ifndef FRAKABLAK_H
#define FRAKABLAK_H
#include <QMainWindow>
#include <QImage>
#include <QPainter>
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"
class FrakSzal;
class FrakAblak : public QMainWindow
{
    Q_OBJECT
public:
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
              double d = 1.35, int szelesseg = 600,
              int iteraciosHatar = 255, QWidget *parent = 0);
    ~FrakAblak();
    void vissza(int magassag , int * sor, int meret) ;
    void vissza(void) ;
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;
protected:
    void paintEvent(QPaintEvent*);
    void mousePressEvent(QMouseEvent*);
    void mouseMoveEvent(QMouseEvent*);
    void mouseReleaseEvent(QMouseEvent*);
    void keyPressEvent(QKeyEvent*);
private:
    QImage* fraktal;
    FrakSzal* mandelbrot;
    bool szamitasFut;
    // A nagyítandó kijelölt területet bal felsõ sarka.
    int x, y;
    // A nagyítandó kijelölt terület szélessége és magassága.
    int mx, my;
```

```
};
#endif // FRAKABLAK_H
```

```
// frakszal.cpp
#include "frakszal.h"
FrakSzal::FrakSzal(double a, double b, double c, double d,
                   int szelesseg, int magassag, int iteraciosHatar,  ↩
                      FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;
    egySor = new int[szelesseg];
}
FrakSzal::~FrakSzal()
{
    delete[] egySor;
}
void FrakSzal::run()
{
    // A [a,b]x[c,d] tartományon milyen sûrû a
    // megadott szélesség, magasság háló:
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság hálót:
    for(int j=0; j<magassag; ++j) {
        //sor = j;
        for(int k=0; k<szelesseg; ++k) {
            // c = (reC, imC) a háló rácspontjainak
            // megfelelõ 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;
            }
            // ha a < 4 feltétel nem teljesült és a
            // iteráció < iterációsHatár sérülésével lépett ki, azaz
            // feltesszük a c-rõl, hogy itt a z_{n+1} = z_n * z_n + c
            // sorozat konvergens, azaz iteráció = iterációsHatár
            // ekkor az iteráció %= 256 egyenlõ 255, mert az esetleges
            // nagyítasok során az iteráció = valahány * 256 + 255
            iteracio %= 256;
            //a színezést viszont már majd a FrakAblak osztályban lesz
            egySor[k] = iteracio;
        }
        // Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.
        frakAblak->vissza(j, egySor, szelesseg);
    }
    frakAblak->vissza();
}
```

```
// frakszal.h
#ifndef FRAKSZAL_H
#define FRAKSZAL_H
#include <QThread>
#include <math.h>
#include "frakablak.h"
class FrakAblak;
class FrakSzal : public QThread
{
    Q_OBJECT
public:
    FrakSzal(double a, double b, double c, double d,
             int szelesseg, int magassag, int iteraciosHatar, FrakAblak * ↩
                frakAblak);
    ~FrakSzal();
    void run();
protected:
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;
```

```
    // Kinek számolok?
    FrakAblak* frakAblak;
    // Soronként küldöm is neki vissza a kiszámoltakat.
    int* egySor;
};
#endif // FRAKSZAL_H
```

```
// main.cpp
#include <QApplication>
#include "frakablak.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    FrakAblak w1;
    w1.show();
    /*
    FrakAblak w1,
    w2(-.08292191725019529, -.082921917244591272,
       -.9662079988595939, -.9662079988551173, 600, 3000),
    w3(-.082921917248806 25, -.0829219172470933,
       -.9662079988581493, -.9662079988563615, 600, 4000),
    w4(.14388310361318304, .14388310362702217,
       .6523089200729396, .6523089200854384, 600, 38655);
    w1.show();
    w2.show();
    w3.show();
    w4.show();
*/
    return a.exec();
}
```

## 5.6. Mandelbrot nagyító és utazó Java nyelven

Második előadás passzolási lehetőség.

# 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:

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!

Az objektumorientált programozásnál elengedhetetlen a classok használata.Egy classban, másnéven osztályban,megjelennhet több függvény és több változó ia melyeket egy egységként kezelhetünk.Osztály megadásánál gyakran szerepel a private vagy a public szavak melyek szerepe meghatározni hogy az adott elemekhez hozzá férhetnek e az osztályon kívüli elemek

Java-ban:

```
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[] args) {

        PolárGenerátor g = new PolárGenerátor();

        for(int i=0; i<10; ++i)
            System.out.println(g.következő());

    }

}
```

C++-ban:

```
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iostream>
class PolarGen
{
public:
  PolarGen ()
  {
    nincsTarolt = true;
    std::srand (std::time (NULL));
  }
```

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

## 6.2. LZW

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

Az LZW egy ,veszteségmentes tömörítési algoritmus,melyeket bináris adatok tömörítésére használnak. Az algoritmus az adatokat fa struktúrában tárolja. A beérkező adatok(melyek 1-esek és 0-ák) feldolgozása a következőképpen zajlik: a gyökértől kiindulunk majd vizsgáljuk hogy van e már 1-es vag 0-ás gyermek.

Ha van addig lépkedünk amíg nem lesz, ha nincs akkor beszúrunk egyet majd visszalépünk a gyökérre és beolvassuk a következő beérkező adatot és az elejéről kezdjük a folyamatot.

Először létrehozunk egy binfa struktúrát és egy binfa típust.A struktúrában megtalálható egy érték és a fa bal valamint a jobb oldali gyermekeire mutató mutató. A binfa struktúra három részből áll: egy értékből és a fa bal és jobb gyermekeire mutató mutatókból. Létrehozunk egy mutatót, mely a binfa típusra mutat. Az uj_elem() függvény segítségével memóriaterületet foglalunk.A main függvényben létrehozzuk a gyökérmutatót melynek a '/' értéket adjuk. Binfa struktúrában a gyökérelemet általában valamiylen speciális karakterrel jelöljük.Ezután elkezdjük az adatok beolvasását.Ha az érték 0 akkor megvizsgáljuk hogy a fa bal gyermeke null értékű-e. Ha nem akkor az azt jelenti hogy a gyökérelemnek nincs bal oldali gyermeke.Ekkor a fa mutató bal gyerekének lefoglalunk egy helyet és 0-ra állítjuk az értékét.Ha a gyökér bal oldalán már van egy 0 és az inputon érkezik a következő akkor ebben az esetben ez a 0 tölti be a gyökérelem szerepét. Ugyanez a helyzet a jobb oldalon is , Ezután pedig csak kiíratjuk a fát.

```
 #include <stdio.h>
#include <stdlib.h>0
#include <unistd.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 szabadit (BINFA_PTR elem);
int
main (int argc, char **argv)
{
  char b;
  BINFA_PTR gyoker = uj_elem ();
  gyoker->ertek = '/';
  BINFA_PTR fa = gyoker;
  while (read (0, (void *) &b, 1))
    {
      write (1, &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;
  printf ("melyseg=%d", max_melyseg);
  szabadit (gyoker);
}
static int melyseg = 0;
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);
      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!

Egy fa struktúrának három bejárási módja van, inorder, preorder és postorder.

Minden bejárás előtt meg kell vizsgálnunk hogy a bejárandó fa tartalmaz e elemeket vagy egy üres fáról van szó. Preorder bejárásnál először a gyökérelemet dolgozzuk fel,ezután bejárjuk a bal oldali részfáját, és ezután a jobb oldali részfáját is.

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

Postorder bejárásnál pedig a gyökérelem bal oldali részfát járjuk be először, utána a jobb oldali részfát és ezután dolgozzuk fel a gyökérelemet.

```
void
kiir (BINFA_PTR elem)
```

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

## 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:

A feladat az hogy ez előző feladatban C-ben megírt binfát írjuk meg C++-ban is ISmét egy class al kezdünk. Amikor elkezdjük bekérni az elemeket akkor C++-ban a new-val tudunk helyet foglalni a beilleszteni kívánt elemnek. A C++ struktúrában 2 kiir() függvényünk is van. ezt azért tehetjük meg mert a c++ már paraméterlista alapján is meg tud különböztetni 2 azonos nevű függvényt.Használni fogjuk a get függvényt is a mélység, a szórás és az átlag kiszámítására.

Az LZWBinFa class-on kívül definiáljuk az osztályon belüli függvényeket. A class belsejében lévő függvényeket a LZWBinFa:: előtaggal érjük el. A get függvényekkel a private részben lévő elemeket tudjuk elérni.

```
#include <iostream>
#include <cmath>
#include <fstream>
class LZWBinFa {
public:
    LZWBinFa():fa (&gyoker) {
    }
    ~LZWBinFa() {
        szabadit(gyoker.egyesGyermek());
        szabadit(gyoker.nullasGyermek());
    }
    void operator<< (char b) {
        if (b == '0') {
```

```
            if (!fa->nullasGyermek()){
                Csomopont *uj = new Csomopont('0');
                fa->ujNullasGyermek(uj);
                fa = &gyoker;
            }
            else {
                fa = fa->nullasGyermek ();
            }
        }
        else {
            if (!fa->egyesGyermek ()) {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else {
                fa = fa->egyesGyermek ();
            }
        }
    }
    void kiir(void) {
        melyseg = 0;
        kiir(&gyoker, std::cout);
    }
    int getMelyseg(void);
    double getAtlag(void);
    double getSzoras(void);
    friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf) {
        bf.kiir (os);
        return os;
    }
    void kiir(std::ostream & os) {
        melyseg = 0;
        kiir (&gyoker, os);
    }
private:
    class Csomopont {
    public:
        Csomopont(char b = '/'):betu (b), balNulla (0), jobbEgy (0) {
        };
        ~Csomopont() {
        };
        Csomopont *nullasGyermek() const {
            return balNulla;
        }
        Csomopont *egyesGyermek() const {
            return jobbEgy;
        }
        void ujNullasGyermek(Csomopont * gy) {
            balNulla = gy;
```

```
        }
        void ujEgyesGyermek(Csomopont * gy) {
            jobbEgy = gy;
        }
        char getBetu() const {
            return betu;
        }
    private:
        char betu;
        Csomopont *balNulla;
        Csomopont *jobbEgy;
        Csomopont (const Csomopont &);
        Csomopont & operator= (const Csomopont &);
    };
    Csomopont *fa;
    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;
    LZWBinFa (const LZWBinFa &);
    LZWBinFa & operator = (const LZWBinFa &);
    void kiir(Csomopont * elem, std::ostream & os) {
        if (elem != NULL) {
            ++melyseg;
            kiir (elem->egyesGyermek (), os);
            for (int i = 0; i < melyseg; ++i)
                os << "---";
            os << elem->getBetu () << "(" << melyseg - 1 << ")" << std:: ↩
               endl;
            kiir (elem->nullasGyermek (), os);
            --melyseg;
        }
    }
    void szabadit(Csomopont * elem) {
        if (elem != NULL) {
            szabadit(elem->egyesGyermek ());
            szabadit(elem->nullasGyermek ());
            delete elem;
        }
    }
protected:
    Csomopont gyoker;
    int maxMelyseg;
    double atlag, szoras;
    void rmelyseg(Csomopont * elem);
    void ratlag(Csomopont * elem);
    void rszoras(Csomopont * elem);
};
int LZWBinFa::getMelyseg (void) {
    melyseg = maxMelyseg = 0;
    rmelyseg(&gyoker);
    return maxMelyseg - 1;
```

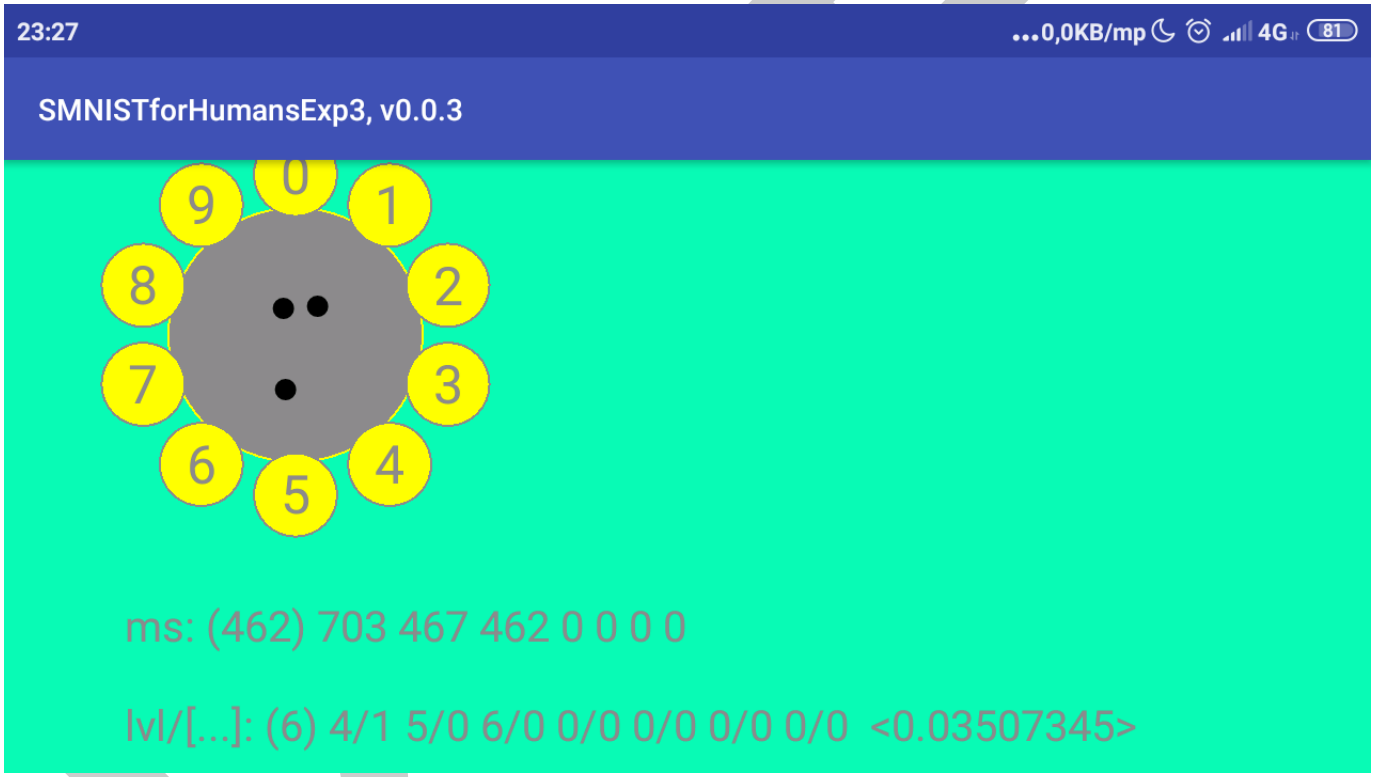
```
}
double LZWBinFa::getAtlag(void) {
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBinFa::getSzoras (void) {
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;
    rszoras (&gyoker);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

```
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}
void usage(void) {
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}
int main (int argc, char *argv[]) {
    if (argc != 4) {
        usage ();
        return -1;
    }
    char *inFile = *++argv;
    if (*((*++argv) + 1) != 'o') {
        usage ();
        return -2;
    }
    std::fstream beFile (inFile, std::ios_base::in);
    if (!beFile) {
        std::cout << inFile << " nem letezik..." << std::endl;
        usage ();
        return -3;
    }
    std::fstream kiFile (*++argv, std::ios_base::out);
    unsigned char b;
    LZWBinFa binFa;
    while (beFile.read ((char *) &b, sizeof (unsigned char)))
        if (b == 0x0a)
            break;
    bool kommentben = false;
    while (beFile.read ((char *) &b, sizeof (unsigned char))) {
        if (b == 0x3e) {
            kommentben = true;
            continue;
        }
        if (b == 0x0a) {
            kommentben = false;
            continue;
        }
        if (kommentben)
            continue;
        if (b == 0x4e)
            continue;
        for (int i = 0; i < 8; ++i) {
            if (b & 0x80)
                binFa << '1';
            else
                binFa << '0';
            b <<= 1;
        }
```

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




## 6.6. Mozgató szemantika

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

Második labor passzolási lehetőség

# 7. fejezet

# Helló, Conway!

## 7.1. Hangyaszimulációk

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

3. labor passzolási lehetőség

## 7.2. Java életjáték

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

A John Horton Conway féle életjáték szabályai a következők : A sejtek cellákban élnek(crllánként 1 sejt).Egy sejt akkor él ha van 2 vagy 3 szomszédja , különben meghal. ha meghalt egy sejt addig halott is marad amig pont 3 szomszédja van.

```
public void időFejlődés() {

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

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

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

                if(rácsElőtte[i][j] == ÉLŐ) {
                /* Élő élő marad, ha kettő vagy három élő
                 szomszedja van, különben halott lesz. */
                    if(élők==2 || élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
```

```
                } else {
                /* Halott halott marad, ha három élő
                 szomszedja van, különben élő lesz. */
                    if(élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
                }
            }
        }
        rácsIndex = (rácsIndex+1)%2;
    }
}
```

A sejtek összessége a sikló. Adott irányba halad,és lemásolja önmagát.

```
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;

    }
```

A sikló ágyú bizonyos iránybalövöldözi a többi siklót. Az x és y érték a bal felso sarkának határoló élei.

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

    }
```

Fordításkor,java fordítót használunk.

Ha megnyomjuk az S-t akkor készül egy felvétel a sejttéről.N-el mő, K-val pedig csökken a sejtméret.A G gyorsít az i pedig lassít.

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

A szabály ugyanúgy működik mint javaban:

```
void SejtSzal::idoFejlodes() {
    bool **racsElotte = racsok[racsIndex];
    bool **racsUtana = racsok[(racsIndex+1)%2];
    for(int i=0; i<magassag; ++i) { // sorok
        for(int j=0; j<szelesseg; ++j) { // oszlopok
            int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);
            if(racsElotte[i][j] == SejtAblak::ELO) {

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

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

Sikló:

```
void SejtAblak::siklo(bool **racs, int x, int y) {

  racs[y+ 0][x+ 2] = ELO;
  racs[y+ 1][x+ 1] = ELO;
  racs[y+ 2][x+ 1] = ELO;
  racs[y+ 2][x+ 2] = ELO;
  racs[y+ 2][x+ 3] = ELO;

}
```

Siklókilövő

```
void SejtAblak::sikloKilovo(bool **racs, int x, int y) {

  racs[y+ 6][x+ 0] = ELO;
  racs[y+ 6][x+ 1] = ELO;
  racs[y+ 7][x+ 0] = ELO;
  racs[y+ 7][x+ 1] = ELO;
```

```
  racs[y+ 3][x+ 13] = ELO;

  racs[y+ 4][x+ 12] = ELO;
  racs[y+ 4][x+ 14] = ELO;

  racs[y+ 5][x+ 11] = ELO;
  racs[y+ 5][x+ 15] = ELO;
  racs[y+ 5][x+ 16] = ELO;
  racs[y+ 5][x+ 25] = ELO;

  racs[y+ 6][x+ 11] = ELO;
  racs[y+ 6][x+ 15] = ELO;
  racs[y+ 6][x+ 16] = ELO;
  racs[y+ 6][x+ 22] = ELO;
  racs[y+ 6][x+ 23] = ELO;
  racs[y+ 6][x+ 24] = ELO;
  racs[y+ 6][x+ 25] = ELO;

  racs[y+ 7][x+ 11] = ELO;
  racs[y+ 7][x+ 15] = ELO;
  racs[y+ 7][x+ 16] = ELO;
  racs[y+ 7][x+ 21] = ELO;
  racs[y+ 7][x+ 22] = ELO;
  racs[y+ 7][x+ 23] = ELO;
  racs[y+ 7][x+ 24] = ELO;

  racs[y+ 8][x+ 12] = ELO;
  racs[y+ 8][x+ 14] = ELO;
  racs[y+ 8][x+ 21] = ELO;
  racs[y+ 8][x+ 24] = ELO;
  racs[y+ 8][x+ 34] = ELO;
  racs[y+ 8][x+ 35] = ELO;

  racs[y+ 9][x+ 13] = ELO;
  racs[y+ 9][x+ 21] = ELO;
  racs[y+ 9][x+ 22] = ELO;
  racs[y+ 9][x+ 23] = ELO;
  racs[y+ 9][x+ 24] = ELO;
  racs[y+ 9][x+ 34] = ELO;
  racs[y+ 9][x+ 35] = ELO;

  racs[y+ 10][x+ 22] = ELO;
  racs[y+ 10][x+ 23] = ELO;
  racs[y+ 10][x+ 24] = ELO;
  racs[y+ 10][x+ 25] = ELO;

  racs[y+ 11][x+ 25] = ELO;

}
```

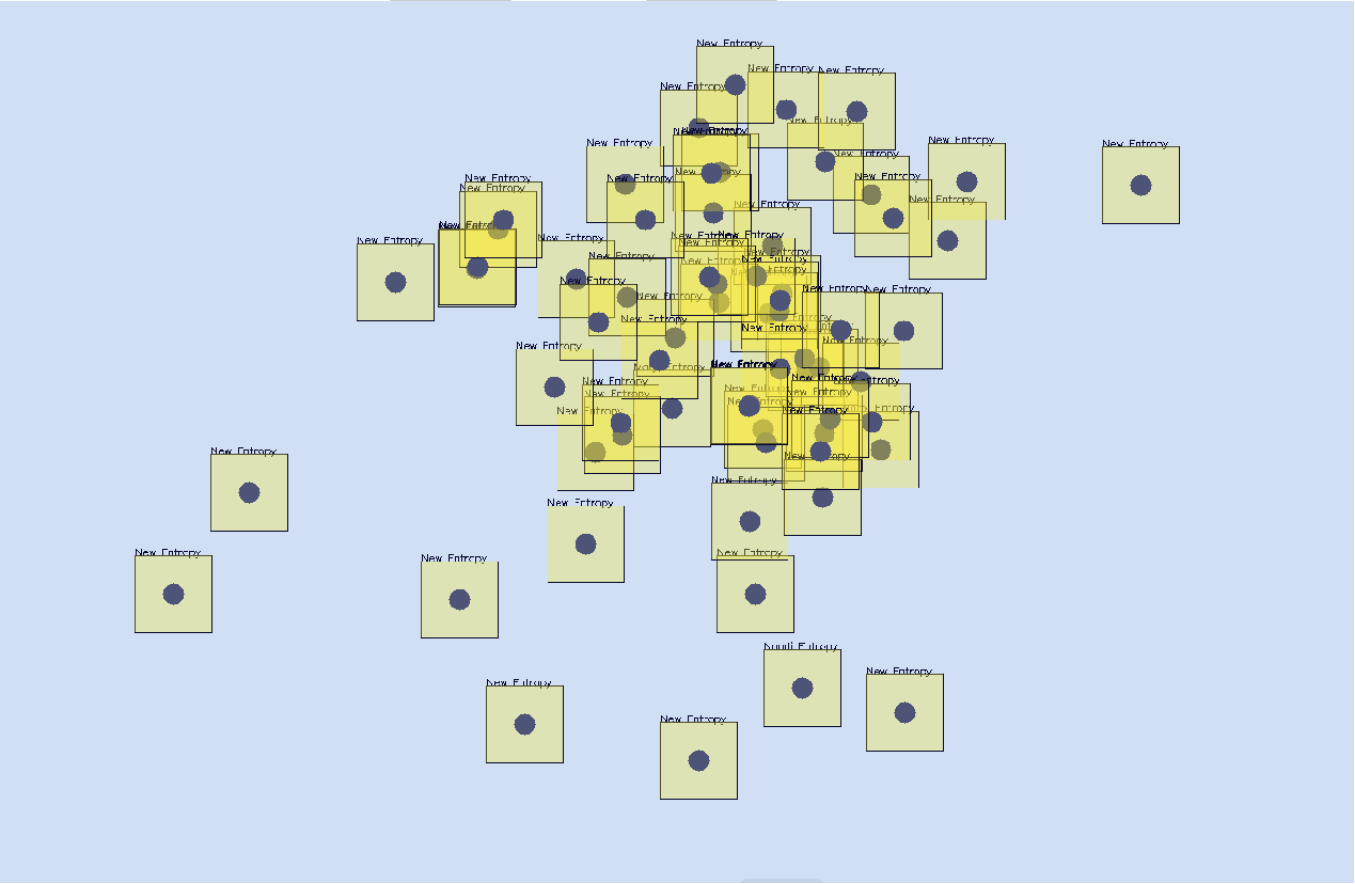
```
#include <QApplication>
#include "sejtablak.h"
#include <QDesktopWidget>
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  SejtAblak w(100, 75);
  w.show();

  return a.exec();
}
```

C++-nál használmi kell a QT-t, ezért fordításkor először Qmake -project , ez majd legenerálja a -pro fájlt és ezután még alkalmaznom kell egy make-et. A futtatás menete ugyanaz mit Javaban

## 7.4. BrainB Benchmark

A program azért íródott hogy a karakterek mozgása követését figyeljék meg e sportolókon. A Samu Entropy dobozon belül, a mozgó körben kell tartani az egérmutatót és eközben újabb dobozok is megjelennek hogy összezavarjanak. Ha 1 másodpercet meghaladja az az idő amikor az egérmutató a körön kívül van akkor romlik az eredmény és a dooboz mozgása lassul.

Fordításkor az OpenCv könyvtárat használjuk.



A mérés 10 percig tart és statisztika is készül a teljesítményről, amit a program könyvtárában egy txt fájlban találunk.

# 8. fejezet

# Helló, Schwarzenegger!

## 8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.5. Minecraft-MALMÖ

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# 9. fejezet

# Helló, Chaitin!

## 9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

## 9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

## 9.3. 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

Tanulságok, tapasztalatok, magyarázat...

## 9.4. 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

Tanulságok, tapasztalatok, magyarázat...

## 9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9.6. Omega

Megoldás videó:

Megoldás forrása:

# 10. fejezet

# Helló, Gutenberg!

## 10.1. Programozási alapfogalmak

Juhász István - Magas szintű programozási nyelvek 1

Az informatikában léteznek különböző dekleratív és imperatív nyelvek. Az imperatív nyelvekben a legfontosabb a változó melyeket különböző algoritmusokban használunk fel.Nagyon meghatározó szempontjukból a Neumann-architektúra. A deklaratív nyelveknek nincs igazán meghatározott közös jellemzőjük,de nem algoritmikusak.

Karakterkészlet

A programok mindig karakterekből állnak ezért érdemes ha ismerjük a karakterkészletünket, azaz hogy milyen karakterekkel dolgozhatunk.A karakterek 3 fő csoportja a betűk a számok és az egyéb karakterek.

Adattípusok

Minden adattípusnak kell rendelkezni névvel és azonosítóval, hogy el lehessen róla mondani hogy egy adattípus.VAnnak programnyelvek ,amelyek ismerik a típusokat és vannak amelyek nem ,így különböztetünk meg típusos és nem típusos nyelveket. Vannak olyan nyelvek melyek engedélyezik a saját adattípus megadását, ekkor meg kell adnunk a tartományát a műveleteit és a reprezentációját.

Kifejezések

A kifejezések olyan szintaktikai eszközök amelyek operandusokból operátorokból és kerek zárójelekből állnak. Az operandus lehet literál, konstans, változó vagy függvényhívás. Az operátorok műveleti jelek. A kerek zárójelek a műveletek végrehajtási sorrendjét befojásolják, rendundásan alkalmazhatóak.

Utasítások

Az utasítások segítségével meg tudjuk adni az alggoritmusk egyes részeit és még a tárgyprogramok is ezek segítségével készülnek. Rengeteg utasítás létezik melyeket 9 nagyobb csoportra osztunk ( értékadó utasítás,üres, ugró, elágazó, ciklusszervező, hívó, vezérlésátadó,I/O utasítások és más utasítások)

A programok szerkezete

Egy program szövege programegységekből áll. Van olyan nyelv hogy egy programot programegységenként kell fordítanunk de vn olyan is amelyben magá a kész programot egyben kell fordítanunk Vannak különböző alprogramuk is .A leghasznosabbnak és a leggyakrabban alkalmazottnak akkor bizonyulnak amikor egy egy hosszabb programban egy adott programrész többször ismétlődik. Ha alprogramban írjuk ezt a részt elég

egyszer megírnunk és nem kell többször a kódba gépelni. Az alprogramnak két fajtája van: eljárás és függvény. Az eljárássaleredményt kapunk amelyet fel tudunk használni,de a függvény csak egy értéket ad vissza.

A folyamatot mikor az alprogram hívásánál egymáshoz rendelődnek a formális és aktuális paraméterek paraméterkiértékelésnek nevezzük. A blokk olyan programegység amely egy másik programegységbe van beágyazva. A blokknak nincs paramétere és bárhol elhelyezhető.

## 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Vezérlési szerkezetek

Egy kifejezés akkor válik utasítássá ha pontosvessző követi.A Kerighanritchie könyv vezérlési szerkezetek fejezete föként utasítás elemzéssel és értelmezéssel foglalkozik. Az if-else utasítást valamilyen választás vagy döntés leírására használhatjuk.Van amikor ez az utasítás az else ág nélkül is helyesen működik,de erre oda kell figyelnünk mert hiányzó else ág esetén nem tudni hogy a meglévő else ág melyik ifhez tartozik.Ennek párja az else-if, mellyel többszörös választást, döntédt írhatunk le. A switch utasítás is a többirányú programelágaztatás egyik eszköze, mely egy kifejezés értékét hasonlitja több állandó értékű kifejezéshez.Ha ki szeretnénk lépni a switch utasításból csak alkalmaznunk kell a break utasítást. A while és for utasításokat ciklusokban használjuk. A while működési elve hogy egy kifejezést ciklikusan addig értékel ki amíg az nullává nem válik.A for utasítás is hasonlóan működik,de itt nem 1 hanem 3 kifejezésünk van. Az első és a harmadik általában függvényhívás és a második egy relációs kifejezés. Ha a kifejezéseket elhagyjuk és csak a pontos vesszőket írjuk ki a for ciklusban egy végtelen ciklust hozunk létre. A do-while utasításban a kifejezés kiértékelése a ciklustörzs végrehajtása után történik,tehát egyszer biztod végrehatódik a törzs. A break utasításról már volt szó, a legfontosabb jellemzője ,hogy lehetővé teszi hogy egy ciklust a befejezése előtt elhagyjunk. A goto utasítás egy megadott címre ugorhatunk.Leggyakoribb használata az anikor egymásba épülő ciklusok vannak a kódban és valamelyik ciklust ki szeretném hagyni de nem akarom törölni.

## 10.3. Programozás

[BMECPP]

A C++ nem objektumorientált tulajdonságai

A C nyelvet továbbfejlesztették annak érdekében, hogy kényelmesebb és biztonságosabb legyen a használata és így jött létre a C++ programozási nyelv.Az első fontos különbség a paraméterek értelmezése C++ban a void függvény üres paraméterlistát jelent még a C nyelvben ez azt jelentené hogy a függvény nem rendelkezik paraméterrel.A C++ -ban bevezették a bool típust ami nagyon hasznos tud lenni kiértékelések esetén. A C-ben ez a típus nem létezik. C nyelvben egy függvényt csaupán a neve azonosítja mé C++ ban a neve és az argumentumlistája, ezért C++-ban már létrehozható több ugyanolyan nevű függvény is. Ez a funkció hosszabb kódoknál nagyon hasznos lehet.C++ban a A címszerinti paraméterátadás abban változott hogy a paraméter deklarációjában csak egy jelet kell írnunk a paraméter elé.

# 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, Berners-Lee!

## 11.1. 0. hét: Az objektumorientált paradigma alapfoglamai. Osztály,objektum, példányosítás.

A Python egy objektumorientált,dinamikus, platformfüggetlen programozási nyelv amelyet még a 90-es években alkottak meg. Egy olyan szkriptnyelv, amelyet a kiegészítő csomagok és eljárások lehetővé tesznek bonyolultabb programok megírásához is.

Mivel egyszerű ,de mégis megbízható ezért ezt használva ajánlott prototípus-alkalmazások és különböző tesztelések megírása.

Ezek mellett a Pythonra magas szintű programozási nyelvként tekintünk. A Python nyelv legfőbb jellemzője hogy behúzásalapú a szintaxisa. A programban szerplő állításokat az azonos szintű behúzásokkal tudjuk csoportba rendezni, nincs szükség kapcsos zárójelekre vagy explicit kulcsszavakra.Fontos hogy a behúzásokat egységesen kezeljük,tehát vagy mindenhol tabot vagy mindenhol szóközt használjunk. Továbbá fontos megjegyezni hogy egy utasítás mindig a sor végéig tart,nincs ';' a sorok végén. Ha egy utasítás nem fér el egy sorban akkor '\'-jelet teszünk a sor végére.

Az értelemző minden sort tokenekre bont melyek lehetnek : azonosító, kulcsszó, operátor,delimiter és literál.

Pythonban minden adatot objektumok reprezentálnak.Különböző adattípusok vannak, melyek a következők: számok (egészek, lebegőpontoska,komplexek), sztringek (Ezeket idézőjelek és aposztrófok közt is megadhatjuk vagy az "u" betű használatával Unicode szöveget is felvehetünk), ennesek (zárójellel közé zárt objektumok gyűjteménye vesszővel elválasztva, listák(elemek rendezett szekvenciája, elemeit szögletes zárójelek közé zárjuk), szótár( kulcsokkal azonosított elemek rendezetlen halmaza)

Pythonban a változók az objektumokra mutató referenciákat jelentik. Maguknak a változóknak nincsenek típusai, így a script futása során bármely akár különdöző típusú objektumra is hivatkozhatnak. A változónak értéket a "=" -lel adunk. A Python nyelvben léteznek lokális és globális változók.Függvényben felvett változó alapértelmezetten lokális lesz. Ha globálissá szeretnénk tenni akkor a függvény elején kell fellvennünk és elé kell írnunk a global kulcsszót.

A Javaban automatikus garbage collector van.A garbage collector segít a nem használt memória kezelésében, és abban, hogy ne kelljen a memórialyukakat "tömködnünk", hiszen ez automatikusan megtörténik.

Pythonban függvényeket a del szóval definiálunk. A függvények rendelkeznek paraméterekkel melyek érték szerint adhatók át kivéve a mutable típusok.

A python nyelvben is megjelenik a kivételkezelés. Ekkor a try kulcsszó után található a kódblokk melyben a kivételes eset előállhat.

Az elméleti bemutatás után a könyvben különböző példákat találhatunk.

## 11.2. C++ vs Java

A következő feladatban a Java és C++ programozási nyelvek közötti fő különbségeket és hasonlóságokat vizsgálom.Az összehasonlításhoz Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak két kötetét illetve Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven c. könyvét használom.

Bjarne Stroustrup kezdte el a C++ programozási nyelv fejlesztését a C programozási nyelv kiterjesztéseként, más nyelvekből véve át megoldásokat. A nyelv használatára 1983-ban került sor, 1991 pedig az ISO szabványosítási kezdeményezés részévé vált. A C++ programozási nyelv szabványát 1998-ban hagyták jóvá. A Java egy ennél újabb objektumorientált programozási nyelv, amelyet a Sun Microsystems fejlesztett a ’90-es évek elejétől kezdve egészen 2009-ig, amikor a céget felvásárolta az Oracle. 2011-ben a Java 1.7-es verzióját az új tulajdonos adta ki.

A hordozhatósághoz való hozzáállása jelentősen eltér a nyelveknek. A C++ forrásszinten hordozható az eltérő platformok között, ami azt jelenti hogy magát a kódot könnyen hordozhatjuk de annak szüksége lesz a új helyén egy fordításra. Tehát a lefordított, végrehajtható fájl nem hordozható. A Javaban él a platformfüggetlenség(hordozhatóság). Ez a tulajdonság azt jelenti, hogy a Java-ban írt programok a legtöbb hardveren nagyon hasonlóan futnak.A Java fordítóprogram a forráskódot csak egy úgynevezett Java bájtkódra fordítja le. Ez azután a virtuális gépen fut, ami az illető hardver gépi kódjára fordítja.De ez nem minden fordítóprogram esetében van így. Vannak olyan Java fordítóprogramok, amelyek a forráskódot natív gépi kódra fordítják le, (GCJ) ezzel valamelyest felgyorsítva annak futtatását. Cserébe a lefordított program elveszíti hordozhatóságát. Tehát a Java egyszerűbb hordozhatóságot kínál, amelyet egy szabványosított Java virtuális gép használatával tesz.

!!!!!!!!!!!!!!!!A memóriát is máshogy kezelik. A C++-ban háromféle memóriakezelés folyik: statikus, automatikus, és dinamikus. A Javaban minden memóriaelérés közvetetten történik, hivatkozásokon keresztül.

A Java egy objektumorientált nyelv. Az objektumorientált programozás (angolul object-oriented programming, röviden OOP) egy programozási módszertan.Itt nem a műveletek megalkotása áll a középpontban, hanem az egymással kapcsolatban álló programegységek megtervezése. Az objektumorientált gondolkodásmód lényegében a valós világ lemodellezésén alapul – például egy hétköznapi fogalom, a „kutya” felfogható egy osztály (a kutyák osztálya) tagjaként, annak egyik objektumaként. Minden kutya objektum rendelkezik a kutyákra jellemző tulajdonságokkal (például szőrszín, méret stb.) és cselekvési képességekkel (például futás, ugatás). Az objektumorientált programozásban fontos szerep jut az úgynevezett öröklődésnek, ami az osztályok egymásból való származtatását teszi lehetővé: a kutyák osztálya származhat az állatok osztályából, így megörökli az állatok tulajdonságait és képességeit, valamint kibővítheti vagy felülírhatja azokat a kutyák tulajdonságaival, képességeivel. Fontos, hogy a C++ alapvetően nem egy objektum orientált nyelv. Bár számos nyelvi eszköz támogatja az objektum orientált stílusú programozást, de kíválóan alkalmas más paradigmák használatára is. A funkcionális programozástól a generatív programozáson át a deklaratív stílusig sok paradigmát támogat. A nyelv nem próbál ráerőltetni egy megközelítést a programozóra, ellenben próbál minél gazdagabb eszköztárat biztosítani, hogy a megfelelő problémát a megfelelő megközelítéssel lehessen megoldani. Még akkor is, ha ez a különböző paradigmák keverését vonja maga után. Ezért ezt a nyelvet gyakran multiparadigmás programozási nyelvnek szokták besorolni.

C++-nál a dinamikus memóriahasználat alapgondolata, hogy adataink számára akkor foglalunk helyet, amikor szükség van rá, ha pedig feleslegessé válnak, azonnal felszabadítjuk a memóriaterületet. A C++-ban különösen fontos szerepe van a memóriafelszabadításnak. A felszabadítatlan memória „memóriaszivárgáshoz” (memory leak) vezethet.Itt ehetőség van használni a C nyelv malloc és free utasítását, de ajánlott az újakat alkalmazni. A new operátor az operandusban megadott típusnak megfelelő méretű területet foglal a memóriában, és arra mutató pointert ad vissza. A delete felszabadítja a new által lefoglalt területet. Javaban nem explicit kell memóriát lefoglalnunk, és felszabadítanunk, hanem a virtuális gép megteszi ezt helyettünk. Pontosabban a szemétgyűjtő mechanizmus (garbage collector - GC), melynek feladata a nem használt objektumok eltakarítása a memóriaterületről. Így sokkal kisebb a hibalehetőség, cserébe egy automatizmus szabadítja fel a memóriát, aminek külön erőforrásra van szüksége, ami magas terhelés esetén akár az alkalmazás teljesítményére is hatással lehet. A hibajelenség, amitől megszabadulunk, az a memóriaszivárgás (memory leak). Ez gyakorlatilag akkor történik, mikor már nincs szükségünk egy objektumra, nincs rá referencia, de a memóriaterületet nem szabadítottuk fel. Szerencsére a szemétgyűjtő mechanizmus megteszi ezt helyettünk.

A Java nyelvi elemeinek szintaxisa a C programozási nyelvek szintaxisát követi,- mint amúgy az ismertebb programozási nyelvek nagyrésze-, ennek eredménye, hogy a C++-ról áttért programozónak kevés dolga lesz a nyelvtana tanulásával, természetesnek fog érződni. A Java szintaxisa ráadásul beszédes is, az új funkcionalitások elsajátítása ennek köszönhetően nagyon gyorsan történik. Egy C++ forráskódot probléma nélkül, rövid időn belül át tudunk írni Java nyelvre.

A másik nagy különbség a két nyelv között az a karakterkészlet, amíg a C++ ASCII karakterkészletet használ addig a Java egy kibővített karakterkészletet használ, ami már például a görög és a magyar betűket is képes értelmezni. Ez azért van mert 8 bájt helyett már 16 bájtos karaktereket kezel.

Fontos különbség, hogy a Javaban nincsenek külön objektumok és a címükre mutató pointerek, hanem mindig referenciákon keresztül érjük el őket.

C++-ban Az osztálydeklaráció két részből áll. Az osztály feje a class/struct alapszó után az osztály nevét tartalmazza. Ezt követi az osztály törzse, amelyet kapcsos zárójelek fognak közre, és pontosvessző zár. A deklaráció az adattagokon és tagfüggvényeken kívül, a tagokhoz való hozzáférést szabályzó, később tárgyalásra kerülő public (nyilvános, publikált), private (saját, rejtett, privát) és protected (védett) kulcsszavakat is tartalmaz, kettősponttal zárva.A Java kódban nem szükséges megadni a záró karaktert, a driver automatikusan a mondat végére teszi a megfelelő jelet.

A main függvényben is van egy kis különbség, a C-ben parancssori argumentumhoz az argv tömböt, ami az argumentumokat magukat, és az argc számot is meg kell adnunk, ami az argumentumok számát tartalmazza, míg a Java-ban csak az argv-re van szükségünk, hiszen minden tömb "tudja", milyen hosszú.

# 12. fejezet

# Helló, Arroway!

## 12.1. 1.hét

Az objektumorientált paradigma: Az OOP, azaz objektum-orientált programozás, olyan zárt programegységek (objektumok) halmazából építi fel a program egészét, amelyek egyedi tulajdonságokkal rendelkeznek és önmagukban is működőképesek. Legfontosabb alapelvek: egységbezárás, adatrejtés, öröklés, absztrakció, polimorfizmus. Egységbe zárás során az osztályok állapotai és viselkedései egy helyen, adatstruktúraként egy egységben jelenik meg. Az adatrejtés során a jellemzőket és a metódusokat el tudjuk rejteni más objektumok elől, pl. egy private vagy protected állapotot létrehozva. Az öröklés során a szülőosztály jellemzői és viselkedése öröklődik az alosztályra és egy új speciális változat jön létre.

A következő kód Java-ban íródott.

```
public class PolarGenerator {

 boolean nincsTárolt = true;
 double tárolt;

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

Javaban mindig osztályokkal dolgozunk. Először létrehozzuk a PolarGenerator nevű osztályt.A tároláshoz boolean típusú változót használunk, ami jelzi, hogy van-e már kiszámolt véletlenszám és egy double változót ami az értéket tárolja.

Ezután a következő nevű metódusban történnek a matematikai számítások.Ha nincs tárolt számunk akkor belépünk az if-be és deklaráljuk a változókat. Az u1 és u2 random generált számok lesznek a v1, v2 és w pedig különböző matematikai műveletek alapján lesznek meghatározva.Ez addig folytatódik amíg a w nagyobb mint 1.

A main metódusban a new operátorral területet foglalunk le, majd g-vel visszatérünk.Ezután egy for ciklust hajtunk végre, amely 10x hajtódik végre és kiírja a generált számokat.

## 12.2. Homokózó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden máris működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen).

Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját! 1

```
public class LZWBinFa {

  public LZWBinFa() {

    fa = gyoker;
  }
public void egyBitFeldolg(char b) {
  if (b == '0') {
```

```
    if (fa.nullasGyermek() == null)
    {
      Csomopont uj = new Csomopont('0');
      fa.ujNullasGyermek(uj);
      fa = gyoker;
    }
       else
      {
      fa = fa.nullasGyermek();
      }
    }
    else {
      if (fa.egyesGyermek() == null) {
        Csomopont uj = new Csomopont('1');
        fa.ujEgyesGyermek(uj);
        fa = gyoker;
      } else {
        fa = fa.egyesGyermek();
    }
  }
}

public void kiir() {

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

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

class Csomopont {

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

public Csomopont nullasGyermek() {
  return balNulla;
}

public Csomopont egyesGyermek() {
  return jobbEgy;
}

public void ujNullasGyermek(Csomopont gy) {
```

```
  balNulla = gy;
}

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

public char getBetu() {
  return betu;
}

  private char betu;
  private Csomopont balNulla = null;
  private Csomopont jobbEgy = null;
};
private Csomopont fa = null;

private int melyseg, atlagosszeg, atlagdb;
private double szorasosszeg;
public void kiir(Csomopont elem, java.io.PrintWriter os) {

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

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

public int getMelyseg() {
  melyseg = maxMelyseg = 0;
  rmelyseg(gyoker);
  return maxMelyseg - 1;
}

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

```
  return atlag;
}

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

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

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

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

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

```
      szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
    }
  }
}

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

public static void main(String args[]) {

  if (args.length != 3) {
    usage();
    System.exit(-1);
  }
  String inFile = args[0];

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

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

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

    byte[] b = new byte[1];

    LZWBinFa binFa = new LZWBinFa();

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

    boolean kommentben = false;

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

      if (b[0] == 0x3e) {
        kommentben = true;
        continue;
      }
      if (b[0] == 0x0a) {
```

```
        kommentben = false;
        continue;
      }
      if (kommentben) {
        continue;
      }
      if (b[0] == 0x4e) // N betű
      {
        continue;
      }
      for (int i = 0; i < 8; ++i) {
        if ((b[0] & 0x80) != 0)
        {
          binFa.egyBitFeldolg('1');
        }else
        {
          binFa.egyBitFeldolg('0');
        }
        b[0] <<= 1;
      }
    }
    binFa.kiir(kiFile);

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

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

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

A fent leírt Java kód hasonló a c++-ban íródotthoz,de a szintaxisa más. Mivel a Java nyelvben minden objektum referencián keresztül érhető el, ezért a Java kódból hiányoznak a pointereket és referenciákat jelző *-ok és az 'és' jelek, így egy kcsiivel egyszerűbbnek mondható.

Különbség még, hogy Javaban függvényeket használunk ,míg C++-ban operátorokat. Nincs szükségünk a C++-ban használt helyfelszabadítás függvényre sem. Java-nál van automatikus garbage collector , c++ -nál a programozónak kell megírnia a destruktorokat/felszabadítani a memóriát.

A feladat következő része egy servletbe rakás, és utána HTTP GET kérés segítségével való mintakérés lesz, viszont ebben a részben elakadtam, ezt a jövőben fogom tudni pótolni.

## 12.3. „Gagyi”

Az ismert formális[2] „while (x <= t && x >= t && t != x);” tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására 3 , hogy a 128-nál inkluzív objektum példányokat poolozza!

```
public class Gagyi2

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

Mivel az előző kódban 2 példányt hoztunk létre ugyanazzal az értékekkel ezért egy végtelen ciklus fog lefutni.Más-más referenciával térnek vissza,így az x!=t is teljesül, de ha a -129 helyett 77-et írunk,akkor a végtelen ciklus megszűnik ugyanis,az Integer értéktartományán [-128;127] belül választunk értéket, így már van egy létező objektumunk és mi erre kapunk egy-egy referenciát, azaz x!=t megbukik.

## 12.4. Yoda

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

A Yoda conditions olyan feltételeket jelent amelyeket fordított sorrendben írunk fel.Ezt az elvet alkalmazva felcseréljük a "változó == konstans" feltételt "konstans == változó"-ra.

```
class yoda
{
  public static void main(String[] args)
  {
    String x = null;

    /*if(x.equals("szoveg"))
    System.out.println("asd");*/

      if("szoveg".equals(x))
      System.out.println("asd");
  }
```

```
}
```

A kommentben lévő kódot használva NullPointerExceptiont kapunk, de a Yoda Conditonssal működik. Ez azért van ,mert ha a kommentes részben hívnánk az equals-t azt nem tudnánk megtenni mert null objektum. De mivel a "szoveg" szót egyből String objektumként kezeli a Java, ennek már lesz equals metódusa. Tehát vannak esetek melyben a Yoda conditions hasznunkra válik,de így a kódunk nem lesz olyan könnyen átlátható, hiszen nem ez a megszokott sorrend.

## 12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp- alg.pdf és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását!

Ha megakadsz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok- javat/apbs02.htm (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

```
public class PiBBP {
  String d16PiHexaJegyek;
  public PiBBP(int d) {

    double d16Pi = 0.0d;

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


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

    d16Pi = d16Pi - StrictMath.floor(d16Pi);

    StringBuffer sb = new StringBuffer();

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

    while(d16Pi != 0.0d) {
      int jegy = (int)StrictMath.floor(16.0d*d16Pi);
    if(jegy<10)
      sb.append(jegy);
    else
      sb.append(hexaJegyek[jegy-10]);

    d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
  }
  d16PiHexaJegyek = sb.toString();
}
```

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

  double d16Sj = 0.0d;

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

  return d16Sj - StrictMath.floor(d16Sj);

}

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;
  }
  public String toString() {

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

A feladat elvégzéséhez a Bailey-Bolwein-Plouffe formula ismerete szükséges.

Megadunk egy Character nevű tömböt, melyben hexadecimális (16-os számrendszer-beli) számokat adunk meg.Erre azért van szükségünk mert a Java alapértelmezetten nem tartalmazza ezeket.

Az számjegyek kiszámolása után a sztring bufferhez a kiszámolt szám hexadecimális megfelelőjét fűzzük hozzá. Ez csak akkor történik meg ha a számjegy minimum 10.

Miután kiszámoltuk a megfelelő számú számjegyet, a string buffert stringgé konvertáljuk.

# 13. fejezet

# Helló, Liskov!

## 13.1. Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés.

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai- Barki/madarak/)

A Liskov-elv szerint, ha S altípusa T-nek, akkor minden olyan helyen ahol T-t felhasználjuk S-t is minden gond nélkül behelyettesíthetjük anélkül, hogy a programrész tulajdonságai megváltoznának.A következő példaprogramban megnézhetjük, hogy ezzel milyen problémákba ütközhetünk.

C++ nyelven írt kód :

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

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

class Sas : public Madar
{};

class Pingvin : public Madar
{};

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

```
  Program program;
  Madar madar;
  program.fgv ( madar );

  Sas sas;
  program.fgv ( sas );

  Pingvin pingvin;
  program.fgv ( pingvin );
}
```

Java nyelven írt kód :

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

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

static class Sas extends Madar{}
static class Pingvin extends Madar{}

public static void main(String[] args) {

  Program program = new Program();
  Madar madar = new Madar();
  program.fgv(madar);

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

  Pingvin pingvin = new Pingvin();
  program.fgv(pingvin);

  }
}
```

Az előző kódokban a Madar nevű osztályt tekintjük T-nek és a Sas-t és a Pingvin-t S-nek.Azaz a Sas és Pingvin osztályok a Madar nevű osztály alosztályai.

A Madar nevű osztályban található egy repul() függvény , ami azt jelenti hogy a madarak tudnak repülni.De

ez a tény nem minden esetben igaz, hiszen nem minden madár tud repülni, de a program szerint a Pingvin is repülne.

Vagyis van olyan leszármazott (pingvin), ami nem tud olyat (repülni), amit a őse (madár) tud, vagyis nem lehet a őst a leszármazottal helyettesíteni. Azaz a Liskov-elv sérül.

Ezt a problémát pontosabb, részletesebb tervezéssel elkerülhetjük, ebben az esetben például segítene egy repülő madár osztály bevezetése.

## 13.2. Szülő-gyerek

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

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

Ehhez a feladathoz ismernünk kell az öröklődést és a polimorfizmust. Ez azt jelenti hogyha egy osztályt egy másik osztály kiterjesztéseként definiálunk, akkor az eredeti osztály lesz a szülőosztály és a kiterjesztett osztály pedig a gyermekosztály.A gyermekosztály rendelkezik a szülők tagváltozóival és metódusaival, de csak azkohoz fér hozzá amelyekhez engedélyt kap. A public kulcsszóval megadott tagváltozók és metódusok mindenki számára elérhetőek és a protected kulcsszóval rendelkezők csak a szülőosztályok saját gyermekosztályai számára.

A gyermekosztályt minden környezetben lehet használni amelyben a szülőosztályát is lehet használni, ez a polimorfizmus.

C++

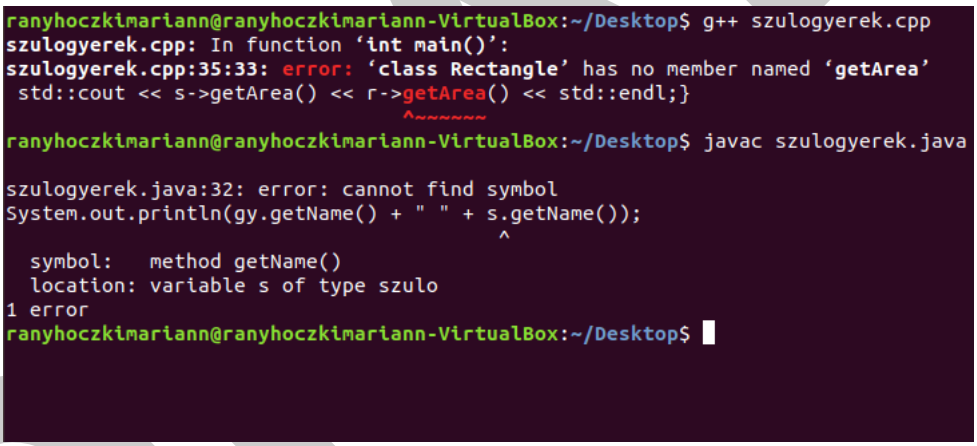
```
#include<iostream>

class Rectangle
{

public:
  int m_width;
  int m_height;

  Rectangle(int width, int height) {
    m_width = width;
    m_height = height;
  }

  int getWidth(){
  return m_width;

  }

  int getHeight(){
  return m_height;
```

```
  }
};

class Square : public Rectangle
{

public:
  Square(int side): Rectangle(side, side)
  {
    m_width = side;
    m_height = side;
  }

  int getArea(){

    return m_width * m_height;
  }
};

int main ()
{
  Rectangle* r = new Square(10);
  Square* s = new Square(10);
  std::cout << s->getArea() << r->getArea() << std::endl;
}
```

Java

```
class szulo
{

  protected int m_age;
  protected String m_name;
  public void setAge(int age) {
  m_age = age;
  }

  public void setName(String name) {
  m_name = name;
  }

  public int getAge(){
  return m_age;
  }
}

class gyerek extends szulo
{
  public String getName() {
```

```
    return m_name;
  }
}

class szuloGyerek
{

  public static void main (String args[])
  {
    szulo s = new gyerek();
    s.setName("Apuci");
    s.setAge(60);

    gyerek gy = new gyerek();
    s.setName("Laci");
    s.setAge(15);

    System.out.println(gy.getName() + " " + s.getName());
  }
}
```




Mint a képen is látható mind a 2 kóddal hibaüzeneteket kaptunk. Ebből látszik,hogy nem tudjuk elérni a gyerek osztályban definiált függvényeket ,úgy hogy a gyerek típust szülővé castoljuk.

## 13.3. Ciklomatikus komplexitás

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

A ciklomatikus komplexitás a vezérlési gráfban megtalálható független utak maximális száma. Két út független, ha mindkettőben létezik olyan pont vagy él, amelyik nem eleme a másik útnak. A ciklomatikus komplexitás értéke arra jellemző, hogy a program vezérlési szempontból mennyire bonyolult.

A ciklomatikus (McCabe) komplexitás értéke:

M = E - N + 2P 1

ahol

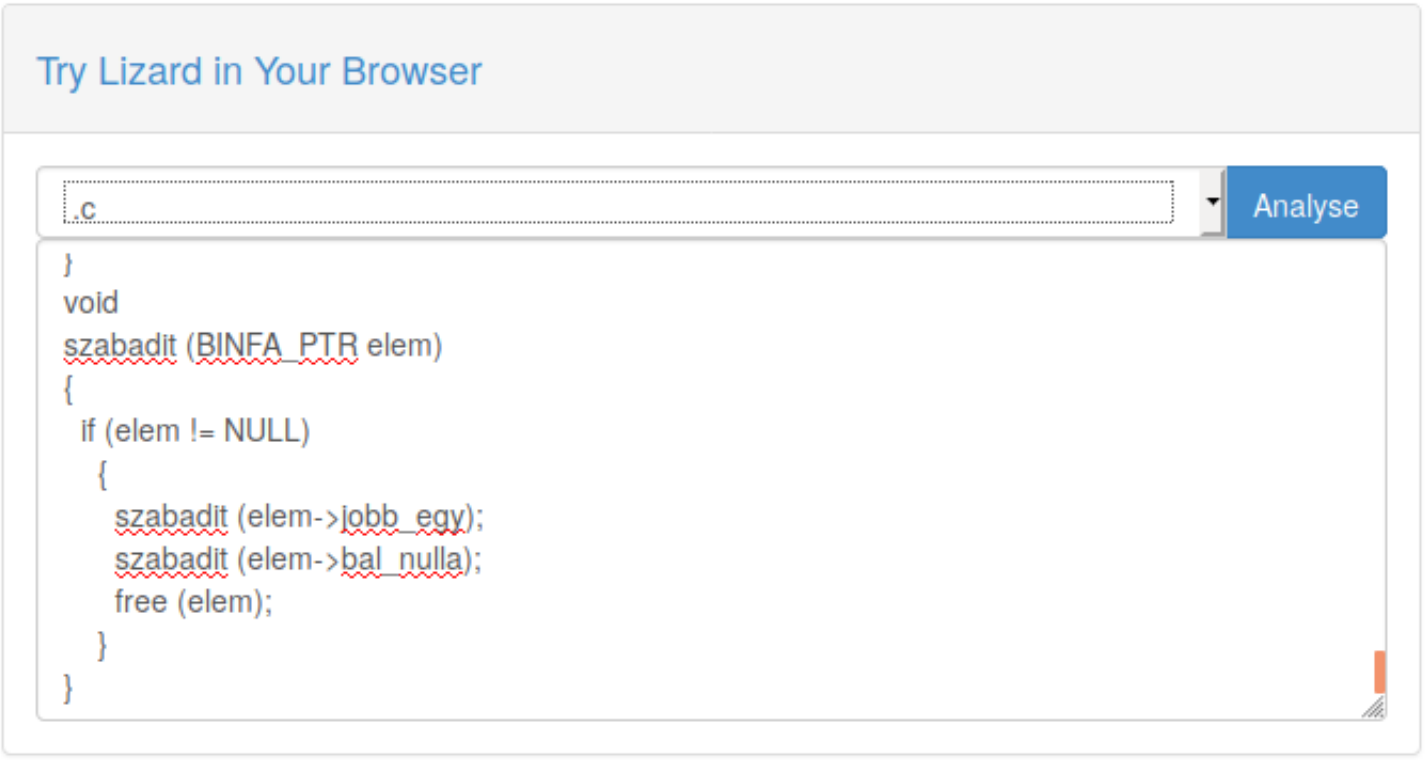
E: A gráf éleinek száma

N: A gráfban lévő csúcsok száma

P: Az összefüggő komponensek száma

A ciklomatikus szám: M = E - N + P

Én a lizars.ws online komplexitás mérő programját használtam, és az LZW binfa C verzióját választottam.




A programban ki kell választanunk hogy milyen nyelven írt kódot szeretnénk elemezni , majd az egész kódunkat beilleszteni.


| Function Name | NLOC | Complexity | Token # | Parameter # |
|---|---|---|---|---|
| uj_elem | 10 | 2 | 42 | |
| main | 44 | 5 | 203 | |
| kiir | 16 | 5 | 87 | |
| szabadit | 9 | 2 | 34 | |

Az alábbi táblázatban kapjuk meg az értékeket.Az érték minél alacsonyabb annál jobb. Ha az érték túl nagy akkor a függvényünk valószínűleg túl komplex és megpróbálhatjuk további bontását.

# 14. fejezet

# Helló, Mandelbrot 2.0!

## 14.1. Reverse engineering UML osztálydiagram

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

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

Az UML egy szabványos modellezőnyelv , amelynek segítségével fejlesztési modellek és különböző rendszerekben lévő kapcsolatok rendkívül jól szemléltethetőek. Célja, a tervezés és dokumentálás grafikus formában, ábrák, diagramok segítségével . A diagramban vonalakat, különböző ábrákat,jeleket és szöveget találunk.

A feladatot az első védési C++ program felhasználásával kell elvégeznünk, azaz a binfával.

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

class LZWBinFa {
public:
  LZWBinFa () :fa ( &gyoker ) {
  }
  ~LZWBinFa () {
    std::cout << "LZWBinFa dtor" << std::endl;
    szabadit ( gyoker.egyesGyermek () );
    szabadit ( gyoker.nullasGyermek () );
  }

  LZWBinFa ( const LZWBinFa & regi ) {
    std::cout << "LZWBinFa copy ctor" << std::endl;

  gyoker.ujEgyesGyermek ( masol ( regi.gyoker.egyesGyermek (), regi.fa ) );
  gyoker.ujNullasGyermek ( masol ( regi.gyoker.nullasGyermek (), regi.fa )  ↩
     );
```

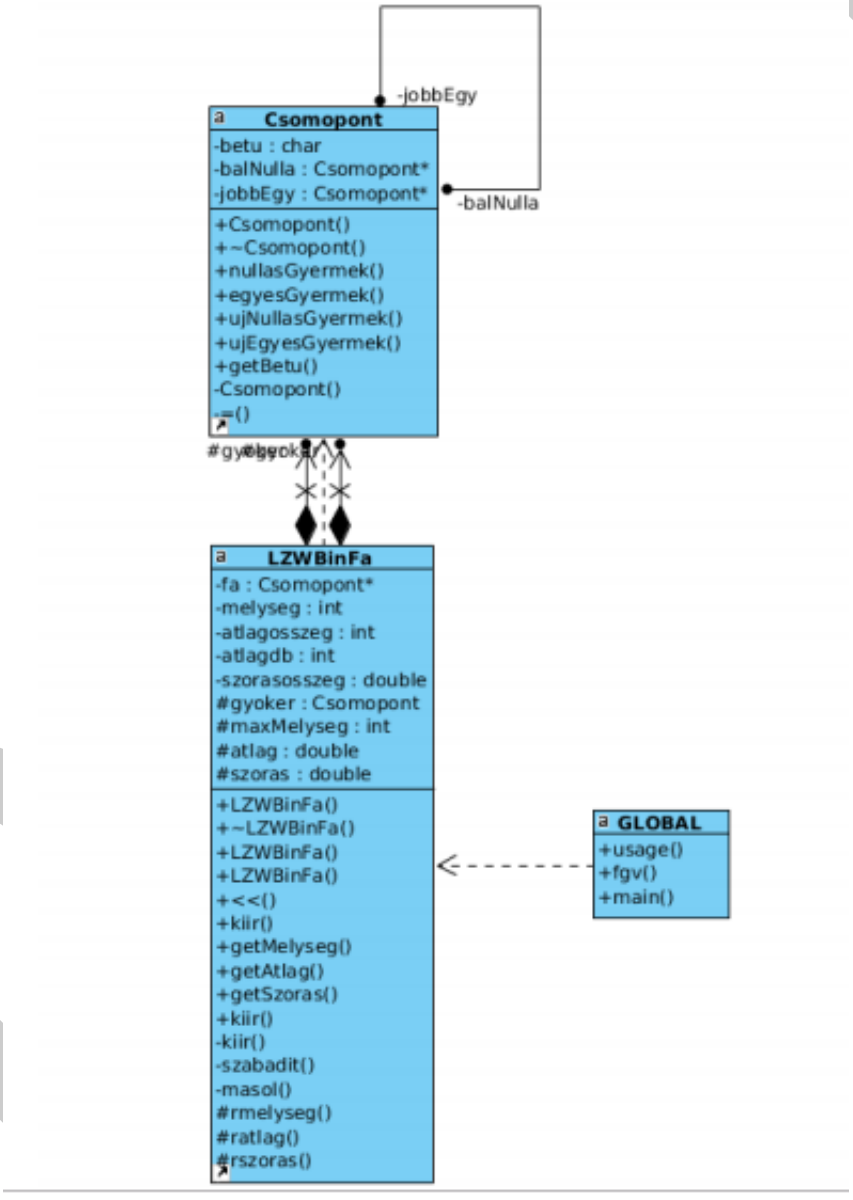
```
if ( regi.fa == & ( regi.gyoker ) )
  fa = &gyoker;

}

LZWBinFa ( LZWBinFa && regi ) {
  std::cout << "LZWBinFa move ctor" << std::endl;

gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

regi.gyoker.ujEgyesGyermek ( nullptr );
regi.gyoker.ujNullasGyermek ( nullptr );
}

LZWBinFa& 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 ();
    }
  }
  return *this;
}

void kiir ( void ) {

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

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

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

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

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

    Csomopont *nullasGyermek () const {
      return balNulla;
    }

    Csomopont *egyesGyermek () const {
      return jobbEgy;
    }


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


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

    char getBetu () const {
      return betu;
    }
private:

  char betu;

  Csomopont *balNulla;
  Csomopont *jobbEgy;

  Csomopont ( const Csomopont & );
  Csomopont & operator= ( const Csomopont & );
};
```

```
Csomopont *fa;

int melyseg, atlagosszeg, atlagdb;
double szorasosszeg

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;
  }
}

Csomopont * masol ( Csomopont * elem, Csomopont * regifa ) {
  Csomopont * ujelem = NULL;
  if ( elem != NULL ) {
    ujelem = new Csomopont ( elem->getBetu() );
    ujelem->ujEgyesGyermek ( masol ( elem->egyesGyermek (), regifa ) );
    ujelem->ujNullasGyermek ( masol ( elem->nullasGyermek (), regifa ) );
    if ( regifa == elem )
      fa = ujelem;

  }
  return ujelem;
}


protected:

  Csomopont gyoker;
  int maxMelyseg;
  double atlag, szoras;
```

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

Az itt megtalálható kódból végeztem el az UML generálását,melyhez a Visual Paradigmot használtam.Ez a program könnyen kezelhető,de főként azért választottam ezt, mert van online verziója is, melyet 7 napig ingyen kipróbálhatunk. A következő eredmény született:




A feladatban ki kell térnem az aggregáció és kompozíció kapcsolatára.Mind a kettő az asszociáció formája, azaz az osztályok közötti kapcsolatokra vonatkozik.

Az aggregáció azt jelenti hogy az adott rész tartozik valakihez,de önmagában is képes létezni.Az aggregációs egység egyszerre több tartalmazónak is lehet a része és a rész előbb létrejöhet mint a tartalmazó és el

is válhat tőle.

A kompozíció egy tartalmazó rész, mely nem létezhet a önmagában.A tartalmazottal együtt jön létre és csak ennek megszűnésével szűnik meg.

A kompozíciót a tartalmazó vége felé lévő fekete vonalon lévő fekete rombusz jelöli, míg az aggregációt hasonlóan de kitöltés nélküli rombusz jelöli.

## 14.2. Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generáljunk belőle forrást!

A Forward engineeringet akkor használhatjuk ha az átláthatóság érdekében előbb szeretnénk megtervezni a diagramot és majd ezután írjuk meg a programot. Ez azoknak kedvező akik jobban szeretik vizuálisan látni,hogy minek kell szerepelni a programjukban.

Mivel egy diagramra volt szükségem amelyből forrást kellett generálnom ezért én az előző feladatból készített uml-t használtam az egyszerűség kedvéért.

Íme az előző UML-ből generált kód:

```
class LZWBinFa {

protected:
  Csomopont gyoker;

private:
  Csomopont* fa;
  int melyseg;
  int atlagosszeg;
  int atlagdb;
  double szorasosszeg;

protected:
  int maxMelyseg;
  double atlag;
  double szoras;

public:
  LZWBinFa();
  void ~LZWBinFa();
  LZWBinFa(const LZWBinFa& regi);
  LZWBinFa(LZWBinFa&& regi);
  void kiir();
  int getMelyseg();
  double getAtlag();
  double getSzoras();
    void kiir(std::ostream& os);
```

```
private:

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

  void szabadit(Csomopont* elem);

  Csomopont* masol(Csomopont* elem, Csomopont* regifa);

protected:

  void rmelyseg(Csomopont* elem);

  void ratlag(Csomopont* elem);

  void rszoras(Csomopont* elem);

  };
class Csomopont {

private:
  char betu;

  Csomopont* balNulla;

  Csomopont* jobbEgy;


public:

  Csomopont(char b = '/');

  void ~Csomopont();

  Csomopont* nullasGyermek();

  Csomopont* egyesGyermek();

  void ujNullasGyermek(Csomopont* gy);

  void ujEgyesGyermek(Csomopont* gy);

  char getBetu();

private:

  Csomopont(const Csomopont& unnamed_1);
  };
```

A kódban láthatjuk a különbséget az első feladatban írt binfához és ehhez a kódhoz képest. Ebben a kódban csak az osztályok felépítését láthatjuk a header fájlokban, a függvények törzse kitöltetlen marad .Ez azért van mert a generálás során a generátor nem tudja, hogy a törzsek mit tartalmaznak ,csak az osztályok közötti kapcsolatokat vizsgálja.

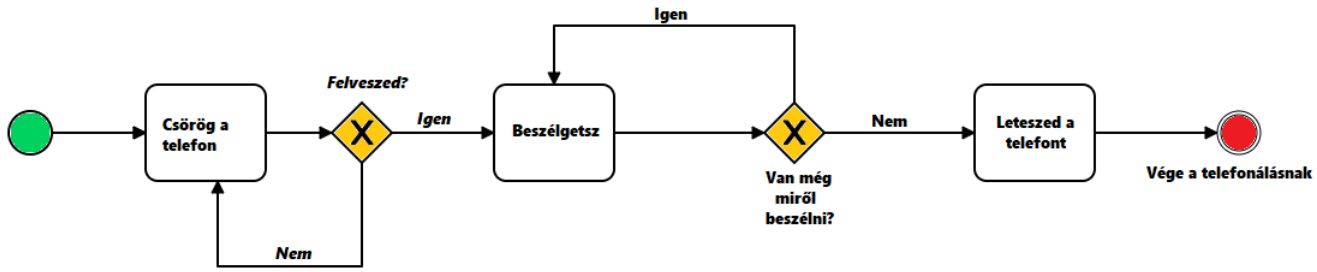
Tehát az elkészített UML-ből kód generálása teljesen működő dolog, és azoknak akik szeretik vizuálisan látni mit készítenek hasznos tud lenni hiszen a diagramból header fájl generálható.

## 14.3. BPMN

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

A feladatom az volt hogy rajzoljak le egy tevékenységet BPMN-ben! A BPMN (Business Process Model and Notation) egy az üzleti folyamatok grafikus ábrázolására vonatkozó szabvány. Azért kötődik témánkhoz mert ezzel a módszerrel is grafikai modellezést végezhetünk.

A rajzomban egy nagyon egyszerű példát, egy telefonhívást láthatunk.




Mint a képen láthatjuk a cselekmény azzal kezdődik hogy csörög a telefon. Ezután választanunk kell hogy felvesszük vagy nem, ha nem akkor újra csörög ,ha igen akkor elkezdünk beszélgetni.Hogyha van téma amiről beszélni kell akkor folytatjuk a beszélgetést, ha nincs akkor letesszük a telefont.

Ez egy nagyon primitív példa, ennél nyílván lehet sokkal komplexebbeket is készíteni. Gyakran alkalmazzák ezt az ábrázolási módot hiszen könnyen átlátható.

# 15. fejezet

# Helló, Chomsky!

## 15.1. Encoding

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

A feladatunkban a lényeg az ékezetes betűk megtartása, a karakterkészlet megfelelő használata volt.

Ha a linken lévő java kódot a szokásos módon futtatjuk rengeteg hibaüzenetet kapunk vissza.

```
ranyhoczkimariann@ranyhoczkimariann-VirtualBox:~/Desktop/Prog/mandelbrot$ javac
MandelbrotHalmazNagyító.java
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xED) for encoding
UTF-8
 * MandelbrotHalmazNagy◆t◆.java
                       ^
MandelbrotHalmazNagyító.java:2: error: unmappable character (0xF3) for encoding
UTF-8
 * MandelbrotHalmazNagy◆t◆.java
                        ^
MandelbrotHalmazNagyító.java:4: error: unmappable character (0xED) for encoding
UTF-8
 * DIGIT 2005, Javat tan◆tok
                      ^
MandelbrotHalmazNagyító.java:5: error: unmappable character (0xE1) for encoding
UTF-8
 * B◆tfai Norbert, nbatfai@inf.unideb.hu
    ^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xED) for encoding
UTF-8
 * A Mandelbrot halmazt nagy◆t◆ ◆s kirajzol◆ oszt◆ly.
                            ^
MandelbrotHalmazNagyító.java:9: error: unmappable character (0xF3) for encoding
UTF-8
 * A Mandelbrot halmazt nagy◆t◆ ◆s kirajzol◆ oszt◆ly.
                             ^
```

```
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
 UTF-8
           // vizsg◆ljuk egy adott pont iter◆ci◆it:
                    ^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
 UTF-8
           // vizsg◆ljuk egy adott pont iter◆ci◆it:
                                          ^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
 UTF-8
           // vizsg◆ljuk egy adott pont iter◆ci◆it:
                                            ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
 UTF-8
               // Az eg◆rmutat◆ poz◆ci◆ja
                             ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
 UTF-8
               // Az eg◆rmutat◆ poz◆ci◆ja
                              ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
 UTF-8
               // Az eg◆rmutat◆ poz◆ci◆ja
                                   ^
100 errors
ranyhoczkimariann@ranyhoczkimariann-VirtualBox:~/Desktop/Prog/mandelbrot$
```

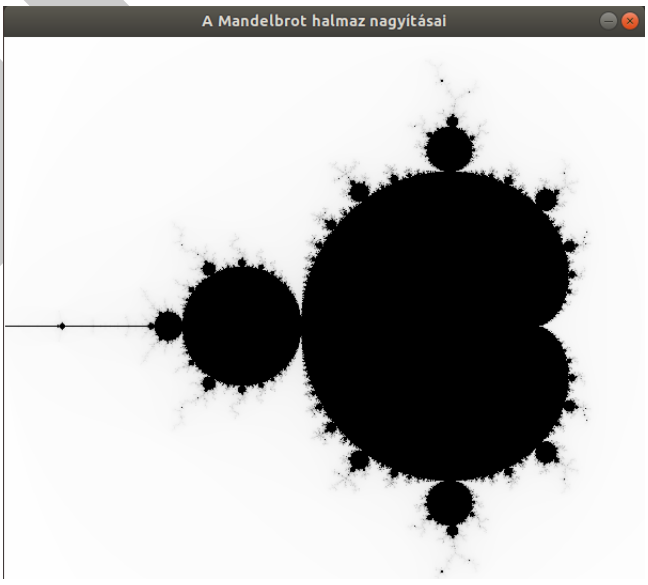
A hibaüzenetekben azt válaszolja a fordító hogy az UTF-8 kódolás számára nem található a karakter. Ebből következtetünk arra hogy a forráskód kódolásával vannak problémák.

Ahhoz hogy ékezetes betűket használjunk a Latin1 vagy a Latin2 kódolást kell használnunk, viszont a Latin2-t érdemesebb használni mert az ’ű’ és az ’ő’ karaktereket a Latin1 nem tartalmazza.

Ha maguk az állomány nevek rendben vannak, mert eleve Linux alatt készítettük el őket, akkor elegendő a fordításnál az encoding kapcsoló használata. Megkerestem a Latin2 kapcsolóját, ami az iso 8859-2 volt és ezekkel már sikerült a fordítás.

```
ranyhoczkimariann@ranyhoczkimariann-VirtualBox:~/Desktop/Prog/mandelbrot$ javac
-encoding iso-8859-2 MandelbrotIterációk.java
ranyhoczkimariann@ranyhoczkimariann-VirtualBox:~/Desktop/Prog/mandelbrot$ javac
-encoding iso-8859-2 MandelbrotHalmazNagyító.java
ranyhoczkimariann@ranyhoczkimariann-VirtualBox:~/Desktop/Prog/mandelbrot$ java M
andelbrotHalmazNagyító
```

És itt a nagyító működés közben:




## 15.2. Full screen (zöld)

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

A feladat egy teljes képernyős Java program megírása volt, én ehhez a korábbi labdapattogás projektet választottam.

A kód:

```
import java.awt.*;
import java.awt.image.BufferStrategy;

public class BouncingBall {

  int startX, x = 200;
  int startY, y = 200;
  int ynov=1, xnov=1;
  int radius = 50;
  int maxX, maxY;

private static DisplayMode[] BEST_DISPLAY_MODES = new DisplayMode[] {

  new DisplayMode(1920, 1080, 32, 0),
  new DisplayMode(1920, 1080, 16, 0),
  new DisplayMode(1920, 1080, 8, 0)
};

Frame mainFrame;
Color ballColor, backgroundColor;
```

```
public void move() {

        if ( x>=maxX-50 ) { // elerte-e a jobb oldalt?
            xnov = xnov * -1;
            startX=x;
            startY=y;
        }
        if ( x<=50 ) { // elerte-e a bal oldalt?
            xnov = xnov * -1;
            startX=x;
            startY=y;
        }
        if ( y<=50 ) { // elerte-e a tetejet?
            ynov = ynov * -1;
            startY=y;
            startX=x;
        }
        if ( y>=maxY-50 ) { // elerte-e a aljat?
            ynov = ynov * -1;
            startY=y;
            startX=x;
        }

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

    }

public BouncingBall (int numBuffers, GraphicsDevice device) {
  try {

  GraphicsConfiguration gc = device.getDefaultConfiguration();
  mainFrame = new Frame(gc);
  mainFrame.setUndecorated(true);
  mainFrame.setIgnoreRepaint(true);
  device.setFullScreenWindow(mainFrame);
  if (device.isDisplayChangeSupported()) {
    chooseBestDisplayMode(device);
  }
  Rectangle bounds = mainFrame.getBounds();
  bounds.setSize(device.getDisplayMode().getWidth(), device.getDisplayMode ↩
     ().getHeight());

  maxX = device.getDisplayMode().getWidth();
  maxY = device.getDisplayMode().getHeight();


  mainFrame.createBufferStrategy(numBuffers);
  BufferStrategy bufferStrategy = mainFrame.getBufferStrategy();
```

```
  ballColor=new Color(150,230,129);
  backgroundColor=new Color(224,224,224);

while(true) {

  Graphics g = bufferStrategy.getDrawGraphics();

  if (!bufferStrategy.contentsLost()) {
    move();
    g.setColor(backgroundColor);
    g.fillRect(0,0,bounds.width, bounds.height);

    g.setColor(ballColor);
    g.fillOval(x, y, radius, radius);
    bufferStrategy.show();
    g.dispose();
    }

try {
  Thread.sleep(3);
} catch (InterruptedException e) {}
  }
} catch (Exception e) {
  e.printStackTrace();
} finally {
  device.setFullScreenWindow(null);
  }
}

private static DisplayMode getBestDisplayMode(GraphicsDevice device) {

for (int x = 0; x < BEST_DISPLAY_MODES.length; x++) {
  DisplayMode[] modes = device.getDisplayModes();
    for (int i = 0; i < modes.length; i++) {
      if (modes[i].getWidth() == BEST_DISPLAY_MODES[x].getWidth()
        && modes[i].getHeight() == BEST_DISPLAY_MODES[x].getHeight()
          && modes[i].getBitDepth() == BEST_DISPLAY_MODES[x].getBitDepth()) ↩
              {
  return BEST_DISPLAY_MODES[x];
      }
    }
  }
  return null;
}


public static void chooseBestDisplayMode(GraphicsDevice device) {

DisplayMode best = getBestDisplayMode(device);
  if (best != null) {
```

```
    device.setDisplayMode(best);
  }
}

public static void main(String[] args) {

try {
    int numBuffers = 2;

    GraphicsEnvironment env = GraphicsEnvironment.
    getLocalGraphicsEnvironment();
    GraphicsDevice device = env.getDefaultScreenDevice();
    BouncingBall ball = new BouncingBall(numBuffers, device);
} catch (Exception e) {
    e.printStackTrace();
  }
    System.exit(0);
  }
}
```

A lényeg a teljes képernyős mód volt melyet úgy tudtunk elérni, hogy a frame-et undecorated-re állítjuk a tényleges teljes képernyős módért.

Ezután megtaláljuk a setFullScreenWindows függvényt amellyel átlépunk a teljes képernyős módba.

A feladat bemutatására egy videó hasznosabb lenne,de a teljes képernyős módot egy képernyőképpel mutatom be.

## 15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

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

Mi is az a Paszigráfia Rapszódia?

A következo idézet a vis_prel_para.pdf-ből jól összefoglalja: ˝ "A Paszigráfia Rapszódia, vagy röviden PaRa egy olyan mesterséges nyelv kialakítására törekvő kezdeményezés, mely lehetoővé teszi a homunkulusz és a mesterséges homunkulusz közötti kommunikációt "

A program OpenGL-t (Open Graphic Library) használ a megjelenítéshez ,melynek használatához szükségünk van a következők telepítésére:

```
sudo apt-get install freeglut3-dev
sudo apt-get install mesa-common-dev libgl1-mesa-dev libglu1-mesa-dev
```

A forráskód :

```
#include <iostream>
#include <vector>
#include <boost/tokenizer.hpp>
#include <GL/glut.h>

class PaRaCube
{
public:
        GLfloat rotx = 0.0f;
        GLfloat roty = 0.0f;
        GLfloat rotz = 0.0f;
        int nn[6]  = {1,2,3,4,8,10};
        std::vector<int> cc[6];
};

int index = 0;
bool transp {false};
GLdouble fovy = 70;
int w = 640;
int h = 480;

std::vector<PaRaCube> cubeLetters;

void drawPaRaCube ( int idx )
{

        glPushMatrix();
```

```
        int d = cubeLetters.size() /2 ;
        glTranslatef ( ( idx-d ) *2.5f, 0.0f, 0.0f );

        glRotatef ( cubeLetters[idx].rotx, 1.0f, 0.0f, 0.0f );
        glRotatef ( cubeLetters[idx].roty, 0.0f, 1.0f, 0.0f );
        glRotatef ( cubeLetters[idx].rotz, 0.0f, 0.0f, 1.0f );

        glBegin ( GL_QUADS );

        glColor3f ( 0.818f, .900f, 0.824f );

        glVertex3f ( -1.0f, 1.0f, 1.0f );
        glVertex3f ( 1.0f, 1.0f, 1.0f );
        glVertex3f ( 1.0f,-1.0f, 1.0f );
        glVertex3f ( -1.0f,-1.0f, 1.0f );

        glVertex3f ( 1.0f, 1.0f, 1.0f );
        glVertex3f ( 1.0f, 1.0f,-1.0f );
        glVertex3f ( 1.0f,-1.0f,-1.0f );
        glVertex3f ( 1.0f,-1.0f, 1.0f );

        glVertex3f ( -1.0f, 1.0f, 1.0f );
        glVertex3f ( -1.0f, 1.0f,-1.0f );
        glVertex3f ( 1.0f, 1.0f,-1.0f );
        glVertex3f ( 1.0f, 1.0f, 1.0f );

        glVertex3f ( -1.0f, 1.0f, 1.0f );
        glVertex3f ( -1.0f, 1.0f,-1.0f );
        glVertex3f ( -1.0f,-1.0f,-1.0f );
        glVertex3f ( -1.0f,-1.0f, 1.0f );

        glVertex3f ( -1.0f, 1.0f,-1.0f );
        glVertex3f ( 1.0f, 1.0f,-1.0f );
        glVertex3f ( 1.0f,-1.0f,-1.0f );
        glVertex3f ( -1.0f,-1.0f,-1.0f );

        glVertex3f ( -1.0f,-1.0f, 1.0f );
        glVertex3f ( 1.0f,-1.0f, 1.0f );
        glVertex3f ( 1.0f,-1.0f,-1.0f );
        glVertex3f ( -1.0f,-1.0f,-1.0f );

        glEnd();

        glBegin ( GL_LINES );
        glColor3f ( .188f, 0.209f, 0.190f );

        for ( int i=0; i<=cubeLetters[idx].nn[0]; i++ ) {

                glVertex3f ( -1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ), ↩
                    1.005f );
```

```
                glVertex3f ( 1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ),  ↩
                   1.005f );
        }
        for ( int i=0; i<=cubeLetters[idx].nn[0]; i++ ) {

                glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ), 1.0f , ↩
                    1.005f );
                glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[0] ), -1.0f  ↩
                   , 1.005f );

        }
        glEnd();


        for ( int i {0}; i<cubeLetters[idx].cc[0].size() /2; ++i ) {
                glBegin ( GL_QUADS );

                glColor3f ( .82f, .15f, .15f );

                glVertex3f ( 1.0f- ( cubeLetters[idx].cc[0][2*i]+1 ) * (  ↩
                   2.0/cubeLetters[idx].nn[0] ),
                                1.0f- ( cubeLetters[idx].cc[0][2*i+1]+1 ) * (  ↩
                                   2.0/cubeLetters[idx].nn[0] ), 1.002f );
                glVertex3f ( 1.0f-cubeLetters[idx].cc[0][2*i]* ( 2.0/ ↩
                   cubeLetters[idx].nn[0] ),
                                1.0f- ( cubeLetters[idx].cc[0][2*i+1]+1 ) * (  ↩
                                   2.0/cubeLetters[idx].nn[0] ), 1.002f );
                glVertex3f ( 1.0f-cubeLetters[idx].cc[0][2*i]* ( 2.0/ ↩
                   cubeLetters[idx].nn[0] ),
                                1.0f-cubeLetters[idx].cc[0][2*i+1]* ( 2.0/ ↩
                                   cubeLetters[idx].nn[0] ), 1.002f );
                glVertex3f ( 1.0f- ( cubeLetters[idx].cc[0][2*i]+1 ) * (  ↩
                   2.0/cubeLetters[idx].nn[0] ),
                                1.0f-cubeLetters[idx].cc[0][2*i+1]* ( 2.0/ ↩
                                   cubeLetters[idx].nn[0] ), 1.002f );

                glEnd();
        }


        glBegin ( GL_LINES );
        glColor3f ( .188f, 0.209f, 0.190f );

        for ( int i=0; i<=cubeLetters[idx].nn[1]; i++ ) {

                glVertex3f ( 1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[1] ) ↩
                   , 1.0f );
                glVertex3f ( 1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[1] ) ↩
                   , -1.0f );
        }
```

```
        for ( int i=0; i<=cubeLetters[idx].nn[1]; i++ ) {

                glVertex3f ( 1.005f, 1.0f , 1.0f-i* ( 2.0/cubeLetters[idx]. ↩
                   nn[1] ) );
                glVertex3f ( 1.005f, -1.0f , 1.0f-i* ( 2.0/cubeLetters[idx ↩
                   ].nn[1] ) );

        }
        glEnd();

        for ( int i {0}; i<cubeLetters[idx].cc[1].size() /2; ++i ) {
                glBegin ( GL_QUADS );

                glColor3f ( 0.15f, .29f, .82f );

                glVertex3f ( 1.002f, 1.0f-cubeLetters[idx].cc[1][2*i]* (  ↩
                   2.0/cubeLetters[idx].nn[1] ),
                                1.0f- ( cubeLetters[idx].cc[1][2*i+1]+1 ) * (  ↩
                                   2.0/cubeLetters[idx].nn[1] ) );
                glVertex3f ( 1.002f, 1.0f-cubeLetters[idx].cc[1][2*i]* (  ↩
                   2.0/cubeLetters[idx].nn[1] ),
                                1.0f-cubeLetters[idx].cc[1][2*i+1]* ( 2.0/ ↩
                                   cubeLetters[idx].nn[1] ) );
                glVertex3f ( 1.002f, 1.0f- ( cubeLetters[idx].cc[1][2*i]+1  ↩
                   ) * ( 2.0/cubeLetters[idx].nn[1] ),
                                1.0f-cubeLetters[idx].cc[1][2*i+1]* ( 2.0/ ↩
                                   cubeLetters[idx].nn[1] ) );
                glVertex3f ( 1.002f, 1.0f- ( cubeLetters[idx].cc[1][2*i]+1  ↩
                   ) * ( 2.0/cubeLetters[idx].nn[1] ),
                                1.0f- ( cubeLetters[idx].cc[1][2*i+1]+1 ) * (  ↩
                                   2.0/cubeLetters[idx].nn[1] ) );


                glEnd();
        }

        glBegin ( GL_LINES );
        glColor3f ( .188f, 0.209f, 0.190f );

        for ( int i=0; i<=cubeLetters[idx].nn[2]; i++ ) {

                glVertex3f ( -1.0f, 1.005f , 1.0f-i* ( 2.0/cubeLetters[idx ↩
                   ].nn[2] ) );
                glVertex3f ( 1.0f, 1.005f , 1.0f-i* ( 2.0/cubeLetters[idx]. ↩
                   nn[2] ) );
        }
        for ( int i=0; i<=cubeLetters[idx].nn[2]; i++ ) {

                glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[2] ), 1.005f ↩
                   , -1.0f );
```

```
                glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[2] ), 1.005f ↩
                   , 1.0f );

        }
        glEnd();

        for ( int i {0}; i<cubeLetters[idx].cc[2].size() /2; ++i ) {
                glBegin ( GL_QUADS );

                glColor3f ( .309f, .820f, .150f );

                glVertex3f ( 1.0f-cubeLetters[idx].cc[2][2*i]* ( 2.0/ ↩
                   cubeLetters[idx].nn[2] ),
                                1.002f , 1.0f-cubeLetters[idx].cc[2][2*i+1]* ( ↩
                                    2.0/cubeLetters[idx].nn[2] ) );
                glVertex3f ( 1.0f- ( cubeLetters[idx].cc[2][2*i]+1 ) * (  ↩
                   2.0/cubeLetters[idx].nn[2] ),
                                1.002f , 1.0f-cubeLetters[idx].cc[2][2*i+1]* ( ↩
                                    2.0/cubeLetters[idx].nn[2] ) );
                glVertex3f ( 1.0f- ( cubeLetters[idx].cc[2][2*i]+1 ) * (  ↩
                   2.0/cubeLetters[idx].nn[2] ),
                                1.002f , 1.0f- ( cubeLetters[idx].cc[2][2*i ↩
                                   +1]+1 ) * ( 2.0/cubeLetters[idx].nn[2] ) );
                glVertex3f ( 1.0f-cubeLetters[idx].cc[2][2*i]* ( 2.0/ ↩
                   cubeLetters[idx].nn[2] ),
                                1.002f , 1.0f- ( cubeLetters[idx].cc[2][2*i ↩
                                   +1]+1 ) * ( 2.0/cubeLetters[idx].nn[2] ) );

                glEnd();
        }

        glBegin ( GL_LINES );
        glColor3f ( .188f, 0.209f, 0.190f );

        for ( int i=0; i<=cubeLetters[idx].nn[3]; i++ ) {

                glVertex3f ( -1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[3]  ↩
                   ), 1.0f );
                glVertex3f ( -1.005f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[3]  ↩
                   ), -1.0f );
        }
        for ( int i=0; i<=cubeLetters[idx].nn[3]; i++ ) {

                glVertex3f ( -1.005f, 1.0f , 1.0f-i* ( 2.0/cubeLetters[idx ↩
                   ].nn[3] ) );
                glVertex3f ( -1.005f, -1.0f , 1.0f-i* ( 2.0/cubeLetters[idx ↩
                   ].nn[3] ) );

        }
        glEnd();
```

```
        for ( int i {0}; i<cubeLetters[idx].cc[3].size() /2; ++i ) {
                glBegin ( GL_QUADS );
                glColor3f ( .804f, .820f, .150f );

                glVertex3f ( -1.002f, 1.0f- ( cubeLetters[idx].cc[3][2*i]+1 ↩
                    ) * ( 2.0/cubeLetters[idx].nn[3] ),
                                1.0f-cubeLetters[idx].cc[3][2*i+1]* ( 2.0/ ↩
                                   cubeLetters[idx].nn[3] ) );
                glVertex3f ( -1.002f, 1.0f-cubeLetters[idx].cc[3][2*i]* (  ↩
                   2.0/cubeLetters[idx].nn[3] ),
                                1.0f-cubeLetters[idx].cc[3][2*i+1]* ( 2.0/ ↩
                                   cubeLetters[idx].nn[3] ) );
                glVertex3f ( -1.002f, 1.0f-cubeLetters[idx].cc[3][2*i]* (  ↩
                   2.0/cubeLetters[idx].nn[3] ),
                                1.0f- ( cubeLetters[idx].cc[3][2*i+1]+1 ) * (  ↩
                                   2.0/cubeLetters[idx].nn[3] ) );
                glVertex3f ( -1.002f, 1.0f- ( cubeLetters[idx].cc[3][2*i]+1 ↩
                    ) * ( 2.0/cubeLetters[idx].nn[3] ),
                                1.0f- ( cubeLetters[idx].cc[3][2*i+1]+1 ) * (  ↩
                                   2.0/cubeLetters[idx].nn[3] ) );

                glEnd();
        }

        glBegin ( GL_LINES );
        glColor3f ( .188f, 0.209f, 0.190f );

        for ( int i=0; i<=cubeLetters[idx].nn[4]; i++ ) {

                glVertex3f ( -1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ), ↩
                    -1.005f );
                glVertex3f ( 1.0f, 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ),  ↩
                   -1.005f );
        }
        for ( int i=0; i<=cubeLetters[idx].nn[4]; i++ ) {

                glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ), 1.0f , ↩
                    -1.005f );
                glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[4] ), -1.0f  ↩
                   , -1.005f );

        }
        glEnd();

        for ( int i {0}; i<cubeLetters[idx].cc[4].size() /2; ++i ) {
                glBegin ( GL_QUADS );
                glColor3f ( .614f, 0.150f, 0.820f );

                glVertex3f ( 1.0f- ( cubeLetters[idx].cc[4][2*i]+1 ) * (  ↩
```

```
                2.0/cubeLetters[idx].nn[4] ),
                        1.0f-cubeLetters[idx].cc[4][2*i+1]* ( 2.0/ ↩
                           cubeLetters[idx].nn[4] ), -1.002f );
        glVertex3f ( 1.0f-cubeLetters[idx].cc[4][2*i]* ( 2.0/ ↩
           cubeLetters[idx].nn[4] ),
                        1.0f-cubeLetters[idx].cc[4][2*i+1]* ( 2.0/ ↩
                           cubeLetters[idx].nn[4] ), -1.002f );
        glVertex3f ( 1.0f-cubeLetters[idx].cc[4][2*i]* ( 2.0/ ↩
           cubeLetters[idx].nn[4] ),
                        1.0f- ( cubeLetters[idx].cc[4][2*i+1]+1 ) * (  ↩
                           2.0/cubeLetters[idx].nn[4] ), -1.002f );
        glVertex3f ( 1.0f- ( cubeLetters[idx].cc[4][2*i]+1 ) * (  ↩
           2.0/cubeLetters[idx].nn[4] ),
                        1.0f- ( cubeLetters[idx].cc[4][2*i+1]+1 ) * (  ↩
                           2.0/cubeLetters[idx].nn[4] ), -1.002f );

        glEnd();
}

glBegin ( GL_LINES );
glColor3f ( .188f, 0.209f, 0.190f );

for ( int i=0; i<=cubeLetters[idx].nn[5]; i++ ) {

        glVertex3f ( -1.0f, -1.005f , 1.0f-i* ( 2.0/cubeLetters[idx ↩
           ].nn[5] ) );
        glVertex3f ( 1.0f, -1.005f , 1.0f-i* ( 2.0/cubeLetters[idx ↩
           ].nn[5] ) );
}
for ( int i=0; i<=cubeLetters[idx].nn[5]; i++ ) {

        glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[5] ), -1.005 ↩
           f , 1.0f );
        glVertex3f ( 1.0f-i* ( 2.0/cubeLetters[idx].nn[5] ), -1.005 ↩
           f , -1.0f );

}
glEnd();

for ( int i {0}; i<cubeLetters[idx].cc[5].size() /2; ++i ) {
        glBegin ( GL_QUADS );
        glColor3f ( .114f, .108f, .156f );

        glVertex3f ( 1.0f-cubeLetters[idx].cc[5][2*i]* ( 2.0/ ↩
           cubeLetters[idx].nn[5] ),
                        -1.002f , 1.0f-cubeLetters[idx].cc[5][2*i+1]*  ↩
                           ( 2.0/cubeLetters[idx].nn[5] ) );
        glVertex3f ( 1.0f-cubeLetters[idx].cc[5][2*i]* ( 2.0/ ↩
           cubeLetters[idx].nn[5] ),
                        -1.002f , 1.0f- ( cubeLetters[idx].cc[5][2*i ↩
```

```
                                +1]+1 ) * ( 2.0/cubeLetters[idx].nn[5] ) );
                glVertex3f ( 1.0f- ( cubeLetters[idx].cc[5][2*i]+1 ) * (  ↩
                   2.0/cubeLetters[idx].nn[5] ),
                                -1.002f , 1.0f- ( cubeLetters[idx].cc[5][2*i ↩
                                   +1]+1 ) * ( 2.0/cubeLetters[idx].nn[5] ) );
                glVertex3f ( 1.0f- ( cubeLetters[idx].cc[5][2*i]+1 ) * (  ↩
                   2.0/cubeLetters[idx].nn[5] ),
                                -1.002f , 1.0f-cubeLetters[idx].cc[5][2*i+1]*  ↩
                                   ( 2.0/cubeLetters[idx].nn[5] ) );

                glEnd();
        }

        glPopMatrix();

}

void draw ( void )
{
        glClearColor ( 1.0f, 1.0f, 1.0f, 1.0f );

        if ( transp )
                glDisable ( GL_DEPTH_TEST );
        else
                glEnable ( GL_DEPTH_TEST );

        glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

        glMatrixMode ( GL_MODELVIEW );
        glLoadIdentity();
        gluLookAt ( 0.0f, 3.0f, 6.0f ,0.0f ,0.0f ,0.0f ,0.0f ,1.0f ,0.0f );

        for ( int i {0}; i<cubeLetters.size(); ++i )
                if ( i == index ) {
                        drawPaRaCube ( i );
                } else {
                        drawPaRaCube ( i );
                }

        glutSwapBuffers();
}

void keyboard ( unsigned char key, int x, int y )
{
        if ( key == '0' ) {
                index=0;
        } else if ( key == '1' ) {
                index=1;
        } else if ( key == '2' ) {
                index=2;
```

```
        } else if ( key == '3' ) {
                index=3;
        } else if ( key == '4' ) {
                index=4;
        } else if ( key == '5' ) {
                index=5;
        } else if ( key == '6' ) {
                index=6;
        } else if ( key == 't' ) {
                transp = !transp;
        } else if ( key == '-' ) {
                ++fovy;

                glMatrixMode ( GL_PROJECTION );
                glLoadIdentity();
                gluPerspective ( fovy, ( float ) w/ ( float ) h, .1f,  ↩
                   1000.0f );
                glMatrixMode ( GL_MODELVIEW );

        } else if ( key == '+' ) {
                --fovy;

                glMatrixMode ( GL_PROJECTION );
                glLoadIdentity();
                gluPerspective ( fovy, ( float ) w/ ( float ) h, .1f,  ↩
                   1000.0f );
                glMatrixMode ( GL_MODELVIEW );

        }

        glutPostRedisplay();

}

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();
}


void reshape ( int width, int height )
{
        w = width;
        h = height;

        glMatrixMode ( GL_PROJECTION );
        glLoadIdentity();
        gluPerspective ( fovy, ( float ) w/ ( float ) h, .1f, 1000.0f );
        glViewport ( 0, 0, w, h );
        glMatrixMode ( GL_MODELVIEW );
}

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

        for ( int i {1}; i<argc; ++i ) {
                std::string s ( argv[i] );
                std::vector<int> nums;
                boost::char_separator<char> separator ( ":" );
                boost::tokenizer<boost::char_separator<char>> items ( s,  ↩
                   separator );

                for ( const auto& token : items )
                        nums.push_back ( atoi ( token.c_str() ) );

                int ii=0;
                for ( int cui {0}; cui<nums[0]; ++cui ) {
                        PaRaCube prc;
                        for ( int s {0}; s<6; ++s ) {
                                ++ii;
                                prc.nn[s]=nums[ii];
                                ++ii;
                                int noc = nums[ii];
                                for ( int coi {0}; coi<noc; ++coi ) {
                                        ++ii;
                                        prc.cc[s].push_back ( nums[ii] );
                                        ++ii;
                                        prc.cc[s].push_back ( nums[ii] );
                                }
                        }

                        cubeLetters.push_back ( prc );

                }
```

```
        }

        glutInit ( &argc, argv );
        glutInitWindowSize ( w, h );
        glutInitWindowPosition (
                ( glutGet ( GLUT_SCREEN_WIDTH )-w ) /2,
                ( glutGet ( GLUT_SCREEN_HEIGHT )-h ) /2 );
        glutInitDisplayMode ( GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH );
        glutCreateWindow ( "Pasigraphy Rhapsody, para6, exp2" );
        glutReshapeFunc ( reshape );
        glutDisplayFunc ( draw );
        glutKeyboardFunc ( keyboard );
        glutSpecialFunc ( skeyboard );

        glutMainLoop();
        return 0;
}
```

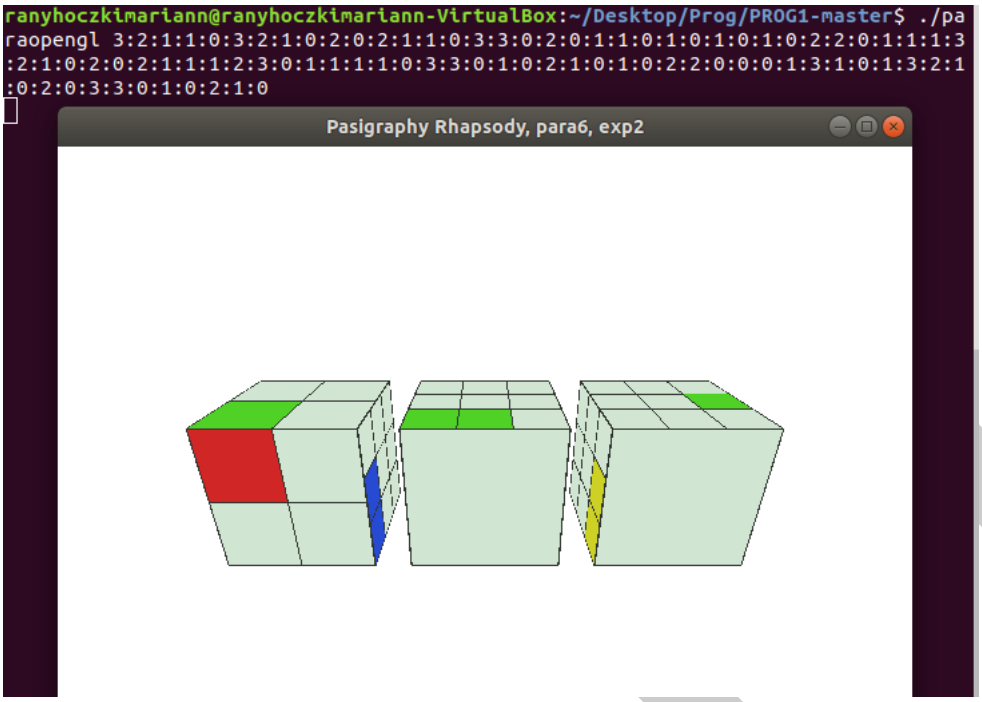
A program futtatásához szükségünk van különböző könyvtárakra melyeket a -l kapcsolóval linkelünk.A program fordítás a következő módon zajlik :

```
g++ para6.cpp -o paraopengl -lGL -lGLU -lglut
```

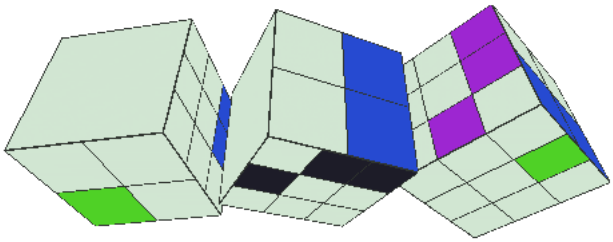
A program futtatása:

```
./paraopengl
3:2:1:1:0:3:2:1:0:2:0:2:1:1:0:3:3:0:2:0:1:1:0:1:0:1:0:1:0:2:2:0:1:1:1:3:2:1: ↩

0:2:0:2:1:1:1:2:3:0:1:1:1:1:0:3:3:0:1:0:2:1:0:1:0:2:2:0:0:0:1:3:1:0:1:3:2:1: ↩

0:2:0:3:3:0:1:0:2:1:0
```

Ezzel a paranccsal a következő képet kapjuk :



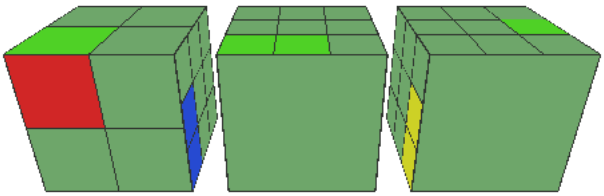

A kockákat egyesével el tudjuk forgatni. A kocka kiválasztása a 0-1-2 gombokkal történik és a nyilakkal történik a forgatás.



Lehetőségünk van a színekt is módoítani. A program RGB színskálában adja meg a színeket. A módosításunk a glColor3f() funkció segítségével valósul meg.

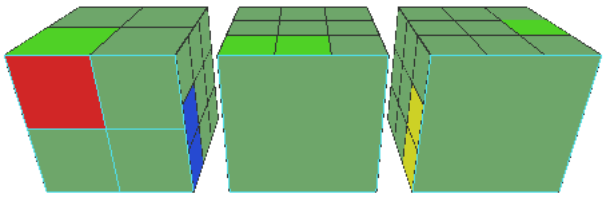
Először a kocka alapszínét módosította a következő helyen random számokkal melyek zöldes színt eredményeztek.

```
glBegin ( GL_QUADS );

glColor3f ( 0.818f, .900f, 0.824f );
```

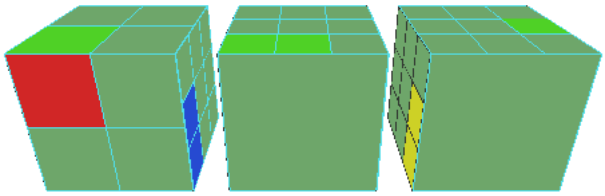


Ezután a kockán lévő négyzetek határolóvonalait színeztem át kékre.

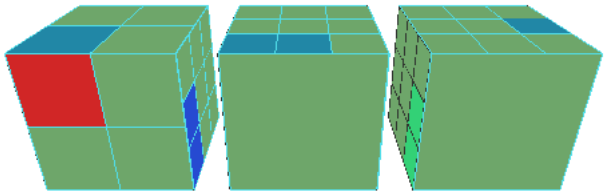
```
glBegin ( GL_LINES );
glColor3f ( .188f, 0.209f, 0.190f );
```



Itt látható,hogy csak egy oldalon változtak az élek színei. Ez azért van mert a kód külön adja meg oldalanként a színeket, tehát oldalanként színezhetjük az éleket. Ha azt szerenénk hogy több kék legyen , több helyen is át kell írnunk a kék színkódot,hogy a következő eredményt kapjuk.



Ezután a színes négyzetekből módosítottam néhányat. Itt is több helyen különböző színeket állíthatunk be. Az én módosításaim után a következő eredményt kaptam:



# 16. fejezet

# Helló, Stroustrup!

## 16.1. JDK osztályok

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

A Boost egy kifejezetten C++hoz használható eszközkönyvtár. 1999-ben adták ki, és azóta folyamatosan fejlesztik.

Ebben a feladatban 2 elemét fogjuk használni a Boost-nak. A Boost.Filesystem és a Boost.Regex könyvtárakat. Előbbire azért lesz szükség, hogy a megadott mappában lévő fájlokat el tudjuk érni, még utóbbira a kiterjesztés szűrése miatt lesz szükség.

```
#include <QCoreApplication>
#include <iostream>
#include <boost/filesystem.hpp>
#include <boost/regex.hpp>


int iterator(boost::filesystem::path p){
    int osztalyok_szama = 0;
    boost::regex expr{"(.*\\.java)"};

    for (const boost::filesystem::directory_entry& x : boost::filesystem:: ↩
       recursive_directory_iterator(p)){

        if(boost::regex_match(x.path().string(), expr) && boost::filesystem ↩
           ::is_regular_file(x.path())){
            std::cout << ++osztalyok_szama << std::endl;
            std::cout << "    " << x.path() << '\n';
        }
    }
    return osztalyok_szama;
}
```

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

    if (argc < 2)
    {
        std::cout << "Usage: tut3 path\n";
        return 1;
    }

    boost::filesystem::path p (argv[1]);

    try
    {
        int osztalyok_szama = iterator(p);
        std::cout << "Java JDK osztályok száma " << osztalyok_szama << std ↩
           ::endl;
    }

    catch (const boost::filesystem::filesystem_error& ex)
    {
        std::cout << ex.what() << '\n';
    }


    return a.exec();
}
```

Az iterator függvény feladata, hogy kiszámolja a Java osztályok számát. Ehhez paraméterként kap egy boost::filesystem::path típusú objektumot, mely tartalmazza a elérési útját a src-nek. Az osztalyok_szama változóban tároljuk el a megtalált osztályok számát. Az expr objektum egy reguláris kifejezést tartalmaz, azokat a fájlokra illeszkedik, amik .java-ra végződnek. Majd a for ciklus segítségével bejárjuk az src mappát. Ehhez a recursive_directory_iterator -t használjuk, melynek segítségével az almappákat is be tudjuk járni. Ha csak a sima directory_iterator-t használnánk, akkor rekurzívan kéne meghívnunk a függvényünket újra és újra. Csak akkor növeljük a megtalált osztályok számát, ha illeszkedik a fájl neve a expr kiefejezésünkre, és maga a fájl egy hagyományos fájl,nem pedig mappa. A ciklus végeztével visszaadjuk a talált osztályok számát.

A main függvényben pedig p objektumban eltároljuk az átadott útvonatal. Majd meghívjuk a korábban tárgyalt iterator függvényt, és kiíratjuk a Java JDK osztályok számát terminálba.

## 16.2. Másoló-mozgató szemantika

Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékadásra!

Az LZWBinFa programot fogom átnézni kiegészítve másoló és mozgató értékadással.

Elsőnek tisztázzuk azt, hogy mikor hívódik meg a konstruktor, és mikor az értékadás.

```
LZWBinFa binFa = binFa_regi; //konstruktor (másoló)
LZWBinFa binFa2;
binFa2 = binFa_regi; //értékadás (másoló)
```

Amikor az objektum létrehozásakor adunk értéket, akkor a mozgató/másoló konstruktor hívódik meg, ha pedig egy már inicializált objektumnak adunk meg új értéket, akkor pedig a másoló/mozgató értékadás.

Lássuk, hogy hogyan kell elkészíteni a másoló konstruktort. Mivel konstruktorról beszélünk, ezért a neve megegyezik az osztály nevével. A vezérlés akkor adódik át erre a konstruktorra, ha magával megegyező típusú referenciát adunk át.

A kód:

```
LZWBinFa::LZWBinFa (const LZWBinFa & forras){
  std::cout << "Copy ctor" << std::endl;
  gyoker = new Csomopont('/');
  gyoker->ujEgyesGyermek(masol(forras.gyoker->
    egyesGyermek(), forras.fa));
  gyoker->ujNullasGyermek(masol(forras.gyoker->
    nullasGyermek(), forras.fa));
  if (forras.fa == forras.gyoker){
    fa = gyoker;
  }
}
```

Látható, hogy ez a konstruktor egy LZWBinFa konstans refernciát vár paraméterként. Mivel ilyenkor nem hívódik meg az alap konstruktor, emiatt a gyöker pointerünknek értéket kell adni, különben memóriacímzési hibát kaphatunk. Ha ezzel kész vagyunk, akkor létre kell hozni a gyökértol kiindulva az egyes gyermekeket, ezért van szükség a ujEgyesGyermek és ujNullasGyermek függvényekre. Mivel egy már elkészített fából indulunk ki, ezért annak a mintájára készítjük el a gyermekeket. Ebben a másol függvény van segítségünkre. Ennek két paramétere van, az egyik a forrás gyökerének a gyermeke, a másik pedig a forrás fa mutatója, mivel azt is szeretnénk átmásolni, hogy a fa mutató éppen hol áll az eredeti fában.

```
Csomopont* LZWBinFa::masol (Csomopont* elem, Csomopont* regi_fa){
    Csomopont* ujelem = nullptr;

    if (elem != nullptr){
      ujelem = new Csomopont (elem->getBetu());
      ujelem -> ujEgyesGyermek(masol(elem->egyesGyermek(), regi_fa));
      ujelem -> ujNullasGyermek(masol(elem->nullasGyermek (), regi_fa));
      if (regi_fa == elem){
```

```
        fa = ujelem;
      }
    }
  return ujelem;
}
```

A masol egy Csomopont mutatót ad vissza. Első lépésben létrehozzuk a később átadni kívánt csomópontot. Majd ellenőrizzük, hogy az eredeti f csomópontjának van-e értéke, vagy null pointer. Utóbbi esetben szimplán visszaadunk egy null pointert. Ellenkező esetben az ujelem pointernek átadunk egy az eredeti csomópont alapján inicializált csomóponthoz tartotó memóriacímet. Ezután újra meghívjuk a masol függvényt annak érdekében, hogy elkészítsük az ujelem gyermekeit. Ezt addig folytatjuk, ameddig az eredeti fa végére nem érünk. Abban az esetben, ha a régi fához tartozó fa mutató a masol függvénynek átadott elemre mutat, akkor az új fánk fa mutatóját ezen elem alapján létrehozott csomópontra állítjuk. Végezetül pedig visszatérünk a csomóponttal. Ahhoz, hogy a fa mutatót a gyoker-re is állíthassuk, a másoló konstruktorban is ellenorizzük, hogy hova mutat az eredeti fa mutatója.

Most, hogy megismerkedtünk a másoló konstruktorral, nézzük át röviden a másoló értékadást. Ezt is a masol függvényre alapozzuk.

```
LZWBinFa & LZWBinFa::operator= (const LZWBinFa & forras){
  std::cout << "Copy assaignement" << std::endl;

  gyoker->ujEgyesGyermek(masol(forras.gyoker-> egyesGyermek(), forras.fa));
  gyoker->ujNullasGyermek(masol(forras.gyoker-> nullasGyermek(), forras.fa) ↩
     );

  if (forras.fa == forras.gyoker){
    fa = gyoker;
  }
  return *this;
}
```

Lényegében egy operátor túlterheléséről van szó. Szemantikailag különbség a konstruktorhoz képest, hogy a másoló értékadásnak van visszatérési értéke, jelen esetben LZWBinFa referencia. Ennek megfelelően visszaadunk egy mutatót arról az objektumról, amely az egyenlőség bal oldalán volt. Ezen kívül a már ismert eljárást követjük.

A C++11 másoló szemantikájának megismerése után folytassuk a mozgató szemantikával, azon belül is a mozgató konstruktorral. A mozgató konstruktor paraméteréül egy jobbértékreferenciát vár.A feladat szerint a mozgató értékadásra kell alapoznunk.

```
LZWBinFa::LZWBinFa (LZWBinFa&& forras)
{
  std::cout<<"Move ctor\n";
```

```
  gyoker = nullptr;
  *this = std::move(forras); //ezzel kényszerítjük ki,hogy a mozgató ↩
     értékadást használja
}
```

Az alapkoncepciója az a mozgató értékadásra alapozásnak, hogy lényegében felcseréljük a két fa gyökér mutatójának értékét. Ennek érdekében az újonan létrehozni kívánt fa gyökerét null mutatóvá tesszük, majd meghívjuk a mozgató értékadást. Ehhez a std::move függvényyre van szükségünk, amely bemenetéül kapott paramétert jobbértékrefernciává alakítja. Mivel a this egy már inicializát objektumot jelöl, ezért nem a mozgató konstruktor, hanem a mozgató értékadás hívódik meg.

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

A mozgató értékadás feladata a már korábban említett érték csere, melyet a std::swap valósít meg.Mivel a megcserélődik a gyoker és a forras.gyoker által mutatott tárterület, ezért a mozgatás teljes egészében megvalósul. Hiszen a gyoker egy null mutató, emiatt a csere után a forras.gyoker is null mutató lesz, vagyis az eredeti fa "törlődött". Természetesen ez nem szó szerint igaz, hiszen csak egy másik pointeren keresztül hivatkozunk a már korábban a memóriában lefoglalt és tárolt fára.

## 16.3. Összefoglaló, A mozgató és másoló konstruktor

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutató „”esszé szöveget!

A másoló és mozgató konstruktorok segítségével objektumok inicializálását hajthatjuk végre, viszont egy másik azonos osztályú objektum alapján. Ha nem szeretnénk megírni ezeket, akkor a legegyszerűbb, ha letiltjuk használatukat, vagyis privát taggá tesszük oket. Erre azért van szükség, mert ha nem tiltjuk le őket, akkor a fordító az alapértelmezett másoló/mozgató konstruktort hívja meg, ami váratlan eredményekhez vezethet, kiszámíthatatlanná teszi a program működését.

A másoló konstruktor, ahogy a nevében is benne van egy objektumot másol le, és az alapján készít egy másikat. Onnan lehet felismerni, hogy paraméterként mindig az őt magába foglaló osztállyal azonos osztályú objektumrefenciát vár. Két fajta másolásról beszélhetünk: sekély és mély másolás. A sekély másolás lényege, hogy csak létrehozunk egy másik mutatót, ami a paraméterként megadott onjektumra mutat. Ennek implementálása a legegszerűbb, viszont használata problémákat okozhat. A legfőbb hátránya ennek a megoldási módnak, hogy közös a memóriaterület, tehát ha az egyiket módosítjuk, akkor az a másikra is kifejti hatását.Tehát ez nem egy igazi másolat, nevezhetnénk alias-nak. A mély másolás ezzel szemben egy különálló onjektumot hoz létre, mely megegyezik a paraméterül kapott elemmel, viszont külön memóriacímen. Sekély társához képest implementálása összetettebb, viszont magabiztos használata jobb C++ programozóvá tesz.

A 2. feladatban megvalósított másoló konstruktor a mély másolást implementálja. Tehát a forrásként kapott LZW fát bejárjuk, és minden elemének a másolatát elkészítjük a konstruálandó fában is. Ennek következtében lesz két külön fánk a meóriában, külön gyökér és fa mutatóval. Binárisfa bejárások közül létezik preorder, postorder és inorder. Ezek közül bármelyiket használhatjuk, mindegyik a megfelelő eredményhez vezet. A mi programunk a postorder bejárást alkalmazza, vagyis elsőnek dolgozzuk fel a gyemekeket, majd gyökeret.

A mozgató konstruktor a többi konstruktorhoz megfelelően egy inicializálást hajt végre. A másoláshoz hasonlóan ez is egy másik objektumra alapul. A különbség annyi, hogy a míg a másolásnál a másolásban résztvevő objektumok megmaradnak, addig a mozgatásnál a forrás objektum megszűnik, pontosabban nem determináns állapotba kerül. A mozgató konstruktor arról ismerszik meg, hogy a paramétere egy jobbérték referencia.Annak érdekében, hogy ez a konstruktor hívódjon meg, a std::move függvényt kell meghívni. Gyakori tévedés, köszönhetően ennek a függvénynek a félreérthető nevének, hogy mozgatást hajt végre. Valójában a paraméteréül kapott objektum jobbérték refernciájával tér vissza.

Az LZWBinFa programunkban maga a mozgató konstruktor a mozgató értékadásra van alapozva. Az alap koncepció az, hogy az inicializálni kívánt fa gyökerét null mutatóvá tesszük, majd megcseréljük a forrás és a cél fa gyökérmutatójának értékét, ezzel megvalósítva a mozgatást. Ezt úgy érjük el, hogy a mozgató konstruktoron keresztül meghívjuk a mozgató értékadást. Fontos látni, hogy mikor hívódik meg a konstruktor és mikor az értékadás. Ha egy még nem inicializát objektumot szeretnénk másolással, mozgatással inicializálni, akkor a konstruktor hívódik. Ezzel szemben egy már inicializát objektumba szeretnénk másolni, mozgatni, akkor már az értékadás hajtódik végre. Ennek tudatában szerepel a mozgató konstruktorban a következo sor:

```
*this=std::move(forras);
```

Szóval meghívódik a mozgató értékadás, ahol a pointer értékek cseréjét hajtjuk végre. Ehhez a std::swap függvényt használjuk. Természetesen lehetne saját cserét is írni, de összességében elmondható, hogy érdemes a már előre implementált függvényeket, metódusokat használni.

Összegezve a leírtakat, a másoló és mozgató szemantika a C++ nyelvben kiemelt szerepet játszik. Magabiztos használatuk elsajátítása nélkülözhetetlen a megbízható programok készítéséhez.

# 17. fejezet

# Helló, Gödel!

## 17.1. Gengszterek

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

Forrás:Tóth Bertalan - A C++11 nyelv új lehetoségeinek áttekintése

Az OOCWC (Robocar World Championship) egy páréves platform, melynek célja volt a forgalomirányítási algoritmusok kutatása és a robotautók terjedésének vizsgálata. A Robocar City Emulator lehetővé tette volna fejlesztők számára új modellek és elméletek tesztelését. A debreceni Justine prototípus gengsztereinek rendezése volt ezúttal a feladat: ehhez eloször meg kell néznünk a GitHub projektet.

github.com/nbatfai/robocar-emulator

A C++11-ben megjelenő lambda kifejezések lehetővé teszik, hogy egy- vagy többsoros névtelen függvényeket definiáljunk a forráskódban. Szerkezetük nem kötött.

Általános alakja:

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

A szögletes zárójelpár jelzi, hogy lambda kifejezés következik. A kerek zárójelpár a függvényhívást jelenti.Ezután jön a függvény törzse, mely return utasításából a fordító meghatározza a függvény értékét és típusát

A myshmclient.cpp-ben a gangsters vektort rendezzük lambda kifejezéssel. Ha x gengszter közelebb van az elkapott cop objektumhoz, mint y gengszter, akkor igaz értéket ad vissza. Rendezés után így a vektor elejére a rendorhöz legközelebb álló gengszterek kerülnek.

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

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

## 17.2. C++11 Custom Allocator

https://prezi.com/jvvbytkwgsxj/high-level-programming-languages-2-c11-allocators/ a CustomAlloc-os példa, lásd C forrást az UDPROG repóban!

A konténerek allokátorokat használnak belső adatstruktúrájuk helyfoglalásához. Nincs éles definíció, aki ilyesmit csinál, az allokátor. A sztenderd könyvtár alapértelmezett allokátora:

```
std::allocator<T>
```

Vagyis ha nem definiálunk saját az-az custom allokátort, akkor minden standard library container (tároló,konténer), ezt fogja használni.

```
#include <stddef.h>
#include <cxxabi.h>
#include <iostream>
#include <vector>

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;

  CustomAlloc() {}
  CustomAlloc ( const CustomAlloc & ) {}
  ~CustomAlloc() {}

  pointer allocate ( size_type n ) {
    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
      << std::endl;
    free ( p );
    return reinterpret_cast<T*> (
      new char[n*sizeof ( T )] );
  }

  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;
  }
};

int main(){

std::vector<int, CustomAlloc<int>> v;

v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
v.push_back(5);
v.push_back(6);
v.push_back(7);
v.push_back(8);
v.push_back(9);
v.push_back(10);
v.push_back(11);
v.push_back(12);
v.push_back(13);
v.push_back(14);
v.push_back(15);

  for(int x : v){
    std::cout << x << std::endl;
  }
  return 0;
}
```

Látható hogy classunk egy template class, ami azt jelenti hogy bármely típusal használható, hogy milyen típussal szeretnénk használni, az osztály neve utáni relációsjelek között tudjuk megadni, például:

```
CustomAlloc<int> xy;
```

Nézzük akkor a kódot részenként:

```
  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;
```

Az using kucsszóval adjuk meg hogy mit milyen néven szeretnénk ezentúl használni, például innentől kezdve ha azt írjuk a programkódba hogy pointer az T*-ot fog jelenteni.

```
  CustomAlloc() {}
  CustomAlloc ( const CustomAlloc & ) {}
  ~CustomAlloc() {}
```

Konstruktorok és destrunktor, nem definiáljuk felül az alapértelmezett viselkedésüket.

```
  pointer allocate ( size_type n ) {
    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
      << std::endl;
    free ( p );
    return reinterpret_cast<T*> (
      new char[n*sizeof ( T )] );
  }
```

Az allocate nevű funkció ami egy pointer (az-az T*-ot) ad vissza valamint egy size_type tipusú paramétere van. Ez a funkció fogja végezni a szükséges memóriaterület lefoglalását.

```
    char* p=abi::__cxa_demangle ( typeid ( T ).name(),0,0, &s);
```

Itt újdonság a __cxa_demangle funkció az abi névtérbol, működésének megértéséhez szükséges, hogy tudjuk mi is az a name mangling. A name mangling-et a fordító végzi, ennek során kódolja a funkció és változó neveket egyedi nevekre, hogy a linker meg tudja különböztetni az általános neveket a nyelvben. Az ilyen nevek visszafejtésére használjuk a __cxa_demangle funkciót ami a demangolt névre mutató pointerrel tér vissza. A funkció elso paramétere az a név amit szeretnénk visszafejteni, második paraméter az output buffer, harmadik a hossz, a negyedik pedig a státusz, ezt tároljuk el a korábban definiált int s változónkban.

```
    std::cout << "Allocating "
      << n << " objects of "
      << n+sizeof ( T )
      << " bytes. "
      << typeid ( T ).name() << "=" << p
      << std::endl;
```

Nyomonkövetjük a memóriafoglalást, kiíratjuk mennyi objektumnak foglalunk helyet, hány bájtnyi méretben. A typeid ( T ).name() visszadja egy típus name mangolt tipusazonosítóját, ezt fejtjük vissza és tároljuk p-ben.

```
    free ( p );
    return reinterpret_cast<T*> (
      new char[n*sizeof ( T )] );
  }
```

Ezután szabadítjuk a p pointert, és viszatérünk egy T típusú pointerrel ami a lefoglalt területre mutat.

```
  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;
  }
```

A deallocate funkció végzi a lefoglalt terület felszabadítását, valamint itt is nyomkövetést végzünk.

```
int main(){

std::vector<int, CustomAlloc<int>> v;

v.push_back(1);
v.push_back(2);
v.push_back(3);
```

```
v.push_back(4);
v.push_back(5);
v.push_back(6);
v.push_back(7);
v.push_back(8);
v.push_back(9);
v.push_back(10);
v.push_back(11);
v.push_back(12);
v.push_back(13);
v.push_back(14);
v.push_back(15);

  for(int x : v){
    std::cout << x << std::endl;
  }
  return 0;
}
```

A main, deklarálunk egy int vektort aminél megadjuk, hogy a mi allokátorunkat használja, ezután beleteszünk a v.push_back() használatával 15 db int-et, majd végig megyünk a vektor minden elemén egy for ciklussal és kiírjuk az elemet

## 17.3. Alternatív Tabella rendezése

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

Az Alternatív Tabella célja a focibajnokságokon belüli csapatok teljesítményére vonatkozó sorba rendezése, amit nem csupán a győzelmek, vereségek és döntetlenek aránya alapján dönt el, hanem az alapján is, hogy azok a csapatok, melyek ellen az adott csapat játszik, mennyire erősek, tehát erősebb csapat ellen többet ér a gyozelem.

Ezt a Google PageRank algoritmusához hasonló módon éri el, amit az előző félévben már láttunk.

Tekintsük a forrást:

```
public static void rendezCsapatok(String[] s, double h[], String[] e, int  ↩
   ep[]) {

System.out.println("\nCsapatok rendezve:\n");

int csapatNevekMeret = h.length;

Csapat csapatok[] = new Csapat[csapatNevekMeret];

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

```
csapatok[i] = new Csapat(s[i], h[i]);
}

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;1
  }

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

Létrehozzuk a Csapat osztályt, ezeknek van neve, értéke, és egy compareTo függvénye, ami két csapat értékének összehasonlítására szolgál, és implementálja a Comparable interface-t.A Csapat objektumok összehasonlításához szükséges ez az interface, ugyanis mivel ezeket rendezni kívánjuk, és a java.util.Collections.sort() függvény csak olyan listákra alkalmazható, amiknek elemei a Comparable interface-t alkalmazzák.

Ez azért van, mert ha nem "összehasonlítható", azaz nem implementálja a Comparable interface-t az egyik listabeli elem, akkor a rendező algoritmus nem lenne képes megmondani, hogy annak épp egy másik elem előtt vagy után kéne lennie, mert ezt a compareTo függvénnyel teszi meg a működő esetben.

# 18. fejezet

# Helló, !

## 18.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! https://github.com/nbatfai/future/tree/master/cs/F6
Itt láthatjuk működésben az alapot: https://www.twitch.tv/videos/222879467

A feladat megoldásához először találnunk kellett egy hibát a program működése közben. Próbálgattam a programot és észrevettem hogy egy mappán belül csak egy altevékenységet tudunk létrehozni. Ha szeretnén többet létrehozni a programban semmi változást nem látunk a terminálban pedig hibaüzenetet kapunk.

A hiba javításához először megkerestem a kódban azt a helyet ahol az új altevékenység létrehozása történik.

```
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
  // rr.println("Cannot create " + f.getPath())rr.println("Cannot create "  ↩
     +
  // f.getPath())rr.println("Cannot create " + f.getPath())rr.println(" ↩
     Cannot
  // create " + f.getPath()) = new javafx.scene.control.TreeItem<java.io. ↩
     File>(f,
  // new javafx.scene.image.ImageView(actIcon));
  = new FileTreeItem(f, new javafx.scene.image.ImageView(actIcon));
  getTreeItem().getChildren().add(newAct);
```

A java.io.File.mkdir()a fájl "elérési útvonalát" kell tartalmaznia a java.io.FIle nak és ha ezen meghívjuk az mkdir() tagfügvényt, létrehozza ezt a mappát. Viszont a doksi átnézése után megtudjuk, hogyha már létezik egy adott mappa az elérési útvonalon, az mkdir nem fog tudni új mappát létrehozni ugyan oda, ugyanazzal a névvel. Ezzel már meg is találtuk a hibánkat, hiszen a program nem változtat a néven, csupán új altvékenyésgnek nevezi el és megpróbálja létrehozni. Itt fogunk módosítani a programon, hogy sikeresen létre tudjunk hozni több új mappát:

```
int i = 1;
  while (true) {
    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
      // rr.println("Cannot create " + f.getPath())rr.println("Cannot  ↩
         create " +
      // f.getPath())rr.println("Cannot create " + f.getPath())rr.println(" ↩
         Cannot
      // create " + f.getPath()) = new javafx.scene.control.TreeItem<java. ↩
         io.File>(f,
      // new javafx.scene.image.ImageView(actIcon));
      = new FileTreeItem(f, new javafx.scene.image.ImageView(actIcon));
      getTreeItem().getChildren().add(newAct);
            break;
    } else {
      i++;
      System.err.println("Cannot create " + f.getPath());
    }
  }
});
```

Egy ciklusba helyezve a mappa készítését, szimplán hozzárakjuk az elérési útvonal végére az "i" változónk értékét, amivel azt számoljuk, hogy hányszor próbálkoztunk már mappát létrehozni. Amennyiben a mappa már létezik, újra kezdjük egyel nagyobb értékkel és azt rakjuk a mappa nevének a végére. Amennyiben a mappa sikeresen létrejött a ciklusból egy break utasítás segítségével kilépünk.

## 18.2. OOCWC Boost ASIO hálózatkezelése

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

Az OOCWC (Robocar World Championship) egy páréves platform, melynek célja volt a forgalomirányítási algoritmusok kutatása és a robotautók terjedésének vizsgálata. A Robocar City Emulator lehetővé tette volna fejlesztők számára új modellek és elméletek tesztelését. A debreceni Justine prototípus része a CarLexer, melynek `sscanf` függvényét kell feldolgoznunk.

A mellékelt fájl helyett, jobban tudjuk érzékeltetni a `std::sscanf` működését a kliens kód alapján, hiszen pontosan amiatt használjuk amit a fent látunk.

```
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});
    }
```

A `sscanf` függvény addig dolgozza fel a formázott stringből az adatokat, amíg meg nem kapja a Gangster négy argumentumát. A %n az olvasott karakterek számát rögzíti, az nn változóba kerülnek tehát az összesen beolvasottak száma. A data segítségével tudjuk olvasni a még beolvasatlan adatokat. Ezt a pointert a beolvasott karakterek méretével toljuk el, így a data+nn az olvasatlan rész elejére fog mutatni. Az <OK %d %u %u %u> alak teljesülése esetén egy gengszter összes adata beolvasásra került, tehát létrehozunk egy új objektumot és belehelyezzük a gangster vektorba.

## 18.3. BrainB

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

A BrainB feladata a tehetségkutatás az esportban, az egyes játékokban előforduló 'karakterelvesztést' fogja előidézni, a karaktervesztés akkor következik be amikor egyszerre annyi minden történik a képernyőn, hogy nem tudjuk már követni hogy hol is van a karakterünk. A program 10 percig fut, ezalatt az idő alatt az a feladatunk hogy a bal egérgombot lenyomva Samu Entropyn tartsuk az egeret. A program futás közben statisztikát készít az eredményeinkről amit a program végeztével megtekinthetünk.

A program Qt segítségével van elkészítve, ami egy keresztplatformos alkalmazás-keretrendszer, amit GUI-s alkalmazások, illetve nem GUI-s programok fejlesztésére használnak.

A program használatához a következők telepítése a szükséges:

```
sudo apt-get install libqt4-dev
sudo apt-get install opencv-data
sudo apt-get install libopencv-dev
```

A mappában, ami tartalmazza a forráskódokat kiadjuk a következő parancsot:

```
~/Qt/5.12.2/gcc_64/bin/qmake -project
```

Értelemszerűen, onnan indítjuk a qmake parancsot, ahova telepítve van a Qt, a -project kapcsolóval létrehozzuk a project fájlt. Ez a fájl fogja tartalmazni azokat az információkat amelyek a qmake számára szükségesek a program létrehozásához.

Majd erre a létrehozott projekt fájlra ismételten futtatjuk a qmake-t:

```
~/Qt/5.12.2/gcc_64/bin/qmake BrainB.pro
```

A qmake egy Makefile-t fog készíteni nekünk a projektfájlban található infomációk alapján, majd a "make" parancsot kiadva elkészül a futatható állományunk amit a következo paranccsal futathatunk:

```
./BrainB
```

A Qt-ben szignálok és slotok az objektumok közötti komunikációra használhatóak. A slot-signal mechanizmus a Qt egy meghatározó eleme, és valószínüleg a Qt e-miatt különbözik a legtöbb keretrendszertol.A GUI programozásban, ha egy elem megváltozik, gyakran szeretnénk hogy egy másik elem erről értesüljön. Még általánosabban szeretnénk hogy az összes objektum tudjon tudjon kommunikálni egymássalfüggetlenül a típusuktól. Például ha a felhasználó a bezár ikonra kattint akkor valószínüleg szeretnénk hogy az ablak close() funkciója meghívódjon. Ennek elérésére a Qt slot-signal mechanizmust használ. Egy signal kerül végrehatjásra, amikor egy bizonyos esemény bekövetkezik. A Qt-s eszközöknek van számos előre definiált szignálja, de ha szeretnénk származtathatunk belőlük saját osztályokat kiegészítve őket a saját szignáljainkkal. A slot egy funkció, ami egy megfelelő szignál jelzésére hívódik meg. A szignálokhoz hasonlóan a származtatott osztályokhoz hozzáadhatjuk a saját slot-jainkat ha szükséges.

Nézzünk a slot-signal mechanizmusra példát a megadott programból:

```
BrainBWin::BrainBWin ( int w, int h, QWidget *parent ) : QMainWindow (  ↩
   parent )
{
  // setWindowTitle(appName + " " + appVersion);
  // setFixedSize(QSize(w, h));

  statDir = appName + " " + appVersion + " - " + QDate::currentDate() . ↩
     toString() + QString::number ( QDateTime::  currentMSecsSinceEpoch() ) ↩
     ;

  brainBThread = new BrainBThread ( w, h - yshift );
  brainBThread->start();
  connect ( brainBThread, SIGNAL ( heroesChanged ( QImage, int, int ) ),
    this, SLOT ( updateHeroes ( QImage, int, int ) ) );
  connect ( brainBThread, SIGNAL ( endAndStats ( int ) ),
    this, SLOT ( endAndStats (int)));
}
```

A connect segítségével tudjuk összekötni a szignálokat a slotokkal. Itt az első paraméter a szignált küldő objektum, a második a szignál amit kezelni akarunk, a harmadik a szignált kezelő objektumra mutató pointer, a negyedik pedig az a slot amit szeretnénk hogy kezelje a kiváltott szignált.

Ennek megfelelően itt azt látjuk hogyha, a brainBThread objektum heroesChanged szignálja kiváltódik (brainBThread végzi a számításokat az ablakban megjelenő objektumok poziciójának kiszámításához, értékek kiszámítása után küldi a szignált, ez a szignál tartalmazni fogja a kiszámolt értékeket is), akkor ez az objektum BrainBWin (ami az ablakban szereplő objektumok megjelenítéséért felel) kezelje az updateHeroes slottal(updateHeroes átveszi a szignál által küldött értékeket és ennek megfelelően frissíti az ablakot).

A második connect-ben a brainBThread objektum endAndStats szignálját kezeljük, ami akkor váltódik ki ha lejár a program futási ideje, akkor ez az objektum (BrainBWin) kezelje az endAndStats slottal, ami bezárja az ablakot és befejezi a program működését.

Egy kép a program futásáról:1

# 19. fejezet

# Helló, Schwarzenegger!

## 19.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/ch01.html#id527287

A kivételkezelésről már többször szóesett ebben a könyvben. A Java nyelvben kiemelt szerepe van a ennek a hibakezelési metódusnak, ugyanis szerves része a nyelvnek. Sok esetben a Java VM nem is engedi fordítani a kódot, ha nem kezeljük a lehetséges kivételeket.

A kivétel a program végrehatása során keletkezik, mely megszakítja az utsaítások végrehajtását. A nem objektumorientált nyelvekben, mint a C, minden gyanús helyen külön kellett kezelnünk az egyes kivételeket, mely egy hibaüzenet kiírásából és a program leállításából állt. Ezzel szemben C++/Java-ban akár egy helyen is kezelhetjük őket. Az alap szintaxisa a objektum orientált nyelvekben a kivételkezelésnek a következő:

```
try{
    ....
}
catch(kivétel deklaráció) {
    ....
}
finally{
    ....
}
```

Tehát a vizsgálni kívánt kódrészt egyszerűen behelyezzük egy `try` blokkba, majd pedig tetszőleges számú `catch` blokk segítségével kezeljük a kivételeket. A `finally` blokk mindig meghívódik, függetlenül attól, hogy történt-e kivételdobás vagy sem. Például itt zárjuk be a megnyitott fájlokat, amire már nem lesz szükség. Ahhoz, hogy kivételkezléssel tudjunk foglalkozni, ahhoz kellenek kivételek, melyeket a metódusoka `throw` kulcszsóval teszik meg. Ezután vesszővel elválasztva tudjuk felsorolni a dobható kivételek típusát. A Java nyelv több osztályt biztosít a kivétlek kezeléséhez, melyek mind a `Throwable` osztály leszármazottai. De mi is hozhatunk létre saját osztályt ennek érdekében.

Most lássuk a feladatot. A tárgyalni kívánt kódrészlet a következő:

```
public class KapuSzkenner {
```

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

        for(int i=0; i<1024; ++i)

            try {

                java.net.Socket socket = new java.net.Socket(args[0], i);

                System.out.println(i + " figyeli");

                socket.close();

            } catch (Exception e) {

                System.out.println(i + " nem figyeli");

            }
    }

}
```

Ennek a kódrészletnek a futtatását a saját géped vizsgálatán kívül nem javasoljuk, mivel a Port scannelés nem teljesen legális dolog, de szemléltetésnek kiváló. Maga a program annyit csinál, hogy a prancssori argumentumként kapott gép 1024-nél kisebb portjahoz próbál kapcsolódni. Java-ban a kapcsolódás nagyon egyszerű, csak egy `Socket` típusú objektumra van szükségünk. Ennek a konstruktorába meg kell adni az IP-címet és a portot, ahova csatlakozni szeretnénk. Viszont mi történik akkor, ha nem tudunk csatlakozni a megadott porthoz? Ekkor a `socket` objektumunk konstruktora `IOException` típusú kivételt dob. Ha tudunk csatlakozni, akkor a cél gép adott portján egy szerver folyamat ül, ellenkező esetben pedig nem. Mivel dobhat kivételt a programunk, ezért azt kezeljük, tehát berakjuk egy `try-catch` blokkba. Ha nem dob hibát a csatlakozás során a program, akkor kiírjuk a terminálba, hogy az adott port figyelt, hiba esetén pedig azt, hogy nem figyelt. A kivételkezlés nélkül nem is tudjuk lefordítani a programunkat, ebből jól látszik, amit korábban említettem, hogy a kivételek kezelése a Java nyelv szerves része.

## 19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Ebben a feladatban az átszövés orientált programozással fogunk megismerkedni, melyhez az AspectJ programozási nyelvet fogjuk használni. Az átszövés lényegében annyit tesz, hogy a régebbi programunk működését úgy tudjuk módosítani, hogy nem kell módosítani a forráskódon. Helyette írunk egy szövést (aspectet), mellyel megváltoztathatjuk például egy függvény működését, vagy esetleg megadhatjuk, hogy a függvény végrehajtása előtt/után mit csináljon a program. Példák ebben a diasorban találhatóak: https://arato.inf.unideb batfai.norbert/UDPROG/deprecated/Prog2_6.ppt. Mi most az AspectJ nyelv `around` függvényével fogunk foglalkozni, melynek utasításait egy adott függvény helyett hajtjuk végre.

Mielőtt megnéznénk a kódot, előtte térjünk ki egy kicsit az aspektusok felépítésére. Tehát van a már korábbi fejezetben megírt `LZWBinFa.java` forrásunk, és ennek a függvényét szeretnénk módosítani. Az

aspektusok önállóan nem használhatóak, mindig kell egy program hozzá, AspectJ esetén Java. Majd létrehozunk egy aspektust, mely olyan, mint egy normál Java osztály. Összességében 3 fogalommal kell megismerkedni ezzel kapcsolatban. Az első a **kapcsolódási pont**, mely az eredeti programban van, vagyis az erdeti program egyik függvénye. A másik fontos fogalom az **vágási pont**, mely az aspektus része, és a csatlakozási pontokat tudjuk jelöljük vele. Végezetül pedig szükség van még **tanácsra**, mely lényegében azt tartalmazza, hogy hogyna szeretnénk módosítani az eredeti program működését.

Most azt nézzük meg, hogyan lehet megoldani azt, hogy a programunk a fát preorder módon járja be. Ezt már korábbi feladatokban megcsináltuk, viszont a akkor a forráskódot közvetlenül módosítottuk.

```
privileged aspect Aspect{
    void around(LZWBinFa fa, LZWBinFa.Csomopont elem, java.io. ↩
       BufferedWriter os):
        call(public void LZWBinFa.kiir(LZWBinFa.Csomopont, java.io. ↩
           BufferedWriter))
            && target(fa) && args(elem, os){
        if (elem != null)
        {
            try{
                ++fa.melyseg;
                for (int i = 0; i < fa.melyseg; ++i)
                    os.write("---");
                os.write(elem.getBetu () + "(" + (fa.melyseg - 1) + ")\n");
                fa.kiir(elem.egyesGyermek (), os);
                fa.kiir(elem.nullasGyermek (), os);
                --fa.melyseg;
            }
            catch(java.io.IOException e){
                System.out.println("Csomópont írása nem sikerült.");
            }
        }
    }
}
```

Láthatjuk, hogy szintaktikailag alig van különbség egy sima Java forráshoz képest. Az első kulcszsó, amit eddig nem láttunk, az a `privilaged`, melyre azért van szükség, hogy az aspektusunk hozzá tudjon férni az osztályok privát tagjaihoz. Az `aspect` kulcsszó jelöli, hogy most aspektust írunk, nem pedig hagyományos osztályt. Az aspektusunk egyetlen függvényből áll, ez a `around`. Ahhoz, hogy a bejárás módját módosítani tudjuk elsőnek a `kiir(Csomopont, BufferedWriter)` függvényre vany szükségünk. Mivel ez a függvény olyan tagváltozókat és tagfüggvényeket tartalmaz, melyek nem statikusak, ezért az `around` függvény paraméterének meg kell adnunk egy `LZWBinFa` objektumot, melyen keresztül ezeket a tagokat el tudjuk érni. Majd megadjuk, hogy az `around` melyik függvény helyett hívódjon meg, ehhez meg kell adnunk a teljes paraméterlistáját. Ezután && elválasztva megadjuk, hogy az `around` paraméterei közül melyiket szeretnénk paraméterként átadni a `kiir` függvénynek, és azt is, hogy melyik `LZWBinFa` objektumra szeretnénk végrehajtani a függvényt. Az előbbihez a `args`, utóbbihoz a `target` kulcsszót használjuk. Ezután következik a `around` függvény törzse, mely a már ismert kódrészletet tartalmazza.

**Program futtatása**

```
sudo apt install aspectj
ajc LZWBinFa.java Aspect.aj
java -cp /usr/share/java/aspectjrt.jar:. LZWBinFa input.txt -o ↩
   output.txt
```

A alap programunk a következő kimenetet adja a `input.txt`-ben található szövegre:

```
---------1(2)
------------0(3)
------------------1(5)
---------------0(4)
------1(1)
---------0(2)
------------0(3)
---------------0(4)
---/(0)
---------1(2)
------0(1)
---------0(2)
------------0(3)
depth = 5
mean = 3.5
var = 1.2909944487358056
```

Aspektus használatával pedig a következő:

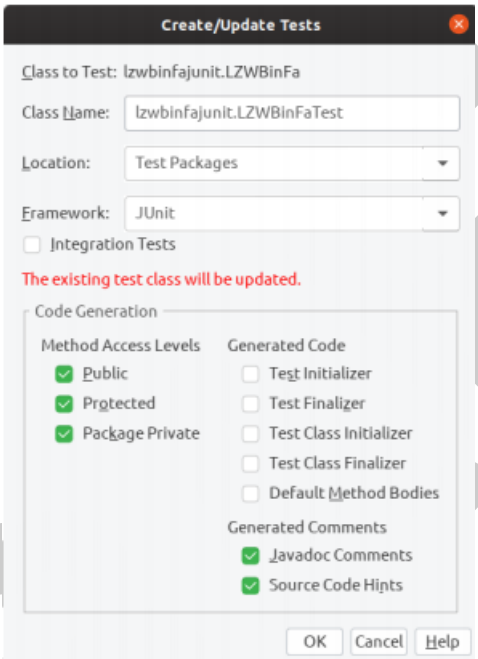
```
---/(0)
------1(1)
---------1(2)
------------0(3)
---------------0(4)
------------------1(5)
---------0(2)
------------0(3)
---------------0(4)
------0(1)
---------1(2)
---------0(2)
------------0(3)
depth = 5
mean = 3.5
var = 1.2909944487358056
```

Az aspektusok használatával a mindig a nekünk szükséges bejárási módot tudjuk használni. Összegezve, az aspektusok legnagyobb előnye, hogy nem szükséges komolyabban belenyúlnunk a régi kódunkba, ahhoz, hogy módosítsunk a működésén.

## 19.3. Junit teszt

A https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat poszt kézzel számított mélységét és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

A JUnit egy a Java nyelvhez kifejleszettet egységtesztelő keretrendszer. Segítségével automatizált módon tudjuk tesztelni, hogy a programunk a várt módon működik-e. Mi ebben a feladatban azt fogjuk leteszteltelni, hogy a mélység, a szórás és az átlag értékek megfelelnek-e annak, amit korábban már láthattunk. A NetBeans-ben létre kell hoznunk egy projektet, amibe belerakjuk a `LZWBinFa` osztály forrását, majd ha erre rákattintunk a Tools menüpontban létrehozunk egy tesztet.



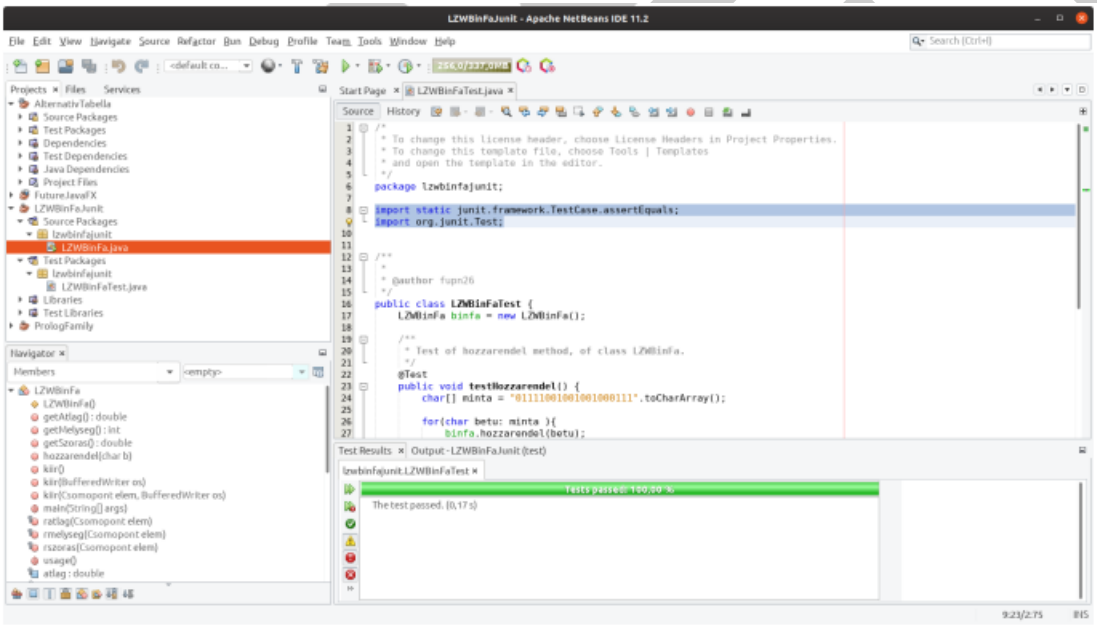

Ha elkészítettük a tesztet, akkor a "Test Libraries"-hoz hozzás kell adni a Junit 4.x könyvtárát. Majd nyissuk meg újra a projektet. Ezután pedig a forrásunkra kattintva jobb egér gombbal lesz egy olyan opció, hogy "Test file". Ha mindent jól csináltunk, akkor lefut a teszt hiba nélkül.

Nézzük meg a teszthez tartozó forrást.

```
import static junit.framework.TestCase.assertEquals;
import org.junit.Test;
public class LZWBinFaTest {
    LZWBinFa binfa = new LZWBinFa();

    @Test
    public void testHozzarendel() {
        char[] minta = "01111001001001000111".toCharArray();

        for(char betu: minta ){
            binfa.hozzarendel(betu);
        }

        assertEquals(4, binfa.getMelyseg());
        assertEquals(2.75, binfa.getAtlag(), 0.01);
        assertEquals(0.957427, binfa.getSzoras(), 0.000001);
```

```
    }

}
```

Szintaktikailag egy normál Java osztályt látunk. A különlegességét az adja, hogy tartalmazza a @Test makrót, melyel jelöljük, hogy ezután egy tesztelő függvényt definiálunk. Az `LZWBinFaTest` osztályban példányosítunk egy `LZWBinFa` típusú objektumot. Majd a `testHozzarendel` függvény segítségével teszteljük, hogy a `hozzarendel` függvény megfelelően építi-e fel a fát. A `minta` tömbben eltároljuk a honlapon található mintát, majd maszkolás nélkül belerakjuk az 1-eseket és a 0-ákat a fába. Ha ezzel végeztünk, akkor a `assertEquals` függvény segítségével ellenőrizzük, hogy azokat a szórás, mélység és átlag értékeket kaptuk, amire számítottunk. A forrásban ennek a függvénynek 2 verzióját használtuk. Az első esetben sima összehasonlítást végzünk, a többi esetben pedig megadunk egy delta számot, amivel eltérhet a várt érték a kapott értéktől. Erre azért van szükség, mert nem egész számokat hasonlítunk össze, és ebben az esetben így kell eljárni.



# 20. fejezet

# Helló, Calvin!

## 20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13/hello_s
bol Háttérként ezt vetítsük le: https://prezi.com/0u8ncvvoabcr/no-programming-programming/

Ebben a feladatban egy neurális hálót fogunk tanítani az MNIST adatbázisát felhasználva és a Softmax regressziót használva. A cél az, hogy a programunk fel tudja ismerni a képen látható számot. Maga a forráskód a hivatalos Tensorflow repo-jában lévő forráson alapszik, csak annyival let kiegészítve, hogy a saját kézzel írt számainkat is fel tudja ismerni.

Tehát az MNIST egy adatbázis, mely kézzel írt számokat ábrázoló képeket tartalmaz. Ezekhez a képekhez tartozik egy címke, ami megadja, hogy mit kéne látnia a képen a programnak. Szóval ennek az adatbázisnak a segítségével betanítjuk a neurális hálónkat, majd pedig a teszteljük. Lássuk a forrást.

```
from tensorflow.examples.tutorials.mnist import input_data
```

Elsőnek importáljuk a tanításhoz szükséges mintákat. Következő feladat a model létrehozása.

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

A `x` változó nem egy konkrét értéket fog tartalmazni, csak placeholder-ként funkcionál. Ez egy 2 dimenziós tensor lesz, melybe akárhány MNIST képet, melyeket 784 dimenziós vektorokká alakítunk, szeretnénk tárolni. A `W` fogja tartalmazni a súlyokat. Ennek típusa `Variable`, mely egy módosítható tensort jelent. Ugyanilyen típusú lesz a `b`, mely az eltolási értékeket fogja tartalmazni. De mind a kettő esetén igaz, hogy inicializálás során nullákkal töltjük fel. Mivel a `W`-t szeretnénk balról szorozni `x`-el, emiatt ez egy 784x10-es mátrix lesz. A szorzás erdeménye pedig egy 10 dimenziójú vektor lesz. Ehhez pedig hozzá tudjuk majd adni a `b`-t. Ezt az értéket adjuk át `y`-nak. Ahhoz, hogy tesztelni tudjuk a modellt, szükségünk van egy indikátorra. Az indikátor azt szokta muatatni, hogy a modell rossz, szokták veszteségnek is nevezni. A cél az, hogy ezt minimalizáljuk. Egy elterjed függvény a veszteség kiszámításra a `cross-entropy`. Tehát ezt a függvényt fogjuk végrehajtani az `y`-on. De ehhez szükség van még `y_`-ra.

```
y_ = tf.placeholder(tf.float32, [None, 10])
```

Az `y`-ba fogjuk tárolni az általunk kiszámított eloszlásokat, míg az `y_`-ban a valódi eloszlást. Ezután kiszámítjuk a `cross-entropy` értékét.

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

A `tf.nn.softmax_cross_entropy_with_logits` függvény kiszámolja a `cross entropy-t` `y` és `y_` értékek között. Majd a `tf.reduce_mean` kiszámolja az így keletkezett újabb vektorban lévő értékek átlagát. Még arra van szükség, hogy ezt az átlagot csökkentsük.

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

A gradient decent algoritmust használjuk, csökkentsük az átlagos veszteséget, és a 0.5-ös paraméter azt adja meg, hogy milyen lépésekkel közelítünk a minimum veszteséghez. Ezután szükségünk van egy metódusra, mely inicializálja változóinkat, és elindít egy `Session`-t. Ez egy olyan osztály, amivel Tensorflow műveleteket hajtunk végre.

```
sess = tf.InteractiveSession()
tf.initialize_all_variables().run(session=sess)
```

Ezután a már csak tanítanunk kell a modellt.

```
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Ezt a tanítást 1000-szer hatjuk végre. Minden alkalommal választunk random 100 batch-et a tanításra használt adatbázisunkból, melyet átadunk a változóinknak. Ezzel újból kiszámítunk egy `y`-ot. Majd ezzel és `y_` segítségével kiszámoljuk az átlagos veszteséget, és minimalizáljuk. Ha ezzel készen vagyunk, akkot teszteljük.

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_ ↩
   , 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf. ↩
   float32))
print("-- Pontossag: ", sess.run(accuracy, feed_dict={x:  ↩
   mnist.test.images,
                                                 y_: mnist.test.labels}) ↩
                                                    )
```

A `tf.argmax` függvény azt adja vissza, hogy az adott tensor adott tengelye mentén mi a legnagyobb értékű elem indexe. Ezt végrehajtjuk a várt értékeken és a kapottakon. Ha a két indexérték egyenlő, akkor helyesen következtette ki a modell a számot. A `tf.cast` függvény átalakítja az igaságértékünket lebegőpontos számmá. Majd adunk bementet és kiszámoljuk, hogy mennyire pontos a modell.

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

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

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

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

print("-- A sajat kezi 8-asom felismerese, mutatom a szamot, a ↩
   tovabblepeshez csukd be az ablakat")
img = readimg()
image = img.eval()
image = image.reshape(28*28)
matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm. ↩
   binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

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

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

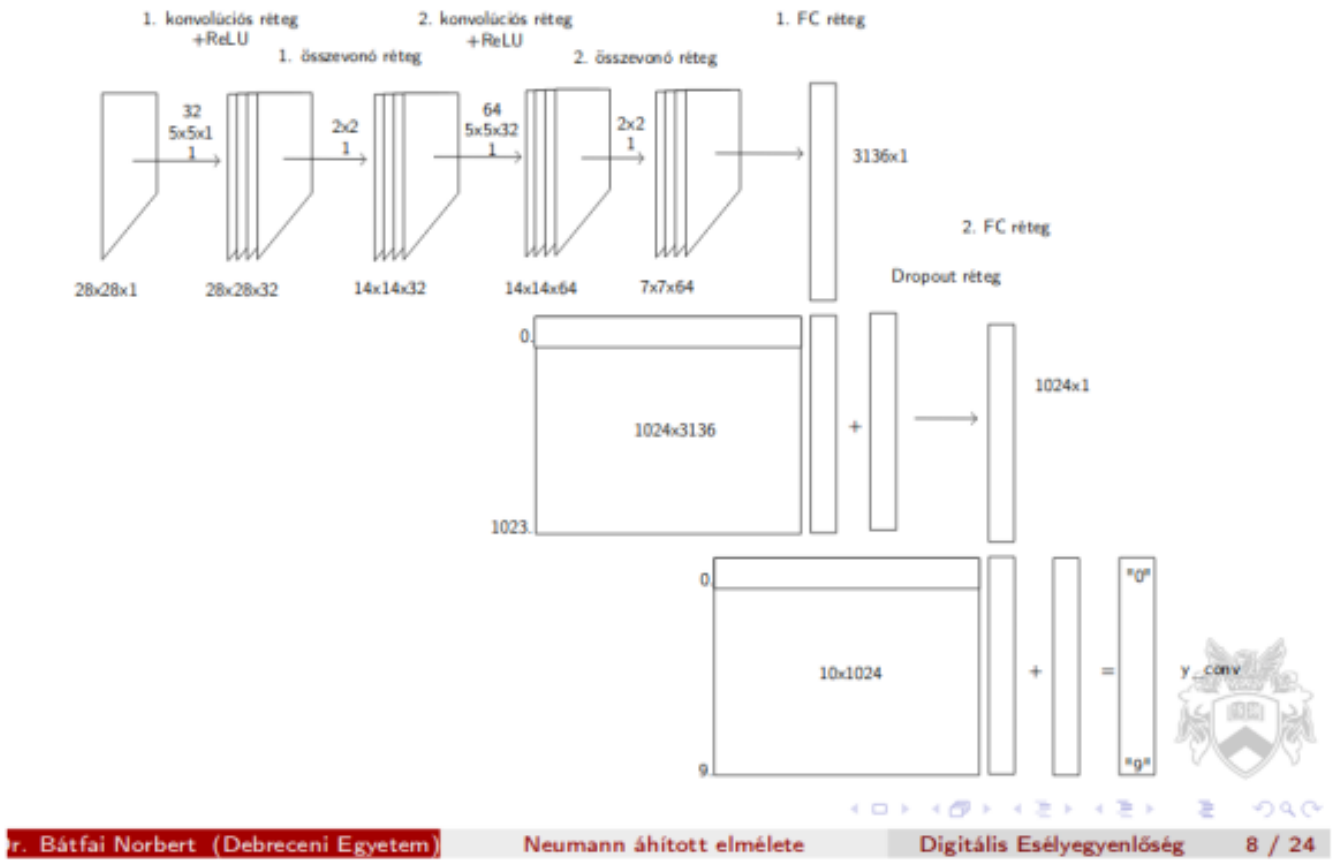
Az utolsó kódrészetben pedig már mi adunk meg képeket a modellnek, amit fel kell ismernie. A képet elsőnek megjelenítjük a felhasználó számára. Erre van a `matplotlib` könyvtár. Majd ha bezárja, akkor a proggram kiírja, hogy minek ismerte fel. A saját kézi 8-ast pedig a `readimg` függvénnyel olvassuk be.

```
def readimg():
    file = tf.read_file("sajat8as.png")
    img = tf.image.decode_png(file, 1)
    return img
```

## 20.2. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a https://arato.inf.unideb.hu/batfai.norb 8. fóliáját!

Az előző feladatban megoldottuk, hogy a modellünk képes legyen felismerni a kézzel írt számokat. Gondolhatnánk, hogy egész jó eredménnyel, hiszen 92%-ban helyes megfejtést adott. Viszont ez egyáltalán nem egy jó érték, ezért is készítették el ennek mély változatát, mellyel 99.2%-osra növeljük a helyes megfejtések arányát. Egy konvolúciós neurális hálót fogunk elkészítni, melynek semantikus ábrája így néz ki:




Lássuk, hoygan épül fel a modellünk. Először is több súlyra és bias-re lesz szükségünk. A súlyokat kis zajszinttel inicializáljuk, hogy elkerüljük a szimmetria megtörését és a 0 grádienseket. Olyan neuronokat használunk, melynek aktivációs függvénye mindig a bemenet pozitív részét adja vissza, vagyis 0-át, ha a szám negatív, ellenben pedig önmagát. Emiatt érdemes pozitív bias-okat használni. A bias amúgy azt adja meg, hoyg mennyire térhet el a várt és a kapott érték egymástól. Az előbbeik alapján készítünk 2 függvényt:

```
def weight_variable(shape):
    """weight_variable generates a weight variable of a given shape."""
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)


def bias_variable(shape):
    """bias_variable generates a bias variable of a given shape."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

A `tf.truncated_normal` függvény a kapott alaknak megfelelő tensort feltölt random számokkal, melyek normlis eloszlásúak, és a szórásuk 0.1. A `tf.constant` pedig a megadott értékkel tölt fel egy tensort, melynek formája megegyezik a `shape` attribútummal. Tehát így kapjuk meg a kezdő súlyokkal és bias-okkal feltöltött tensorainket.

Ezután szükségünk van még 2 függvényre, mellyel a konvolúciót és az összevonást hajtjuk végre.

```
def conv2d(x, W):
    """conv2d returns a 2d convolution layer with full stride."""
```

```
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    """max_pool_2x2 downsamples a feature map by 2X."""
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')
```

A `tf.nn.conv2d` függvény a 4D-s bemenetből és a szűrőből egy 2D konvolúciót számol és visszaadja tensor formájában. Folyamat abból áll, hogy a szűrőként megadott tensort 2D-ssé alakítja, a bemenetből képrészleteket szed ki, és egy virtuális tensort készít. Minden egyes képrészlet esetén jobbról szorozza a szűrő mártixor a képrészlet vektorával. A padding értéke pedig azért "SAME", hogy a bemenettel megegyező méretű tensort kapjunk. A `tf.nn.max_pool` függvény pedig a bemenetként kapott tensoron maximum összevonást hajt végre, vagyis a a legnagyobb értékeket tartjuk csak meg.

Elkészíthatjük mostmár az első réteget. Ez egy konvolúcióból és egy maximum összevonásból fog állni. Mielőtt alkalmaznánk, a bementként kapott képet kell átalakítani.

```
def deepnn(x):
    x_image = tf.reshape(x, [-1,28,28,1])
    with tf.name_scope('reshape'):
    x_image = tf.reshape(x, [-1, 28, 28, 1])
# First convolutional layer - maps one grayscale image to 32 feature maps.
with tf.name_scope('conv1'):
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
# Pooling layer - downsamples by 2X.
with tf.name_scope('pool1'):
    h_pool1 = max_pool_2x2(h_conv1)
```

A súlyokat tartalmazó tensor a `W_conv1`, melynek alakja [5, 5, 1, 32]. Ebből az első kettő azt adja meg, hogy mekkora a képrészleteket viszgálunk, a harmadik a bemenő csatornák, a negyedik a kimenőekét adja meg. A konvolúció pedig minden 5x5-ös képrészletekhez 32 db összeget számít ki. A `b`-ben tároljuk a 32 db bias-t. Végül a végrehatjuk a konvolúciót a paraméterként kapott képre és a súlyokra, majd hozzáadjuk a bias-okat. A `tf.nn.relu` függvény pedig, nullát ad vissza, ha negatív az érték, és az értéket, ha pozitív. Ezután pedig alkalmazzuk az maximum összevonást a rétegünkön.

Mivel most mély neurális hálót készítünk, ezért több ilyan rétegünk is lesz. Folyatssuk tehát a másodikkal.

```
# Second convolutional layer -- maps 32 feature maps to 64.
with tf.name_scope('conv2'):
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
# Second pooling layer.
with tf.name_scope('pool2'):
    h_pool2 = max_pool_2x2(h_conv2)
```

A másodiik réteg az elsőből számítjuk ki. Most a súlyokat tartalmazó tensorunk 32-es bemenetet kap, és 64-eset ad vissza. Ebből következik, hogy a konvolúcia minden 5x5-ös képrészlethez 64 összeget számol ki. Ezután a lépés után a 28x28-as képünket átalakítottuk egy 7x7-essé alakítottuk.

Ezután egy 1024 neuronból álló, fully-connected (FC) réteget adunk hozzá a modellünkhöz. Ezzel pedig feldolgozzuk az egész képet.

```
with tf.name_scope('fc1'):
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

Ehhez a réteghez is kiszámoljuk a súlyokat és a bias-okat. Majd a `tf.reshape` függvénnyel pedig az `h_pool2` réteget ix7*7*64-es mátrixxá alakítjuk. Majd az így kapott réteget jobbról szorozzuk a súlyokat tartalmazó `W_fc1` mátrixxal. Ehhez hozzáadjuk a bias-okat tartalmazó `b_fc1` mátrixot, majd az így kapott mátrixon alkalmazzuk a `tf.nn.relu` függvényt. Mely a negatív számokból nullát csinál, a többit pedig hagyja.

Annak érdekében, hogy a csökkentsük a modellünk bonyolultságát. Lényegében a felesleges neuronokat eldobjuk. De annak érdekében, hogy ezeket lehessen tárolni, létrehozunk egy új `placeholder`-t.

```
with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

A `tf.nn.dropout` függvény pedig `keep_prob` valószínűséggel ejt ki neuronokat a `h_fc1` tensorból. A maradék neuronokat pedig `1 / (1-rate)` módon skálázza, vagy 0-át ad vissza. Erre azért van szükség, hogy a `h_fc1`-ből kiszámítható összeg ne változzon.

Az utolsó lépsét pedig már ismerjük, ez lesz a szoftmax réteg.

```
with tf.name_scope('fc2'):
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])

    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

Ezzel elkészítettük a mély neurális hálónkat, most jöhet a betanítás. Ebben az esetben a betanítás elég hosszú ideig tarthat, ezért, ha egyszer lefuttatjuk, akkor már érdemes el is meneteni a kapott súlyokat.

```
with tf.name_scope('loss'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                            logits=y_conv)
cross_entropy = tf.reduce_mean(cross_entropy)
with tf.name_scope('adam_optimizer'):
    train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)
graph_location = tempfile.mkdtemp()
    print('Saving graph to: %s' % graph_location)
    train_writer = tf.summary.FileWriter(graph_location)
    train_writer.add_graph(tf.get_default_graph())
```

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

Az utolsó két sorban látható, hogyan kell elmenetni a modellt. Látható, hogy a tanítás nagyjából megegyezik a Szoftmax-os példával, annyi különbséggel, hogy ebben a 20 000 lépést hajtunk végre, és a képekeen is 50-esével haladunk. A hivatalos forrsában nincs különválsztva, de mi szétszedtük 2 fájlra a programot. Az egyik az `mnist_deep_train.py`, ezt kell futtani elsőnek, onnantól pedig az `mnist_deep_eval.py` lehet használni, mely csak betölti a kiszámolt értékeket. Ezzel végére értünk a Deep MNIST-es feladatnak.

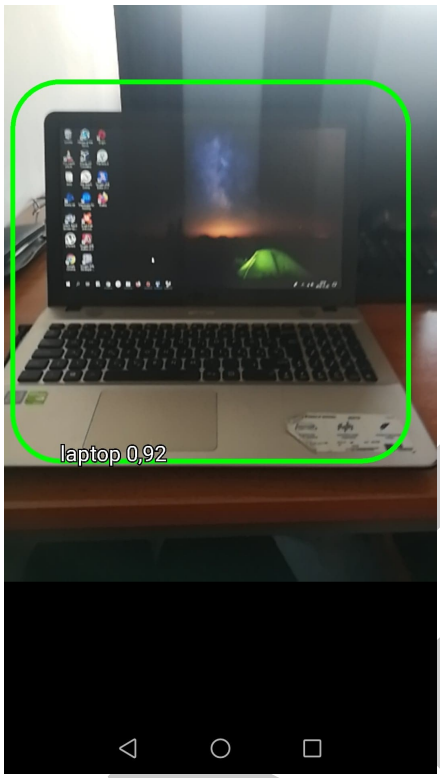
## 20.3. Android telefonra a TF objektum detektálója

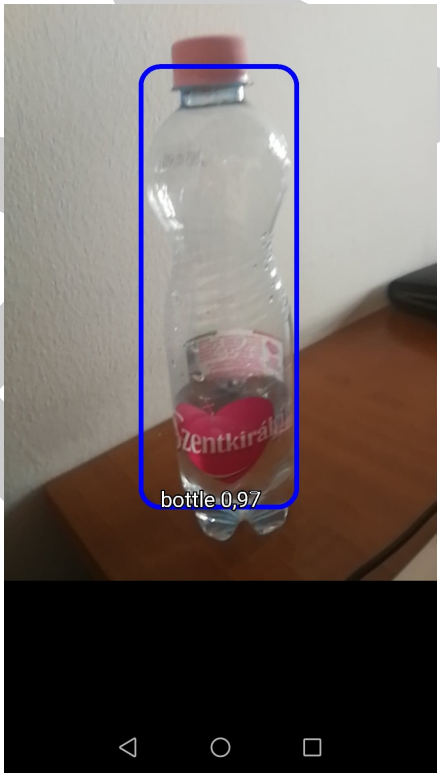
Telepítsük fel, próbáljuk ki!

A feladatom a Goggle által kifejlesztett Tensorflow kipróbálása, mely egy nyílt forráskódú könyvtár.

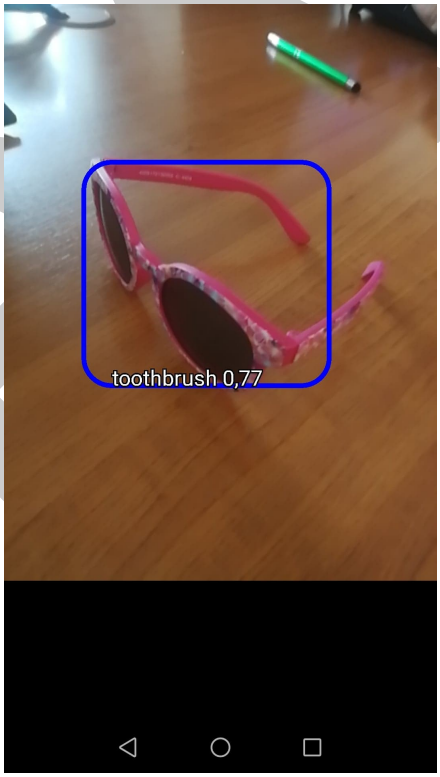
A TensorFlow képes mély ideghálózatokat tanítani és futtatni a kézzel írott számjegy-osztályozáshoz, képfelismeréshez, szóbeágyazásokhoz, ismétlődő neurális hálózatokhoz, szekvencia-sorozat modellekhez gépi fordításhoz, természetes nyelvfeldolgozáshoz és PDE (parciális differenciálegyenlet) alapú szimulációkhoz.

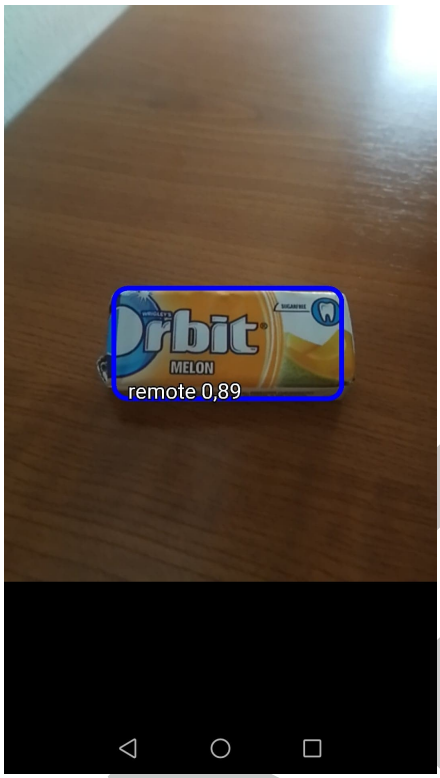
A program működés közben tárgyakat detektál. A próbálgatások során , a hétköznapi, egyszerűbb dolgokat sikeresen felismerte a program. Ilyeneka laptop vagy egy flakon víz.




De sok tárgyat nem sikerült felismerni, eddig ezekhez kevés adat lett betáplálva.



# IV. rész

# Irodalomjegyzék

## 20.4. Általános

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

## 20.5. C

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

## 20.6. C++

[BMECPP] Benedek, Zoltán Ã©s Levendovszky, Tihamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

## 20.7. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/-pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, https://groups.google.com/forum/#!forum/nemespor, az 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.