# Univerzális programozás

## Programkód vadászok

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

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* :<br><br>Univerzális programozás | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Bátfai, Norbert Ã©s Gilszki, Patrik | 2019. május 10. | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| 0.0.1 | 2019-02-12 | Az iniciális dokumentum szerkezetének kialakítása. | nbatfai |
| 0.0.2 | 2019-02-14 | Inciális feladatlisták összeállítása. | nbatfai |
| 0.0.3 | 2019-02-16 | Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába. | nbatfai |
| 0.0.4 | 2019-02-19 | Aktualizálás, javítások. | nbatfai |
| 0.0.5 | 2019-03-03 | Turing Fejezet | glszKK |
| 0.0.6 | 2019-03-10 | Chomsky Fejezet | glszKK |
| 0.0.7 | 2019-03-16 | Caesar Fejezet \| Hiányzás | glszKK |
| 0.0.8 | 2019-03-27 | Mandelbrot Fejezet | glszKK |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| 0.0.9 | 2019-04-04 | Welch Fejezet és Gutenber olvasó napló elkezdése | glszKK |
| 0.1.0 | 2019-04-13 | Conway Fejezet \| Gutenberg olvasónapló \| Hiányzás | glszKK |
| 0.1.1 | 2019-04-22 | Schwarzenegger Fejezet \| Chaitin Fejezet \| Gutenberg olvasónapló | glszKK |
| 0.1.2 | 2019-04-29 | Olvasónapló befejezése | glszKK |
| 0.1.3 | 2019-05-05 | Simítások | glszKK |
| 0.1.4 | 2019-05-10 | Könyv befejezése \| probléma a welch fejzettel \| Olvasónapló véletlen kitörlése, majd emlékezetből újraírása | glszKK |

# Ajánlás

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

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

# Tartalomjegyzék

**I. Bevezetés** 1

**1. Vízió** 2
1.1. Mi a programozás? . . . 2
1.2. Milyen doksikat olvassak el? . . . 2
1.3. Milyen filmeket nézzek meg? . . . 2

**II. Tematikus feladatok** 3

**2. Helló, Turing!** 5
2.1. Végtelen ciklus . . . 5
2.2. Lefagyott, nem fagyott, akkor most mi van? . . . 6
2.3. Változók értékének felcserélése . . . 7
2.4. Labdapattogás . . . 8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 10
2.6. Helló, Google! . . . 11
2.7. 100 éves a Brun tétel . . . 14
2.8. A Monty Hall probléma . . . 15

**3. Helló, Chomsky!** 17
3.1. Decimálisból unárisba átváltó Turing gép . . . 17
3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 18
3.3. Hivatkozási nyelv . . . 20
3.4. Saját lexikális elemző . . . 20
3.5. l33t.l . . . 21
3.6. A források olvasása . . . 23
3.7. Logikus . . . 24
3.8. Deklaráció . . . 25

**4. Helló, Caesar! 27**
4.1. int *** háromszögmátrix . . . 27
4.2. C EXOR titkosító . . . 29
4.3. Java EXOR titkosító . . . 30
4.4. C EXOR törő . . . 31
4.5. Neurális OR, AND és EXOR kapu . . . 34

**5. Helló, Mandelbrot! 38**
5.1. A Mandelbrot halmaz . . . 38
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 40
5.3. Biomorfok . . . 41
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 44
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 46
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 50

**6. Helló, Welch! 54**
6.1. Első osztályom . . . 54
6.2. LZW . . . 54
6.3. Fabejárás . . . 54
6.4. Tag a gyökér . . . 54
6.5. Mutató a gyökér . . . 55
6.6. Mozgató szemantika . . . 55

**7. Helló, Conway! 56**
7.1. Hangyaszimulációk . . . 56
7.2. Java életjáték . . . 59
7.3. Qt C++ életjáték . . . 65
7.4. BrainB Benchmark . . . 67

**8. Helló, Schwarzenegger! 70**
8.1. Szoftmax Py MNIST . . . 70
8.2. Mély MNIST . . . 71
8.3. Minecraft-MALMÖ . . . 75

**9. Helló, Chaitin! 82**
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 82
9.2. Gimp Scheme Script-fu: króm effekt . . . 83
9.3. Gimp Scheme Script-fu: név mandala . . . 86

**10. Helló, Gutenberg!** **91**

10.1. Juhász István - Magas szintű programozási nyelvek . . . . . 91

10.2. Kerninghan és Richie . . . . . 92

10.3. BME: Szoftverfejlesztés C++ nyelven / Benedek Zoltán, Levendovszky Tihamér . . . . . 93

**III. Második felvonás** **94**

**11. Helló, Arroway!** **96**

11.1. A BPP algoritmus Java megvalósítása . . . . . 96

11.2. Java osztályok a Pi-ben . . . . . 96

**IV. Irodalomjegyzék 2.0** **97**

11.3. Általános . . . . . 98

11.4. C . . . . . 98

11.5. C++ . . . . . 98

11.6. Lisp . . . . . 98

# 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

Megoldás forrása:https://github.com/glszKK/Turing/tree/master

Az első feladatban három egyszerű végtelen ciklust kell létrehoznunk, az elsőben egy olyat ami 0 százalékban dolgoztatja a magot:

```
#include <unistd.h>
            int
            main ()
            {
            for (;;)
            sleep (1);
            return 0;
            }
```

A második feladatban azt kell elérnünk hogy 100 százalékban dolgozzon a mag:

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

A feladat harmadik részében pedig azt kell elérnünk hogy a gépünk összes magja 100 százalékon dolgozzon, ehhez először utána kell járnunk annak hogy hány magos is a gépunk. Ha ez megvan utána neki kezdhetünk a feladatnak, ami annyiban változik a második feladat-tól hogy annyi új threadot kell nyitnunk ahány magos a számítógépünk.

```
#include <unistd.h>
           int main ()
           {
```

```
                    int p1, p2, p3;

                    if(!(p1 = fork()))
                    {
                        for( ;; );
                    if(!(p2 = fork()))
                    {
                        for( ;; );
                    if(!(p3 = fork()))
                    {
                        for( ;; );
                }
                for (;;){}
                return 0;
                }
```

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

A második feladatban azt kell megmutatnunk hogy tudunk-e olyan programot írni amely el tudja dönteni egy adott programról hogy le fog-e fagyni vagy sem.

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

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

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

Ez a program úgy működik/működne hogy kap bemenetként egy másik programot amiről el kéne döntenie hogy működik e vagy nem sem, tehát lefagy. Ez mind szép és jó hisz nagyon megkönnytené a programozók életét, de van egy nagy hibája. Ha saját magát adjuk meg bemenetnek, mert ha végtelen ciklus nincs a bemenetben akkor oda kerül, ha pedig van benne akkor megfog állni. Ezért nem működne helyesen a program.

## 2.3. Változók értékének felcserélése

Harmadik feladat, itt egy olyan C programra lesz szükségünk amely felcseréli két változó értékét, bármiféle logika utasítás vagy kifejezés használata nélkül.

Megoldás forrása:https://github.com/glszKK/Turing/tree/master

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

```
#include<stdio.h>
int main()

{

            int valtozo_1 = 2,valtozo_2 = 4;
 printf("valtozo_1=%d valtozo_2=%d\n",valtozo_1,valtozo_2);

  valtozo_1 = ( valtozo_1 - valtozo_2 );
  valtozo_2 = ( valtozo_1 + valtozo_2 );
  valtozo_1 = ( valtozo_2 - valtozo_1 );
 printf("valtozo_1=%d valtozo_2=%d\n",valtozo_1, valtozo_2);
return 0;
}
```

Megoldás forrása segédváltozóval:

```
int main()
{
  int v1=1, v2=2, v3;
  v3=v1;
  v1=v2;
  v2=v3;
  return 0;
}
```

A feladatban megvolt adva hogy nem használhatunk a program elkészítésekor logikai utasításokat, ami azt jelenti hogy matematikai egyenletekkel kell megoldanunk ezt a feladatot. Nem kell megijedni nem deriválás vagy hasonló nehézségű egyenletekről van itt szó, csak egyszerű alapműveletekről.(szorzás,osztás,kivonás,összeac
A feladatunk tehát az hogy két változó értékét felcseréljük tehát tegyük fel hogy az a az 20 a b pedig 2. a=20+2=22 -> b=22-2=20 -> a=22-20=2 tehát a-ra megkaptuk a b értékét tehát a kettőt. Ezt még meglehet oldani az XOR-al is ami ugyan úgy felcseréli a változókat csak a bitek segítségével.

## 2.4. Labdapattogás

A negyedik feladatban legy olyan programot kell létrehoznunk mely egy labdát pattogtat a karakteres konzolon, először if-ekkel, majd utána bármi féle logika utasítás vagy kifejezés használata nélkül.

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

Megoldás forrása:https://github.com/glszKK/Turing/tree/master

```
#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;
}
```

Itt pedig ugyan az a labdapattogtatási program csak if nélkül

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

int
main (void)
```

```
{
    int xj = 0, xk = 0, yj = 0, yk = 0;
    int mx, my;

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

    for (;;)
    {
  getmaxyx(ablak, my, mx);
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;

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

        //clear ();

        mvprintw (abs (yj + (my - yk)),
                  abs (xj + (mx - xk)), "@");

        refresh ();
        usleep (150000);

    }
    return 0;
}
```

Két új header filet használunk ( curses és unistd ) amik segítségével tudjuk a kurzorunkat mozgatni a terminálban. Az első programunkat if-el a másodikat pedig if nélkül oldottuk meg. Az első feladatunkat egy ablak = initscr ()-el kezdünk ahol az ablakunk adatait adjuk meg ahol a labda pattogni fog, megkell adnunk a sorok számát és az oszlopok számát is. Végtelen ciklust használunk ami azt jelenti hogy egy adott sémán végtelen ideig fog menni a labdánk addig ameddig meg nem szakítjuk. Ha a programunkba beleírjuk a clear(); parancsot akkor törli a labdákat de ha nem írjuk oda egy folytonos labda mintát láthatunk. Az mvprint-el adjunk meg karakterünket, jelen esetben mivel egy labdáról van szó a karakterünk egy "o" betű lesz. Az usleep-el pedig a labda gyorsaságát adjuk meg, minél nagyobb számot írunk be annál lassabb lesz a labdánk.

## 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Az ötödik feladatban olyan programot kell írnunk amely azt nézi meg, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Azt a while ciklust kell használnunk amit Linus Torvalds a BogoMIPS rutinjában használtt.

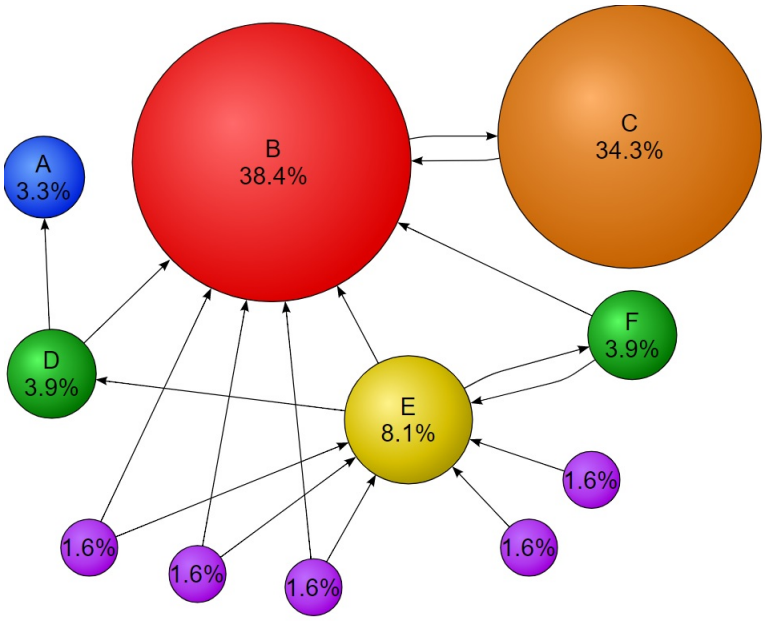
Megoldás forrása:https://github.com/glszKK/Turing/tree/master

```
#include <stdio.h>
int
main (void)
{
 int h = 0;
 int n = 0x01;
 do
   ++h;
 while (n <<= 1);
 printf ("A szohossz ezen a gepen: %d bites\n", h);
 return 0;
}
```

A MIPS jelentése az a "Millió utasítás másodpercenként". A bogoMIPS program melyet Linus Torvald talált ki, a gép elinditásakor megméri hogy meddig fut egy adott ciklus, de a nevéből is következtethetünk arra hogy itt valami nem stimmel. Hisz a bogo az angol bogus szóbol ered ami azt jelenti hogy nem igaz. Tehát itt is van egy kis probléma, már pedig az hogy nem egy pontos értéket hanem csak egy megközelítő értéket kapunk. Egy gépi szó változója 4byte ezért 32-őt fog eredményül adni. A célunk az hogy nullát kapjunk eredményül, tehát addig shifteljük a biteket balra, amig azt a nullát meg nem kapjuk.

## 2.6. Helló, Google!

Hatodik feladat, itt egy olyan programra lesz szükségünk, amely egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét. A programot Larry Page és Sergey Brin találták ki, akik amúgy a Google alapítói. A programmal egy weboldal fontosságát és értékét tudjuk megállapítani, az alapján hogy hány link mutat arra az oldalra amit ugye vizsgálunk. Minél több annál jobb! És az is számít hogy a linkeket milyen erősségű oldalon találjuk meg. És ez a PageRank nem a főoldaladra hivatkozott hanem aloldalanként működött. Azért csak működött mert 2013 óta nem frissült ez a PageRank érték, de valójában az emberek csak egy "gagyibb" változatát láthatták ennek a programnak, hisz megtudtuk egy 10-es skálán hogy mennyire erős az oldalaink PageRank száma, de ez a program ennél sokkal több. És a google-nak jóval több adatot és információt add át néhány számnál.




```
    #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){

  int i;
  double osszeg=0;

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

  return sqrt(osszeg);
}

void
pagerank(double T[4][4]){
  double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény
  double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0}; //ezzel szorzok

  int i, j;

  for(;;){
```

```
    // ide jön a mátrix művelet

    for (i=0; i<4; i++){
      PR[i]=0.0;
      for (j=0; j<4; j++){
        PR[i] = PR[i] + T[i][j]*PRv[j];
      }
    }

      if (tavolsag(PR,PRv,4) < 0.0000000001)
        break;

    // ide meg az átpakolás PR-ből PRv-be

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

  kiir (PR, 4);
}

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 L1[4][4] = {
    {0.0,  0.0,      1.0/3.0,  0.0},
    {1.0,  1.0/2.0,  1.0/3.0,  0.0},
    {0.0,  1.0/2.0,  0.0,      0.0},
    {0.0,  0.0,    1.0/3.0,  0.0}
  };

  double L2[4][4] = {
    {0.0,  0.0,      1.0/3.0,  0.0},
    {1.0,  1.0/2.0,  1.0/3.0,  0.0},
    {0.0,  1.0/2.0,  0.0,      0.0},
    {0.0,  0.0,    1.0/3.0,  1.0}
  };

  printf("\nAz eredeti mátrix értékeivel történő futás:\n");
  pagerank(L);

  printf("\nAmikor az egyik oldal semmire sem mutat:\n");
  pagerank(L1);
```

```
  printf("\nAmikor az egyik oldal csak magára mutat:\n");
  pagerank(L2);

  printf("\n");

  return 0;
}
```

Megoldás forrása: https://github.com/glszKK/Turing/tree/master

## 2.7. 100 éves a Brun tétel

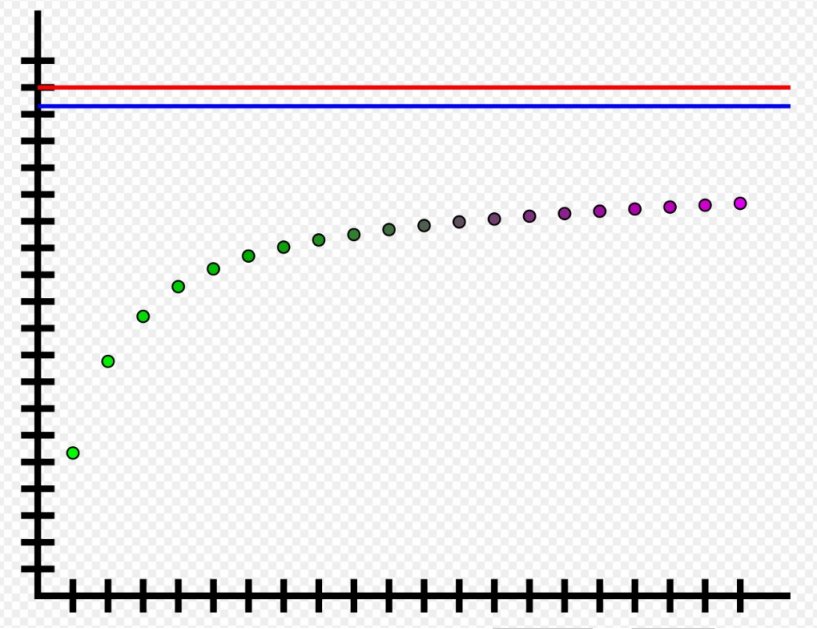
A hetedik feladatban R szimulációt kell írnunk a Brun tétel demonstrálására.

```
 stp <- function(x){

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

x=seq(10, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

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

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

A programunk a prímszámokkal foglalkozik azonbelül is az ikerprímekkel, ugye a prímszámok azok a számok amik egyel és önmagukkal oszthatóak, ikerprímnek pedig azt a két prímszámot nevezzük amik 2-vel térnek el egymástól. És azt tudjuk hogy a prímszámokból végtelensok létezik, de azt nem tudjuk hogy az ikerprímekből is végtelen számú van e. A prímek a végtelenbe tartanak ezért feltételezzük azt hogy az ikerprímek is, de amit elkezdjük őket vizsgálni úgy veszzük észre mintha egy adott szám felé tartanának. A Brun tétel azt mondja ki hogy az ikerprímek reciprokösszege egy véges értékhez konvergál, amit Brun-konstansnak nevezünk.



## 2.8. A Monty Hall probléma

Az utolsó feladatban R szimulációt kell írnunk a Monty Hall problémára.

```
kiserletek_szama=100
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)
```

A Monty Hall címet a program egy műsorvezetőről kapta aki a Lets make a deal című műsör volt. Aminek az volt a lényege hogy a játékosoknak 3 ajtó közül kellett választaniuk az egyik ajtó mögött egy óriási nyeremény lapult, a másik két ajtó mögött meg értéktelen nyeremények. Ez mind szép és jó hisz azt hiszi az ember hogy 3 ajtó közül nem is olyan nehéz eltalálni azt ami mögött a főnyeremény lapul hisz erre 33.3% esélyünk lenne. Ám itt jön a csavar, miután a játékos választott egy ajtót de mielőtt azt tényleg kinyitnák a műsorvezető ki nyit a maradék kettő közül választ egyet amelyik mögött nem a főnyeremény lapul hisz ő tudja hogy melyik ajtó mögött mi van. És ilyenkor az adott játékos módosíthatja a választását. A nagy kérdés az hogy megéri-e változtatni vagy sem? A válasz egyszerű, igen. Hisz amikor 3 ajtó közül kell választanuk, mint már mondtam 33% esélyünk van arra hogy a jó ajtót választottuk, viszont miután a műsorvezető kinyit egy ajtót az esélyünk a duplájára növekszik, hisz igy csak két ajtó közül kell választanunk. A programunk pedig ezeket a lehetőségeket vizsgálja meg.


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

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

# 3. fejezet

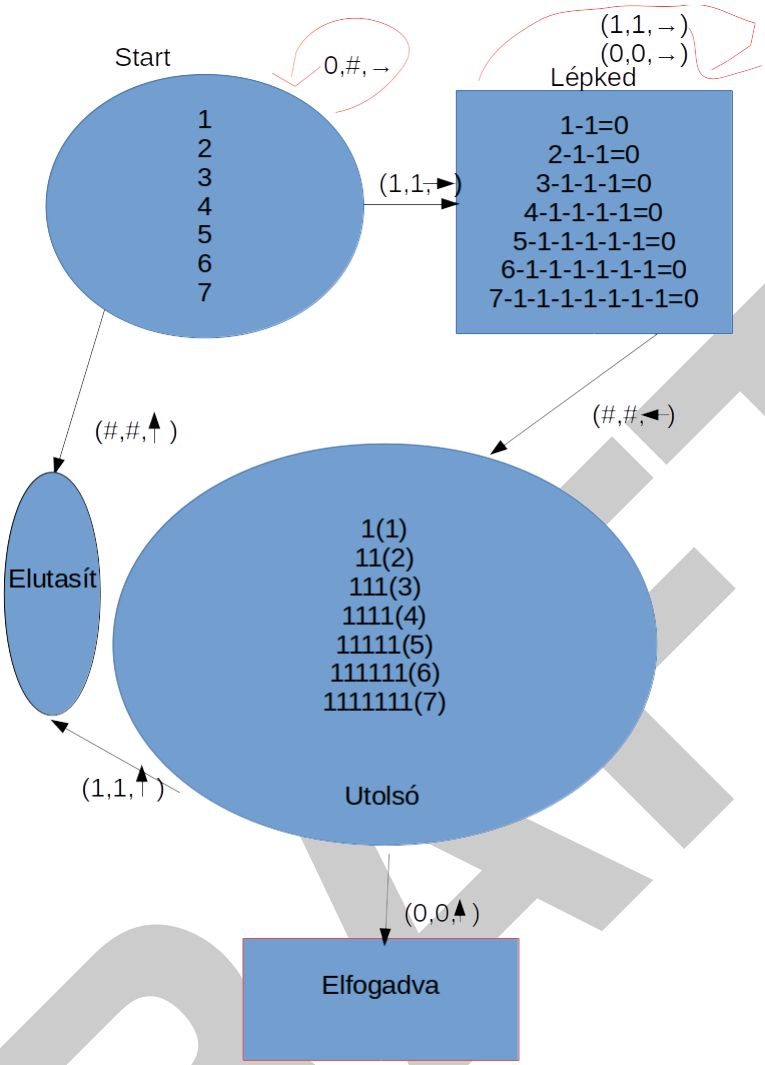
# Helló, Chomsky!

## 3.1. Decimálisból unárisba átváltó Turing gép

A harmadik fejezet első feladatában mint már a címben is látható, Decimálisból unárisba átváltó Turing gépet kell létrehoznunk, állapotátmenet gráfjával megadva.

Megoldás forrása: https://github.com/glszKK/Chomsky/tree/master

```
#include <stdio.h>
int main()
{
  int x;
  printf("Tizes szamrendszerbeli szam:");
  scanf("%d",&x);
  printf("\nUnaris alakja:");
  for(int i = 0; i < x; i++)
    {
      printf("/");
    }
  printf("\n");
  return 0;
}
```




A feladatban az unáris számrendszerről lesz szó, ami ugye tudjuk az egyes számrendszer. A programunk lényege hogy decimálisból unárisba váltas át a számokat. Mindez úgy történik hogy folyamatosan kivonunk 1-et az átváltani kívánt számból és ezeket a kivont 1-eseket tároljuk. Addig még 0-át nem kapunk eredményül. Ehhez egy for ciklust használunk.

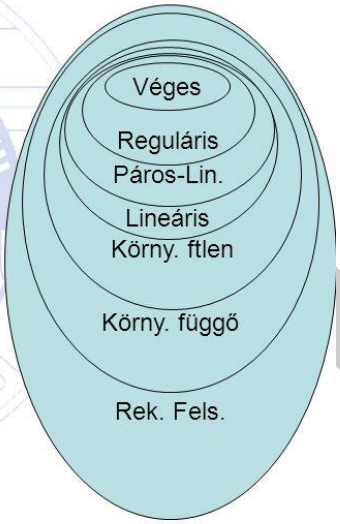
## 3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

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

Megoldás forrása: https://github.com/glszKK/Chomsky/tree/master

# Chomsky hierarchia

- Véges nyelvek
- Reguláris nyelvek
- Páros-lineáris nyelvek
- Lineáris(-környftl.) nyelvek
- Környezetfüggetlen ny.
- Környezetfüggő nyelvek
- Rekurzívan felsorolható ny.




A generatív nyelvtan kitalálója, a fejezetünk névadója volt: Noam Chomsky. A neve ismert lehet még a Chomsky-hierarchiából is, ezt fentebb szúrtam be. A generatív nyelvtan a formális nyelvek közé tartozik aminek 2 másik fajtája is van, a környezetfüggetlen nyelvtan és a szabályos nyelvtan. Igy 3-an alkotják meg a Formális nyelveket. Mivel a feladatunk az hogy két példát mutassunk be a generatív nyelvre így allíthatjuk hogy nem környezettfüggetlen nyelvről beszélünk.

```
  S -> aBSc
S -> abc
Ba -> aB
Bb -> bb
S -> aBSc -> aBaBScc -> aBaBabccc -> aaBBabccc -> aaBaBbccc ->  ↩
   aaaBBbccc ->  -
aaaBbbccc -> aaabbbccc
```

```
         S -> abc
S -> aXbc
Xb -> bX
Xc -> Ybcc
bY -> Yb
aY -> aaX
aY -> aa
S -> aXbc -> abXc -> abYbcc -> aYbbcc->aXbbcc -> aabXbcc-> aabbXcc->  ↩
   aabbYbccc -> aaYbbccc -> aaYbbbccc->aaabbbccc
```

## 3.3. Hivatkozási nyelv

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

Megoldás forrása: https://github.com/glszKK/Chomsky/tree/master

Ebben a részben a C nyelvről lesz szó. Mint tudjuk hisi nekünk is olvasnunk kell, létezik egy referencia könyv ahol a C nyelvhesz szükséges információkat és utasításokat megtaláljuk. Ilyenek például a szabványok. Amik igazából mindig valami új featurest hoznak be a C nyelvbe. Csak a probléma az hogy visszafele nem futnak és nem működnek az újabb szabványok. Például van a C99 szabvány, amiben már lehet dekralálni viszont a C89-ben még ez nem tud lefutni hisz ott még külön kellett dekralálni. A feladatunk az most hogy mutassunk kettő olyan kód csipetet amivel betudjuk bizonyítani hogy visszafele nem fordulnak.

```
int main(){
//int a;
return
```

Ilyen például a fenti kódcsipetünk, ez C99-be lefüt,C89-ben nem,mégpedig azért mert megengedett a //-el való kommentelés viszont a C89-ben még nem.

```
        #include <stdio.h>

int main()
{
for(int a=5;a>10;a++);
return 0;
}
```

Itt pedig a leírásban említett dekralálás látható.

## 3.4. Saját lexikális elemző

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

Megoldás forrása: https://github.com/glszKK/Chomsky/tree/master

A programunk működése úgy néz ki hogy include-oljuk az stdio.h header filet és jelzem hogy en ezzel szeretnék dolgozni. Majd ezután definiáljuk és írjuk le a lexer számlálót ami ugye számolni fogja a bemenetet betűként. A digit függvénnyel adjuk meg a számokat majd növeljük azokat egyesével. A következő sorba láthatjuk hogy milyen karaktereket ismer fel [A-Za-z][A-Za-z0-9]. Aztán jön a feladatunk main része ahol meghívjuk az elemzőt ahogy a commentben is olvashatjuk, azaz a lexert (yylex). Majd kiírjuk az eredményt amit kaptunk. A returtn 0.-val pedig véget vetünk a programunknak.

```
/*** definíciós rész ***/

%{
/* Ez a kód bemásolódik a generált C forrásba*/
#include <stdio.h>
int valos=0;
%}

/* Ez az opció azt mondja meg, hogy csak egy input file kerüljön ↩
   beolvasásra. */
%option noyywrap

%%
/*** Szabályok ***/


[:digit:]+  { valos++; }
[A-Za-z][A-Za-z0-9]*       {   /* Minden más karaktert ignorálunk. */  ↩
     }

%%
/*** C kód. Ez is bemásolódik a generált C forrásba. ***/

int main()
{
   /* Meghívjuk az elemzőt, majd kilépünk.*/
      yylex();
printf("%d valós számot talált a lexer: \n",valos);
   return 0;
}
```

## 3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása: https://github.com/glszKK/Chomsky/tree/master

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
int x=0;
typedef struct{
char c;
char *d[7];
} cipher;
```

```
    cipher L337[] = {
    {'a',{"4","/-\\","a","/\\","a","a","a"}},
    {'b',{"!3","|3","8","ß","b","b","b"}},
    {'c',{"[","<","{","c","c","c","c"}},
    {'d',{"|}","d","|>","T)","d","d","d"}},
    {'e',{"3","&","e","€","e","e","e"}},
    {'f',{"|=","|#","/=","f","f","f","f"}},
    {'g',{"&","6","g","(_+","g","g","g"}},
    {'h',{"|-|",")-(","[-]","h","h","h","h"}},
    {'i',{"1","[]","!","|","i","i","i"}},
    {'j',{"_|",";","1","j","j","j","j"}},
    {'k',{">|","1<","|c","k","k","k","k"}},
    {'l',{"1","|_","l","|","l","l","l"}},
    {'m',{"/\\/\\","/V\\","[V]","m","m","m","m"}},
    {'n',{"</>","n","|\\|","^/","n","n","n"}},
    {'o',{"0","Q","o","<>","o","o","o"}},
    {'p',{"|*","|>","p","|7","p","p","p"}},
    {'q',{"(_,)","q","9","&","q","q","q"}},
    {'r',{"I2","|?","Iz","r","r","r","r"}},
    {'s',{"s","5","z","§","s","s","s"}},
    {'t',{"4","-|-","7","t","t","t","t"}},
    {'u',{"(_)","u","v","L|","u","u","u"}},
    {'v',{"v","\\/","|/","\\|","v","v","v"}},
    {'w',{"\\/\\/","w","\\x/","\\\\//\\\\//","w","w","w"}},
    {'x',{"4","}{",">","x","x","x","x"}},
    {'y',{"y","j","`/","\\|/","y","y","y"}},
    {'z',{"2","-/_","z",">_","z","z","z"}},
    {'1',{"I","1","L","I"}},
    {'2',{"R","2","2","Z"}},
    {'3',{"E","3","E","3"}},
    {'4',{"4","A","A","4"}},
    {'5',{"S","5","S","5"}},
    {'6',{"b","6","G","6"}},
    {'7',{"7","7","L","T"}},
    {'8',{"8","B","8","B"}},
    {'9',{"g","q","9","9"}},
    {'0',{"0","()","[]","0"}},
    };
    %}
    %option noyywrap
    %%
    \n    {
    printf("\n");
    }
.     {
    srand(time(0)+x++);
      char c = tolower(*yytext);
      int i=0;
      while(i<36 && L337[i++].c!=c);
      if(i<36)
```

```
      {
        char *s=L337[i-1].d[rand()%7];
        printf("%s",s);
      }
    else
    {
      printf("%c",c);
    }
    }
    %%
    int main()
    {
    yylex();
    return 0;
    }
```

Itt megint egy lexer-re lesz szükségünk, a fordításhoz szükségünk lesz a lex leet.c parancsra amellyel gcc által olvasható forráskódot fog létrehozni. De ez a program nem számolgatni fog, itt a dolga az lesz hogy szöveg fájlból vegye be / olvassa be a szabályokat majd ezekalapján készítsen egy C forráskódot. Egy ilyen lex forráskód általában 3 részből áll. Az első iylen rész a definiciós a második az a rész ahol a szabályokat tároljuk a harmadik rész pedig maga a C kód amit a szabályok által generált. A programunk alapja ezenkívül az hogy létrehoz egy cipher típusú tömböt ami a leet kódokat tárolja. Ha kap a programunk egy bemenetet akkor megvizsgálja hogy a létrehozott tömbünkben benne van-e? Ha igen kiválasztja a megfelelő kódolási formát és kiírja kimenetként, ha pedig nem találja meg akkor a bemenetet írja ki kimenetként. A mainbe megint találkozhatunk a yylex() parancsal, amit már az előző feladatból tudunk hogy meghívja magát a lexet, majd megint egy return 0-val zárjuk a programunkat.

## 3.6. A források olvasása

A feladatunk az lesz hogy különféle kódcsipeteket kaptunk és elkell magyaráznunk hogy mit jelentenek.

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?

Megoldás forrása: https://github.com/glszKK/Chomsky/tree/master

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

Ezt már ugye a fenti feladat leírásban láthattuk példaként.

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

Ez egy for ciklus, ami 0-tól 4-ig tart hisz nincs megengedve az egyenlőség,i-t megnövelve. Majd i ezt a megnövelt értéket veszi fel

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

Ez ugyan az mint az előző, csak itt az i nem a növelt értéket hanem a növelés előtti értéket veszi fel.

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

Az eleje megint ugyan az, de a végén i-vel egyenlővé teszi a tömb-i-ik elemét

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

Ez megint egy forciklus, szintén 0 a kezdőérték, és addig megyünk amíg az i értékünk kisebb az n-nél. És az S mutató hozzárendeli ahhoz az értékhez amire a d mutat és minden kör végén az i-t egyel növelnünk kell.

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

A printf a kiíratás, szóval tudjuk hogy két eégszet fogunk kiíratni, amikhez tartozik 1-1 művelet.

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

Megint két egészet fogunk kiírni, az egyik az f függvény értékét, a másik pedig az a függvény értékét.

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

Itt ugyan az a helyzet mint az előzőnél, csak most az f függvényünk tudja változtatni az a függvényünket.

## 3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

Az első formula értelmezése az hogy végtelen sok prímünk van.

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

A második formula jelentése az hogy végtelen sok IKERPRÍM van.

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

Ez azt jelenti hogy Véges sok prím van.

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

Ez pedig ugyan azt jelenti mint az előző, csak kicsit más környezetbe van leírva és máshogy fogalmazva. Tehát ugyan úgy Véges sok prímünk van.

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

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

## 3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

Megoldás forrása: https://github.com/glszKK/Chomsky/tree/master

- 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

A feladatunk az hogy fentiekben felsorolt elemekből létrehozzunk pár forráskódot ami lefut. Az első ilyen kódom egy faktoriális számolás lesz.

```
int* fakt(int szam) {
  static int a = 1;
  if (szam < 2)
    return &a;
  while (szam>1) {
    a = a*szam;
    --szam;
  }
  return &a;
```

Mit vezetnek be a programba a következő nevek?

- ```
  int a;
  ```

  Egész típusú változó
- ```
  int *b = &a;
  ```

  Egy egész típusú mutatót ami a-ra mutat.
- ```
  int &r = a;
  ```

  a változónak a referenciája.
- ```
  int c[5];
  ```

  Egy 5 elemű egész típusú tömböt.
- ```
  int (&tr)[5] = c;
  ```

  Egészek tömbjének referenciáját.
- ```
  int *d[5];
  ```

  5 elemű int-re mutató mutatók tömbjét.
- ```
  int *h ();
  ```

  Egy függvényt ami int-re mutató mutatót ad vissza.
- ```
  int *(*l) ();
  ```

  Egy int-re mutató mutatót visszaadó függvényre mutató mutatót.(pl. az előző függvényre)
- ```
  int (*v (int c)) (int a, int b)
  ```

  int-et visszaadó, két intet kapó függvényre mutató mutatót visszaadó egészet kapó függvényt.
- ```
  int (*(*z) (int)) (int, int);
  ```

  int-et visszaadó, két intet kapó függvényre mutató mutatót visszaadó egészet kapó függvényre mutató mutatót.

# 4. fejezet

# Helló, Caesar!

## 4.1. int *** háromszögmátrix

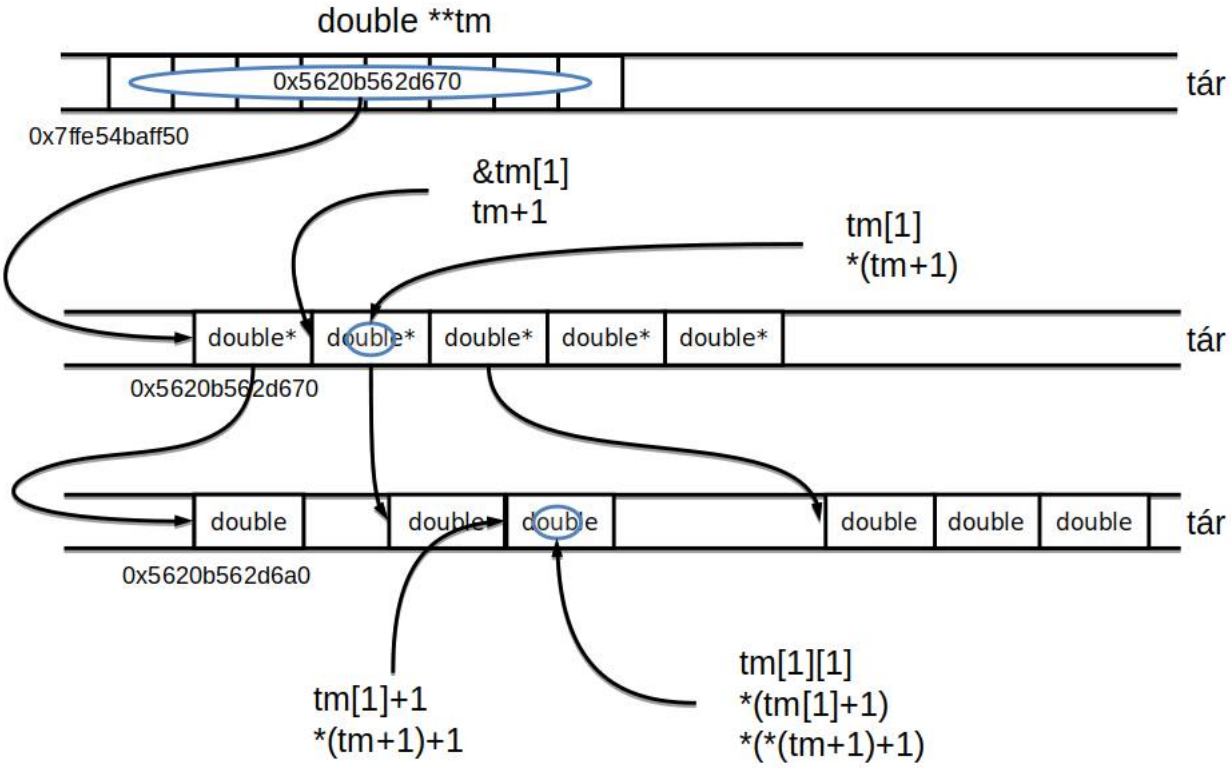
Írj egy olyan malloc és free párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

Megoldás videó:https://www.youtube.com/watch?v=1MRTuKwRsB0

Megoldás forrása:https://github.com/glszKK/Caesar

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

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

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

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

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

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

    }
```

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

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

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

    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
    *(tm[3] + 2) = 44.0;
    *(*(tm + 3) + 3) = 45.0;

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

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

    free (tm);

    return 0;
}
```




Elérkeztünk a harmadik fejezethez, ahol nagyrészt az EXOR-okról lesz szó. Első feladatunkban a memóriakezelésről és a pointerekről fogunk tanulni. A háromszögmátrix a nevétől eltérően négyzetes, ami azt jelenti hogy a sorai és az oszlopainak a száma megegyeznek mert ugye így jön létre egy szabályos négyzet ha minden oldala egyenlő. Emellett a főátlója alatt nullák helyezkednek el. És pontosan ezaz ami kell nekünk, ez a mátrix, hisz a programunk ezt fogja elkészíteni. Szokásosan a header file-ok include-olásával kezdem a feladatot majd létrehozzuk az int nr parancsal a soraink számát. Majd a programba bevezetjük/deklaráljuk a tm változót. Majd memóriát foglalunk le a doublekból álló háromszögmátrixunknak, mégpedig 8 byte-ot. Aztán jön a malloc parancs ami visszaad egy pointert ami nullára mutat ha a programunk hibába ütközik. A ciklusunk 0-tól 4-ig megy még el nem éri az nr-t. És a lényeg az hogy az első sorunkban egyetlen egy double-t akarunk lefoglalni, a második sorunk mutasson olyan helyre ahol két doublenak van lefoglalva hely,és így szabályszerűen novekedik. Ezután standart outputra megkapjuk az eredményt, majd egy return 0 parancsal zárjuk a programunkat.

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása:https://github.com/glszKK/Caesar

Ebben a feladatban egy titkosító programot fogunk írni C nyelven melynek a feladata az lesz hogy egy bármekkora méretű kulcs megadásával titkosít szövegeket. A kódunk elején meghatároztuk hogy a max kulcsunk 100 lehet. És ezt nem fogjuk tudni sehogysem megváltoztatni később sem. És még a buffer méretet is beállítjuk, ennek a maximuma 256 lesz. Ez azt adja meg hogy a programunk hány karaktert tud feldolgozni és beolvasni. A következő parancsunk a strlen és a strncpy ahol a karakterek információit találjuk, hogy milyen hosszú a karakterunk. A kulcsot parancssori argumentumként kéri be. Aztán jön a while ciklus ami a read parancs segítségével elkezdi ugye olvasni a programunkat még a fálj végén lévő

"vége" jellel nem fut össze. És ugye a programunk elején megadtuk a max buffert, tehát 256-nyi karaktert olvasott. És most jön a titkosítás a write-al. A programunk a bitenként megvizsgál 2 értéket, és mindig 1-et kapunk vissza, kivéve akkor ha a 2 érték megeggyezik. Ekkor nullát kapunk vissza eredményül.

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

int
main (int argc, char **argv)
{
  char kulcs [MAX_KULCS];
  char buffer[BUFFER_MERET];

  int kulcs_index = 0;
  int olvasott_bajtok = 0;

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

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

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:https://github.com/glszKK/Caesar

```
        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();

        }

    }

}
```

A feladatunk ugyan az,titkosítás, csak most Java-ban. Aminek az előnye az hogy ugye Java-ban létre tudunk hozni bizonyos osztályokat amikkel különféle utasításokat tudunk végrehajtani. Ezért létre is hozunk egy ilyen osztályt amit ExorTitkosítonak nevezünk el. És itt kell ugye elvégezni az utasításokat ami azt jelenti hogy itt fogjuk a következő lépéseket definiálni. Láthatunk a main résznél egy public -ot, ami azt mutatja meg hogy ez a rész a class(osztály)-on kívül is elérhető. Visszatérési értéke nincs a void miatt. Két újabb dologgal is találkozhatunk, a Try-al és a Catch-el, amik feladata hogy ha hibát talál a program akkor a catch elkapja és kíírja a hibaüzenet.

## 4.4. C EXOR törő

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

Megoldás forrása:https://github.com/glszKK/Caesar

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

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

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

                      && strcasestr (titkos, "hogy") && strcasestr ( ↩
                         titkos, "nem")
                      && strcasestr (titkos, "az") && strcasestr (titkos, ↩
                          "ha");

              }

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

              int kulcs_index = 0;

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

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

              }

          }

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

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

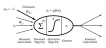
```
                return tiszta_lehet (titkos, titkos_meret);

              }


              char kulcs[KULCS_MERET];
              char titkos[MAX_TITKOS];
              char *p = titkos;
              int olvasott_bajtok;

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


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


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

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

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

Egy picit összetetteb és bonyolultabb függvényről lesz szó most, hisz több függvényből fog állni. A fejezet második feladatához térünk vissza, ahol ugye az előállított titkos szövegünket akarjuk feltörni és visszafejteni. Ebbe lesz segítségünk az atlagos_szohossz függvényünk amely pont ezt csinálja, a programunk által generált kulcsal visszafejti a titkosított szöveget és vissza adja az átlagos szóhosszát. Ezt is úgy kezdjük mint a titkosítást, hogy meghatározzuk az értékeket, a maximumokat, a kulcs méretet és a buffer számot. Aztán jön a tiszta_lehet függvény ami az előzőleg említett atlagos_szohossz függvény által generált és kiszámolt értékkel dolgozik és megnézi hogy tiszta e a kód(amit úgy dönt el hogy a vizsgált szövegben van-e hogy,nem,az vagy ha szavak egyike).És ha a vizsgált szavak kozül egyiket sem találja meg akkor egy null értéket kapunk eredményül, ami sajnos a programunk negatívuma is. Ebből következik az exor_tores függvényünk ami szintén az előzőleg használt függvénykkel és eredményekkel dolgozozik, ő is megnézi hogy tiszta e már a fejtés alatt lévő szövegünk, és vagy 0-val vagy 1-el tér vissza. A main-en belül adjuk meg a kulcsunk hosszát amit a-z-ig az összes betűvel vagy 0-9-ig az összes számmal megadhatunk. Itt találhatjuk a for ciklusokat amik a kulcsot készítik el, és ezek próbálják meg a törést. És ha valamelyik ciklus sikerrel jár akkor a tiszta_lehet függvényünknek megfelel. A program forráskódja commentelve van így sokkal könnyebben tudjuk értelmezni a feladatot és hogy mi mit is csinál.

## 4.5. Neurális OR, AND és EXOR kapu

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

Ebben a feladatban a neurális hálókról lesz szó, ami egy neuronokból összeálló háló. Maga a neuron egy idegsejt ami az idegrendszerünk legkisebb egysége. Ezeknek a feladata a jelek összegyüjtése és továbbítása. Többekközött a mesetséges intelligánciával is fogunk itt foglalkozni, mert ez a programunk képes a tanulásra. A feltételezés az hogy az emberi agy információfeldolgozását ezeknek a neuronoknak, és az általuk alkotott neurális hálók alkotják. Ezért próbálnak a mesterséges intelligencia továbbfejlesztésénél arra törekedni hogy egy ilyen fajta hálót tudjanak reprodukálni mint az agyunkban fellelhető. Mi is ezért fogjuk ezeket segítségül hívni hisz egy olyan programot szeretnénk létrehozni ami alapvető logikai műveleteket képes elvégezni. A neuronokról még azt kell tudni hogy akkor fognak "tüzelni" hogyha a bemenet összege meghaladnak egy limitet. Es ezt szeretnénk reprezentálni. A kódunk első részében az OR-t azaz a logikai vagy-ot tanítjuk meg a hálózatunknak, majd utána az AND azaz a logika és-t. Majd legvégül az EXORT ami a logika kizáró vagy. Az OR-t és az AND-et hibátlanul megtanulja, ám az EXOR nem működik rendesen. Sokáig nem tudták a probléma okát, ám később pár matematikus rájött hogy miért is működik hibásan az EXOR kapu. A megoldás az hogy rejtett neutronokat kell még plusszba létrehoznunk a többi mellé, amik szintén segítik a tanulást. Amit a neuralnet hidden parancsal tudunk beállítani.



```
      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])
]]<>/programlisting>
```

```
        <para>
            Az AND (és) csak abban az esetben tér vissza 1-es értékkel, ha A AND B esetében A és B is 1-es értékkel rendelkezik. Minden egyéb esetben
            pl: A = 0 B = 0, A = 1 B = 0, A = 0 B = 1 0 értékkel tér vissza .

            Az OR (megengedő vagy) minden esetben 1-es értékkel tér vissza, kivéve ha A OR B esetében mind a 2 0 értéket tartalmaz.
            pl: A = 0 B = 0 esetben A OR B = 0, minden más esetben A OR B = 1
            pl: A = 1 B = 1, A = 0 B = 1, A = 1 B = 0.

            Az XOR (kizáró vagy) akkor tér vissza 1-es értékkel, ha A XOR B esetében mindkét elem értéke eltérő.
            Ha a két elem értéke megegyezik A XOR B = 0 lesz.
            pl: A = 1 B = 0 és A = 0 B = 1 esetében lesz A XOR B értéke 1.
            A = 1 B = 1, A = 0 B = 0 esetében lesz A XOR B értéke 0.
        </para>
    </section>
    <section>
        <title>Hiba-visszaterjesztéses perceptron</title>
        <para>
            C++
        </para>
        <para>
            Megoldás forrása:<link xlink:href="https://github.com/glszKK/Caesar">https://github.com/glszKK/Caesar</link>
        </para>
      <programlisting language="C++"><![CDATA[
      include <iostream>
#include "ml.hpp"
#include <png++/png.hpp>

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

    int size = png_image.get_width() * png_image.get_height();

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

    double* image = new double[size];

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

    double value = (*p) (image);
```

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

    delete p;
    delete [] image;

}
```

A következő feladatunk a neuronokhoz és a mesterséges intelligenciához kapcsolható, mivel ez a program a neuron hálózatok egyik legelterjedtebb változata, ami alakfelismerő, tehát megadunk neki egy alakzatot egy formát és ő meg mondja hogy mit láthatunk a bemeneten. A programunknak 3 fő része van, az egyik ilyen rész a retinacella ami a bemenetet fogadja, ami igen vagy nem választ adja. A második részünk az asszociatív cella aminek néhány cellája az előzőleg említett retinacellához kapcsolódik, a maradék pedig a harmadik fő részhez csatlakozik. Ezek vizsgálják a beérkező adatokat, jeleket. A harmadik nagy rész pedig a döntés cella ami a vége a programunknak, itt kapjuk meg a kimenetunket. A programunk úgymond megtanítja a számítógépünknek a bináris osztályozást. Majd csak a következő fejezetben fogunk beszélni a Mandelbrot halmazról ám most ezt hívjuk segítségül, magát a halmazt ábrázoló kép RGB kódját adjuk meg a neurális hálónak, majd a programunk megnézi az összes pixelt és számolja a piros pixeleket. És ezáltal határozza meg hogy a pixelek hány %-a volt piros így megtudja határozni az alakját a bemenetnek.

# 5. fejezet

# Helló, Mandelbrot!

## 5.1. A Mandelbrot halmaz

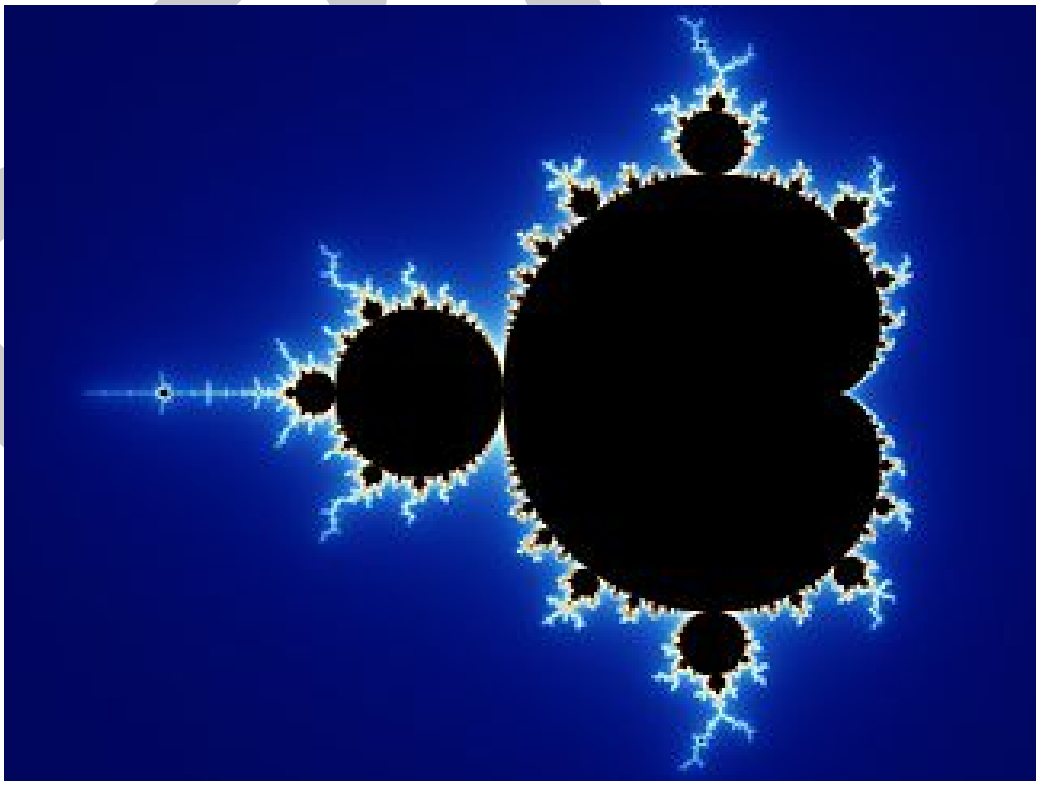
Írjunk olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás forrása:https://github.com/glszKK/Mandelbrot

Benoít Mandelbrot, így hívták azt a lengyel matematikust akiről a fejezetünk és a fejezetben lévő programok a nevüket kapták. Az első feladatban a Mandelbrot halmazt kell létrehoznunk, ami egy komplex számsíkon van ábrázolva. Es egy fura alakzatot vesz fel, amit a matematikában fraktálnak nevezünk. A fraktálok olyan matematikai alakzatok melyek végtelenül komplexek. Ezeknek a fraktáloknak két ismertető jele van, az egyik az hogy a szélei gyűrüttek, szakadásosak, töredezettek. A másik ismertető jel pedig az önhasonlóság. Ahhoz hogy a programunk működjön szükségünk lesz a png++ header file-ra is. Először megadjuk a szükséges függvényünk Rf és Df-jét. Majd létrehozunk egy üres png-t, amibe az egész halmazunk kerülni fog. Majd beállítjuk hogy a komplex számsíkon hogy szeretnénk lépkedni. Majd ezeknek az értékeknek létrehozunk 3 változót. A megadott paraméterek és értékek által a for ciklusunk lépked. Majd miután befejezte a lépkedést a GeneratePNG-vel meghívjuk a függvényt.

```
#include <png++/png.hpp>
#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35
void GeneratePNG(int tomb[N][M]) {
  png::image<png::rgb_pixel> image(N, M);
  for (int x = 0; x < N; ++x) {
    for (int y = 0; y < M; ++y) {
      image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y]);
    }
  }
  image.write("kimenet.png");
}
struct Komplex {
  double re, im;
```

```
};
int main() {
  int tomb[N][M];
  int i, j, k;
  double dx = (MAXX - MINX) / N;
  double dy = (MAXY - MINY) / M;
  struct Komplex C, Z, Zuj;
  int iteracio;
  for (i = 0; i < M; ++i) {
    for (j = 0; j < N; ++j) {
      C.re = MINX + j * dx;
      C.im = MAXY - i * dy;
      Z.re = 0;
      Z.im = 0;
      iteracio = 0;
      while (Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < 255) {
        Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
        Zuj.im = 2 * Z.re * Z.im + C.im;
        Z.re = Zuj.re;
        Z.im = Zuj.im;
      }
      tomb[i][j] = 256 - iteracio;
    }
  }
  GeneratePNG(tomb);
  return 0;
}
```



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

Megoldás forrása:https://github.com/glszKK/Mandelbrot

Ez a feladat szinte ugyan az mint az előző, néhány apróságban térnek el egymástól. Mégpedig abban hogy itt a C++-ban találhatunk egy beépített komplex osztályt aminek a segítségével lépkedünk. Ezenkívül jobban elengedi a kezünket a program, hisz itt már betudjuk állítani saját magunknak a méreteket ha pedig lusták vagyunk akkor van egy basic alapértelmezett képméret és azt fogja használni a programunk. A dolgunk annyi még hogy a változókat int és double alakra át kell írnunk. Majd megint létrehozunk egy üres png-t amibe készül az egész halmazunk. Különbség még az hogy itt két forciklust ágyazunk egymásba amik segítségével járjuk be az egészet és legvégezetül egy színesebb alakzatot fogunk kapni.

```
#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 )
    {
```

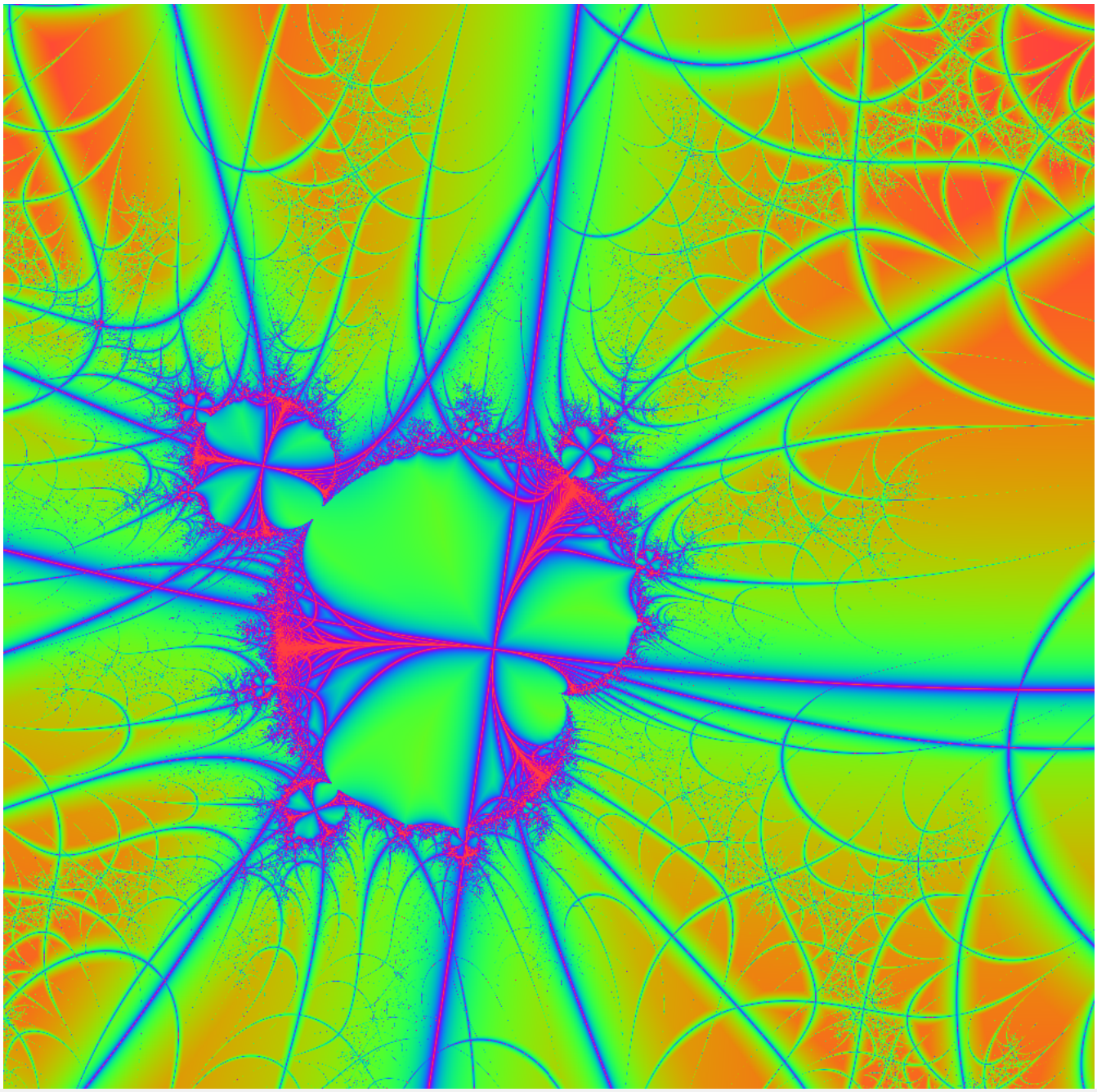
```
      // k megy az oszlopokon
      for ( int k = 0; k < szelesseg; ++k )
        {
          // c = (reC, imC) a halo racspontjainak
          // megfelelo komplex szam
          reC = a + k * dx;
          imC = d - j * dy;
          std::complex<double> c ( reC, imC );
          std::complex<double> 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;
}
```

## 5.3. Biomorfok

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

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

Ez a program azért került ebbe a csokorba mert az előzőleg megírt Mandelbrot halmazok segítségével fogjuk ezeket a biomorfokat is létrehozni. Ezek a biomorfok élő organizmusokhoz hasonlítanak, amiket Pickover talált fel, és ő meg volt győződve arról hogy valami újat fedezett fel a természetben. De mint később kiderült mégsem, hisz ilyen alakzatokat és biomorfokat akár számítógép és matematikai egyenletek segítségével is létrehozhatunk. A program eleje tisztára olyan mint a Mandelbrot halmazé annyi különbséggel hogy itt a felhasználó saját magának adhatja meg a cc konstans értékét és a küszöbszámokat. Itt is két egymásba ágyazott forciklust használunk, és ezek segítségével ugyanúgy végig lépkedünk a pontokon. De van egy harmadik for ciklus is ami a számításokat végzi, egészen addig míg a programunkban leírt határt el nem éri.



```
#include <iostream>
#include "png++/png.hpp"
#include <complex>
int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
```

```
    double reC = .285, imC = 0;
    double R = 10.0;
    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag =  atoi ( argv[3] );
        iteraciosHatar =  atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );
    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↩
            d reC imC R" << std::endl;
        return -1;
    }
    png::image < png::rgb_pixel > kep ( szelesseg, magassag );
    double dx = ( xmax - xmin ) / szelesseg;
    double dy = ( ymax - ymin ) / magassag;
    std::complex<double> cc ( reC, imC );
    std::cout << "Szamitas\n";
    // j megy a sorokon
    for ( int y = 0; y < magassag; ++y )
    {
        // k megy az oszlopokon
        for ( int x = 0; x < szelesseg; ++x )
        {
            double reZ = xmin + x * dx;
            double imZ = ymax - y * dy;
            std::complex<double> z_n ( reZ, imZ );
            int iteracio = 0;
            for (int i=0; i < iteraciosHatar; ++i)
            {
                z_n = std::pow(z_n, 3) + cc;
                //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
                if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                {
                    iteracio = i;
                    break;
                }
            }
            kep.set_pixel ( x, y,
                            png::rgb_pixel ( (iteracio*20)%255, (iteracio ↩
                               *40)%255, (iteracio*60)%255 ));
        }
```

```
        int szazalek = ( double ) y / ( double ) magassag * 100.0;
        std::cout << "\r" << szazalek << "%" << std::flush;
    }
    kep.write ( argv[1] );
    std::cout << "\r" << argv[1] << " mentve." << std::endl;
```

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása:https://github.com/glszKK/Mandelbrot

```
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>
#include <sys/times.h>
#include <iostream>
#define MERET 600
#define ITER_HAT 32000
__device__ int
mandel (int k, int j)
{
  // Végigzongorázza a CUDA a szélesség x magasság rácsot:
  // most eppen a j. sor k. oszlopaban vagyunk
  // számítás adatai
  float a = -2.0, b = .7, c = -1.35, d = 1.35;
  int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;
  // a számítás
  float dx = (b - a) / szelesseg;
  float dy = (d - c) / magassag;
  float reC, imC, reZ, imZ, ujreZ, ujimZ;
  // Hány iterációt csináltunk?
  int iteracio = 0;
  // c = (reC, imC) a rács csomópontjainak
  // megfelelő komplex szám
  reC = a + k * dx;
  imC = d - j * dy;
  // z_0 = 0 = (reZ, imZ)
  reZ = 0.0;
  imZ = 0.0;
  iteracio = 0;
  // z_{n+1} = z_n * z_n + c iterációk
  // számítása, amíg |z_n| < 2 vagy még
  // nem értük el a 255 iterációt, ha
  // viszont elértük, akkor úgy vesszük,
  // hogy a kiinduláci c komplex számra
  // az iteráció konvergens, azaz a c a
  // Mandelbrot halmaz eleme
  while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
```

```
      // z_{n+1} = z_n * z_n + c
      ujreZ = reZ * reZ - imZ * imZ + reC;
      ujimZ = 2 * reZ * imZ + imC;
      reZ = ujreZ;
      imZ = ujimZ;
      ++iteracio;
    }
  return iteracio;
}
/*
__global__ void
mandelkernel (int *kepadat)
{
  int j = blockIdx.x;
  int k = blockIdx.y;
  kepadat[j + k * MERET] = mandel (j, k);
}
*/
__global__ void
mandelkernel (int *kepadat)
{
  int tj = threadIdx.x;
  int tk = threadIdx.y;
  int j = blockIdx.x * 10 + tj;
  int k = blockIdx.y * 10 + tk;
  kepadat[j + k * MERET] = mandel (j, k);
}
void
cudamandel (int kepadat[MERET][MERET])
{
  int *device_kepadat;
  cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));
  // dim3 grid (MERET, MERET);
  // mandelkernel <<< grid, 1 >>> (device_kepadat);

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

  cudaMemcpy (kepadat, device_kepadat,
        MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
  cudaFree (device_kepadat);
}
int
main (int argc, char *argv[])
{
  // Mérünk időt (PP 64)
  clock_t delta = clock ();
  // Mérünk időt (PP 66)
  struct tms tmsbuf1, tmsbuf2;
```

```
  times (&tmsbuf1);
  if (argc != 2)
    {
      std::cout << "Hasznalat: ./mandelpngc fajlnev";
      return -1;
    }
  int kepadat[MERET][MERET];
  cudamandel (kepadat);
  png::image < png::rgb_pixel > kep (MERET, MERET);
  for (int j = 0; j < MERET; ++j)
    {
      //sor = j;
      for (int k = 0; k < MERET; ++k)
  {
    kep.set_pixel (k, j,
       png::rgb_pixel (255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT));
  }
    }
  kep.write (argv[1]);
  std::cout << argv[1] << " mentve" << std::endl;
  times (&tmsbuf2);
  std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
    + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;
  delta = clock () - delta;
  std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}
```

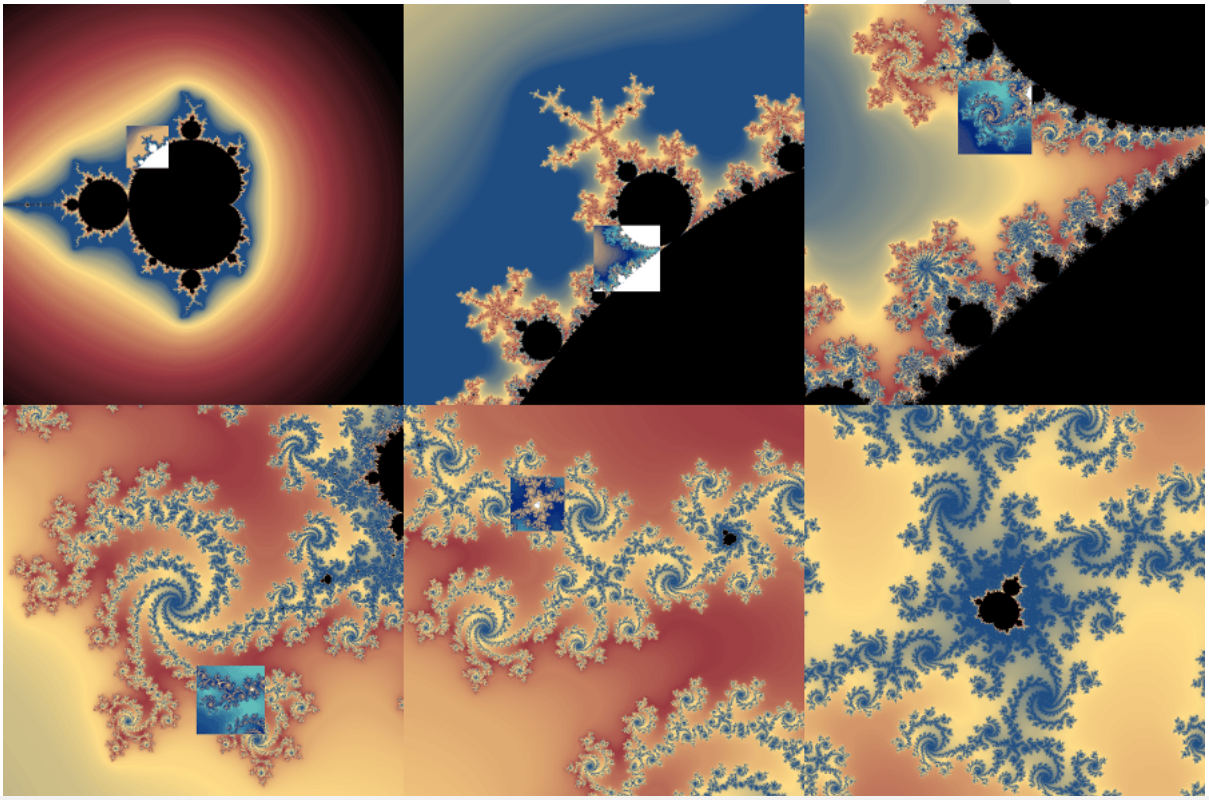
Ezt a programot az NVIDIA fejlesztette ki, melynek segítségével a korábban megírt programokat és Mandelbrot halmazok generálását és elkészítését jelentősen meg tudjuk növelni. A program lényege létrehozunk egy új gride-t ami 600x600 blokkból tevődik össze, és minden egyes blokkhoz tartozik egy szál. Az egész programhoz egy NVIDIA GPU szükséges. A define-al meghatározzuk a méreteket és az értékeket. majd a Mandelbrot halmazunkat egy mandel parancsal hozzuk létre, ám a kódunkba a mandel előtt szerepel egy device szó, ami annyit takar hogy jelzi hogy maga a program/függvény csakis a videókártyáról lesz elérhető. Aztán ott a mandelkernel, ezelőtt is áll egy szó, mégpedig a global szó. Amely azt jelzi hogy ez egy kernel függvény. Ezek voltak a program érdekes részei, mivel a forrást úgy kaptuk meg hogy commentelve van és minden fontos információ a parancsokról és a függvényekről megtalálható ott, így nem fogom leírni külön ide őket.

## 5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:https://github.com/glszKK/Mandelbrot

A következő programunk alapja megint a Mandelbrot halmaz, és mint a címben is láthatjátok, nagyításról lesz szó. A programunk rá fog tudni zoomolni a halmaz szélére ahol újabb halmazt/halmazokat találhatunk. De először is telepítenünk kell két alapvető csomagot.(sudo apt-get install libsfml-dev). Mint a fejezetben mindenhol, itt is a méretek beállításával kezdünk. Azután mindent úgy csinálunk mint a fejezet második feladatában a komplex halmaznál. A színeket itt tudjuk állítani, mégpedig az iterációk számával. Beállítottuk hogy az X-re kattintás esetén a programunk bezáródjón, és még azt is beállítottuk hogy kattintáskor a kurzor helyére zoomoljon a programunk, ezért beszélünk nagyításról.



```
#include "SFML/Graphics.hpp"

//resolution of the window
const int width = 1280;
const int height = 720;

//used for complex numbers
struct complex_number
{
    long double real;
    long double imaginary;
};

void generate_mandelbrot_set(sf::VertexArray& vertexarray, int ↩
   pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
```

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

            long double x = ((long double)j - pixel_shift_x) / zoom;
            long double y = ((long double)i - pixel_shift_y) / zoom;
            complex_number c;
            c.real = x;
            c.imaginary = y;
            complex_number z = c;
            int iterations = 0;
            for (int k = 0; k < precision; k++)
            {
                complex_number z2;
                z2.real = z.real * z.real - z.imaginary * z.imaginary;
                z2.imaginary = 2 * z.real * z.imaginary;
                z2.real += c.real;
                z2.imaginary += c.imaginary;
                z = z2;
                iterations++;
                if (z.real * z.real + z.imaginary * z.imaginary > 4)
                    break;
            }

            if (iterations < precision / 4.0f)
            {
                vertexarray[i*width + j].position = sf::Vector2f(j, i);
                sf::Color color(iterations * 255.0f / (precision / 4.0f),  ↩
                   0, 0);
                vertexarray[i*width + j].color = color;
            }
            else if (iterations < precision / 2.0f)
            {
                vertexarray[i*width + j].position = sf::Vector2f(j, i);
                sf::Color color(0, iterations * 255.0f / (precision / 2.0f) ↩
                   , 0);
                vertexarray[i*width + j].color = color;
            }
            else if (iterations < precision)
            {
                vertexarray[i*width + j].position = sf::Vector2f(j, i);
                sf::Color color(0, 0, iterations * 255.0f / precision);
                vertexarray[i*width + j].color = color;
            }
        }
    }
}

int main()
{
    sf::String title_string = "Mandelbrot Set Plotter"; /
```

```
    sf::RenderWindow window(sf::VideoMode(width, height), title_string);
    window.setFramerateLimit(30);
    sf::VertexArray pointmap(sf::Points, width * height);



    float zoom = 300.0f;
    int precision = 100;
    int x_shift = width / 2;
    int y_shift = height / 2;


    generate_mandelbrot_set(pointmap, x_shift, y_shift, precision, zoom);


    /**
     *
     *
     *
     *
     * */
    while (window.isOpen())
    {


        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
        {
            sf::Vector2i position = sf::Mouse::getPosition(window);
            x_shift -= position.x - x_shift;
            y_shift -= position.y - y_shift;
            zoom *= 2;
            precision += 200;

            #pragma omp parallel for
            for (int i = 0; i < width*height; i++)
            {
                pointmap[i].color = sf::Color::Black;
            }
            generate_mandelbrot_set(pointmap, x_shift, y_shift, precision,  ↩
               zoom);
        }
        window.clear();
        window.draw(pointmap);
```

```
        window.display();
    }

    return 0;
}
```

## 5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása:https://github.com/glszKK/Mandelbrot

A feladatunk ugyan az lesz, egy olyan programot kell létrehoznunk ami ugyanúgy zoomolni tud, csak most Java-ba. Először szokás szerint importálunk, majd létrehozunk egy osztályt. A setbounds résznél állítjuk be az ablakunk méretét. Az alatta lévő setResizable pedig arra szolgál hogy ha false értéket adunk meg neki akkor nem tudjuk változtatni az ablak méretét. C++-ban ugye new-al foglaltunk le memóriát, itt java-ban szintén a new parancsal fogunk, mégpedig a new JPanel parancsal. Majd létrehozzuk a gombokat is, amelyekkel a számokat és a tartományt tudjuk varriálni. És legvégül találkozhatunk 2 résszel, az egyik a plotPoints ami arra szolgál hogy bejárja a megadott tartományunkat és elkészítse a halmazunkat és kiszámolja azt. A másik rész pedig az actionPerformed aminek a segítségével tudjuk megmondani a programunknak hogy hova is szeretnénk zoomolni.

```
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;
import java.awt.event.*;

public class Mandelbrot extends JFrame implements ActionListener {

  private JPanel ctrlPanel;
  private JPanel btnPanel;
    private int numIter = 50;
    private double zoom = 130;
    private double zoomIncrease = 100;
    private int colorIter = 20;
    private BufferedImage I;
    private double zx, zy, cx, cy, temp;
    private int xMove, yMove = 0;
    private JButton[] ctrlBtns = new JButton[9];
    private Color themeColor = new Color(150,180,200);

    public Mandelbrot() {
        super("Mandelbrot Set");
        setBounds(100, 100, 800, 600);
        setResizable(false);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        plotPoints();

        Container contentPane = getContentPane();
```

```
      contentPane.setLayout(null);




      ctrlPanel = new JPanel();
      ctrlPanel.setBounds(600,0,200,600);
      ctrlPanel.setBackground(themeColor);
      ctrlPanel.setLayout(null);

      btnPanel = new JPanel();
      btnPanel.setBounds(0,200,200,200);
      btnPanel.setLayout(new GridLayout(3,3));
      btnPanel.setBackground(themeColor);

      ctrlBtns[1] = new JButton("up");
      ctrlBtns[7] = new JButton("down");
      ctrlBtns[3] = new JButton ("left");
      ctrlBtns[5] = new JButton("right");
      ctrlBtns[2] = new JButton("+");
      ctrlBtns[0] = new JButton("-");
      ctrlBtns[8] = new JButton(">");
      ctrlBtns[6] = new JButton("<");
      ctrlBtns[4] = new JButton();

      contentPane.add(ctrlPanel);
      contentPane.add(new imgPanel());
      ctrlPanel.add(btnPanel);

      for (int x = 0; x<ctrlBtns.length;x++){
        btnPanel.add(ctrlBtns[x]);
        ctrlBtns[x].addActionListener(this);
      }

      validate();

  }

  public class imgPanel extends JPanel{
    public imgPanel(){
      setBounds(0,0,600,600);

    }

    @Override
    public void paint (Graphics g){
      super.paint(g);
      g.drawImage(I, 0, 0, this);
    }
```

```
    }

    public void plotPoints(){
      I = new BufferedImage(getWidth(), getHeight(), BufferedImage. ↩
          TYPE_INT_RGB);
        for (int y = 0; y < getHeight(); y++) {
            for (int x = 0; x < getWidth(); x++) {
                zx = zy = 0;
                cx = (x - 320+xMove) / zoom;
                cy = (y - 290+yMove) / zoom;
                int iter = numIter;
                while (zx * zx + zy * zy < 4 && iter > 0) {
                    temp = zx * zx - zy * zy + cx;
                    zy = 2 * zx * zy + cy;
                    zx = temp;
                    iter--;
                }
                I.setRGB(x, y, iter | (iter << colorIter));
            }
        }
    }

    public void actionPerformed(ActionEvent ae){
      String event = ae.getActionCommand();

      switch (event){
      case "up":
        yMove-=100;
        break;
      case "down":
        yMove+=100;
        break;
      case "left":
        xMove-=100;
        break;
      case "right":
        xMove+=100;
        break;
      case "+":
        zoom+=zoomIncrease;
        zoomIncrease+=100;
        break;
      case "-":
        zoom-=zoomIncrease;
        zoomIncrease-=100;
        break;
      case ">":
        colorIter++;
        break;
      case "<":
```

```
        colorIter--;
        break;
      }



      plotPoints();
      validate();
      repaint();
    }



    public static void main(String[] args) {
        new Mandelbrot().setVisible(true);
    }
}
```

# 6. fejezet

# Helló, Welch!

## 6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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!

## 6.2. LZW

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

Megoldás videó:

Megoldás forrása:

## 6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

## 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:

## 6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

## 6.6. Mozgató szemantika

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

Megoldás videó:

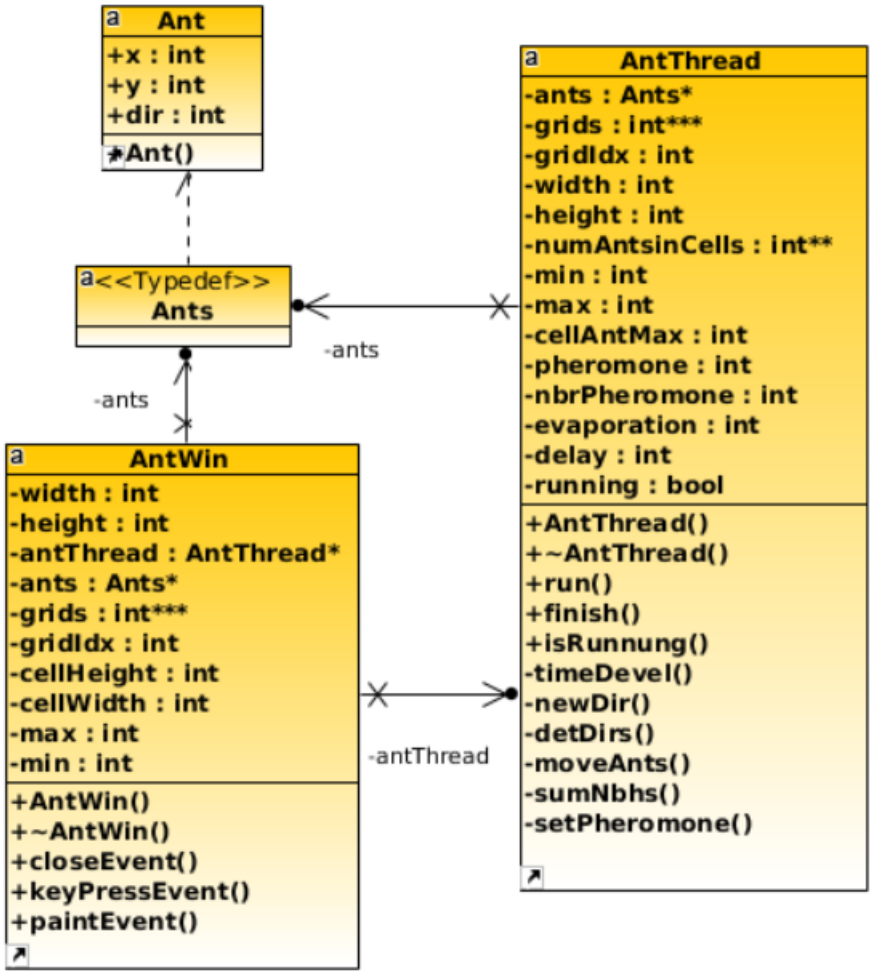
Megoldás forrása:

# 7. fejezet

# Helló, Conway!

## 7.1. Hangyaszimulációk

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




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

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

Ebben a programban a hangyák viselkedését és mozgását fogjuk reprezentálni. Mint tudjuk a hangyák feromon nyomokat hagynak maguk után hogy megjelöljék az útvonalukat és ha egy másik hangya belebotlik ebbe a feromon nyomba, akkor azt kezdi el követni. Ez segíti őket a tájékozódásban. Ám az elhagyott feromon nyomok kezdetben zöld szinűek de elkezdenek halványodni abban az esetben ha egy másik hangya rá nem téved a nyomra. Ebben a programban pont ez fogjuk megcsinálni. Elég összetett programról van szó így a megszokottaktól eltérően itt több scriptre van szükségünk. Az első fájlunk neve az ant.h, melyben a hangyák mozgását és koordinátájukat adjuk meg(x,y,dir). Típusdefiniálnunk is kell, hisz nem csak egy hangya útját szeretnénk megnézni,hanem sok sok hangyájét.

```
      class Ant
{

public:
    int x;
    int y;
    int dir;

    Ant(int x, int y): x(x), y(y) {

        dir = qrand() % 8;

    }

};

typedef std::vector<Ant> Ants;
```

A következő fájlunkban adjuk meg az ablak tulajdonságát, melyet QMainWindow parancsal hozunk létre. Itt tudjuk megadni hogy az ablak milyen széles és magas legyen pixelekben, a cellwidht és cellheigh pedig a cellák magasságát mutatja meg, ahol a hangyák mozognak. A key eventeket is itt találjuk amik ebben a programban nem nagy számba vannak jelen, van egy pause parancs melyet a P vagy a Q betű lenyomásával érhetünk el, és van a kilépés parancs melyet az ESC lenyomásával érünk el. Itt csak a header filet mutatom meg, de a forrásban elérhető a cpp file is.

Antwin.h

```
      #ifndef ANTWIN_H
#define ANTWIN_H

#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCloseEvent>
#include "antthread.h"
#include "ant.h"

class AntWin : public QMainWindow
{
    Q_OBJECT
```

```
public:
    AntWin(int width = 100, int height = 75,
           int delay = 120, int numAnts = 100,
           int pheromone = 10, int nbhPheromon = 3,
           int evaporation = 2, int cellDef = 1,
           int min = 2, int max = 50,
           int cellAntMax = 4, QWidget *parent = 0);

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {

        antThread->finish();
        antThread->wait();
        event->accept();
    }

    void keyPressEvent ( QKeyEvent *event )
    {

        if ( event->key() == Qt::Key_P ) {
            antThread->pause();
        } else if ( event->key() == Qt::Key_Q
                    ||  event->key() == Qt::Key_Escape ) {
            close();
        }

    }

    virtual ~AntWin();
    void paintEvent(QPaintEvent*);

private:

    int ***grids;
    int **grid;
    int gridIdx;
    int cellWidth;
    int cellHeight;
    int width;
    int height;
    int max;
    int min;
    Ants* ants;

public slots :
    void step ( const int );

};
```

```
#endif
```

Az AntThread végzi a számításokat és itt vannak a program eventjei is, kiszámolja a cellák feromonértékeket és a hangyák irányát is. A main.cpp-ben leírt utasítások szerint ezzel a parancsal tudjuk futtatni a programot:

```
./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s 3 ↩
    -c 22
```

A betűk különféle kapcsolókat jelentenek:

- A cellák beállítását(szélességét,magasságát, és azt hogy hány oszlopból álljon) a -w és az -m kapcsolóval tudjuk változtatni

- A hangyák számát az -n kapcsolóval.

- A -t kapcsolóval azt tudjuk beállítani hogy a hangyák léptei milyen gyakoriak legyenek.

- A feromonok párolgását a -p kapcsolóval állítjuk be.

- A -d kapcsolóval állítjuk be azt hogy mennyivel nőjjön a feromonértéke a celláknak amint egy hangya belép abba.

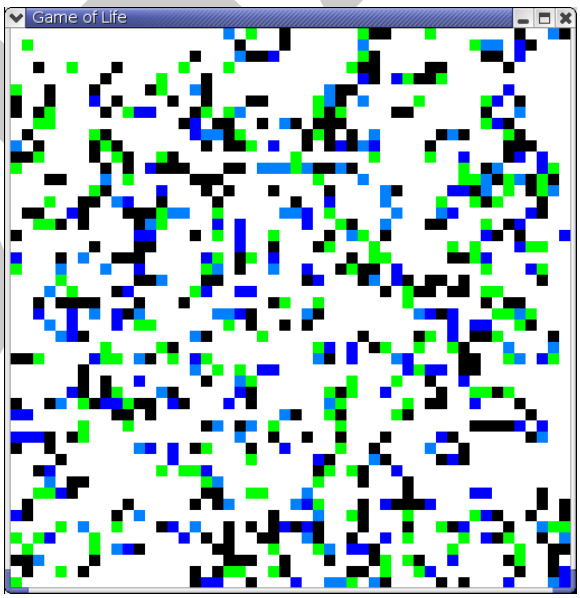
- A feromonok minimumát és maximumát az -a és az -i kapcsolóval szabályozzuk

- A -s kapcsolóval azt állítjuk be hogy mennyi feromont hagyjanak a szomszédos cellákban a hangyák.

## 7.2. Java életjáték

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

Megoldás forrása: https://www.algosome.com/articles/conway-game-of-life-2d.html




Conway 1970-ben az egyik legérdekesebb sejtautomata rendszert készítette el, amit életjátéknak nevezett el, mert az egész program hasonlított az élő szervezetek viselkedésére. Gondolunk itt a születésükre,halálukra,fejlődésükre vagy éppen visszafejlődésükre. A program egy nagy négyzethálóból áll,egy cellában csak egy sejt élhet(!), és minden sejtet 8 szomszédos cella vesz körül, 4 átlósan, 4 pedig keresztbe.

Conway olyan szabályokat akart hogy azok ne legyenek előre megjósolhatóak, ezért sokat kisérletezett mire kiválasztotta a megfelelőeket. A sejtek generációról generációra változnak. Az első szabály a "Túlélés" ami azt jelenti hogy minden sejt aminek kettő vagy három szomszédja van az megmenekült és tovább léphet a következő generációba. A második szabály a "Halál", ez ugye az első szabályból következve úgy működik hogy minden sejt aminek négy vagy több szomszédja van vagy kettő és annál kevesebb azok meghalnak és nem mehetnek tovább a következő generációba. A harmadik szabály pedig a "Születés". Ha egy cellának pontosan három élő sejt van a szomszédjá-ban, akkor ott új sejt születik. És ezeket a szabályokat követve folytatódik a végtelenségbe a program.

```
import javax.swing.JFrame;
import javax.swing.JPanel;

import java.awt.Color;
import java.awt.Font;
import java.awt.FontMetrics;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Rectangle;
import java.awt.Shape;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.ImageObserver;
import java.text.AttributedCharacterIterator;
import java.util.ArrayList;
import java.awt.Event;

public class game_of_life extends JFrame {
    RenderArea ra;
    private int i;

    public game_of_life() {
        super("Game of Life");
        this.setSize(1005, 1030);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
        this.setResizable(false);
        ra = new RenderArea();
        ra.setFocusable(true);
        ra.grabFocus();
        add(ra);

        ra.edit_mode = true;
        ra.running = true;
    }

    public void update() {
        ArrayList<ArrayList<Boolean>> entities = new ArrayList<ArrayList< ↩
           Boolean>>();// = ra.entities;
```

```
        int size1 = ra.entities.size();
        int size2 = ra.entities.get(0).size();
        for(int i=0;i<size1;i++)
        {
            entities.add( new ArrayList<Boolean>());
            for(int j=0;j<size2;j++)
            {
                int alive = 0;

                if(ra.entities.get((size1+i-1)%size1).get((size2+j-1)%size2 ↩
                   )) alive++;
                if(ra.entities.get((size1+i-1)%size1).get((size2+j)%size2)) ↩
                    alive++;
                if(ra.entities.get((size1+i-1)%size1).get((size2+j+1)%size2 ↩
                   )) alive++;

                if(ra.entities.get((size1+i)%size1).get((size2+j-1)%size2)) ↩
                    alive++;
                if(ra.entities.get((size1+i)%size1).get((size2+j+1)%size2)) ↩
                    alive++;

                if(ra.entities.get((size1+i+1)%size1).get((size2+j-1)%size2 ↩
                   )) alive++;
                if(ra.entities.get((size1+i+1)%size1).get((size2+j)%size2)) ↩
                    alive++;
                if(ra.entities.get((size1+i+1)%size1).get((size2+j+1)%size2 ↩
                   )) alive++;



                /*for(int k=-1;i<2;k++)
                {
                    for(int l = -1; l < 2 ;l++)
                    {
                        if(!(k==0 && l == 0))
                        {
                            if(ra.entities.get((size1+i+k)%size1).get(( ↩
                               size2+j+l)%size2)) alive++;
                        }
                    }
                }*/

                if(ra.entities.get(i).get(j))
                {
                    if(alive < 2 || alive > 3)
                    {
                        //ra.entities.get(i).set(j,false);
                        entities.get(i).add(false);
                    }
                    else
```

```
                    {
                        entities.get(i).add(true);
                    }
                }
                else
                {
                    if(alive == 3)
                    {
                        //ra.entities.get(i).set(j,true);
                        entities.get(i).add(true);
                    }
                    else
                    {
                        entities.get(i).add(false);
                    }
                }

            }
        }
        ra.entities = entities;

    }

    class RenderArea extends JPanel implements KeyListener {
        public ArrayList<ArrayList<Boolean>> entities;

        public int diff;
        public boolean edit_mode;
        public boolean running;
        public RenderArea() {
            super();
            setSize(1000, 1000);
            setVisible(true);
            setBackground(Color.WHITE);
            setForeground(Color.BLACK);
            setLocation(0, 0);

            diff = 20;


            this.addMouseListener((MouseListener) new MouseListener(){

                @Override
                public void mouseReleased(MouseEvent arg0) {

                }

                @Override
                public void mousePressed(MouseEvent arg0) {
                    clicked(arg0);
```

```
            }

            @Override
            public void mouseExited(MouseEvent arg0) {

            }

            @Override
            public void mouseEntered(MouseEvent arg0) {

            }

            @Override
            public void mouseClicked(MouseEvent arg0) {

            }
        });
        this.addKeyListener(this);
        entities = new ArrayList<ArrayList<Boolean>>();
        for(int i=0;i<1000/diff;i++)
        {
            entities.add(new ArrayList<Boolean>());
            for(int j=0;j<1000/diff;j++)
            {
                entities.get(i).add(false);
            }
        }

    }

    void clicked(MouseEvent arg0)
    {
        System.out.println("Button "+(arg0.getButton()== 1 ? "Left" : " ↩
           Right"));
        System.out.println("X:"+arg0.getX()/diff);
        System.out.println("Y:"+arg0.getY()/diff);
        if(edit_mode)
        {
            entities.get(arg0.getX()/diff).set(arg0.getY()/diff,! ↩
               entities.get(arg0.getX()/diff).get((arg0.getY()/diff)));
            this.update(this.getGraphics());
        }

    }


    @Override
    public void keyTyped(KeyEvent e) {
        //System.out.println(e.getKeyChar());
    }
```

```
@Override
public void keyReleased(KeyEvent e) {
    System.out.println("Key pressed:"+e.getKeyChar());
    if(e.getKeyChar()=='e')
    {
        edit_mode = !edit_mode;
    }
    else if(e.getKeyChar()=='q')
    {
        this.running = false;
    }
    else if(e.getKeyChar()=='c')
    {
        if(edit_mode)
        {
            for(int i=0;i<this.entities.size();i++)
            {
                for(int j=0;j<this.entities.get(1).size();j++)
                {
                    this.entities.get(i).set(j,false);
                }
            }
            this.update(this.getGraphics());
        }

    }

}

@Override
public void keyPressed(KeyEvent e) {
    //System.out.println(e.getKeyChar());
}

@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.clearRect(0, 0, 1000, 1000);
    for(int i=0;i<1000;i+=diff)
    {
        g.drawLine(i, 0, i, 1000);
    }
    for(int j=0;j<1000;j+=diff)
    {
        g.drawLine(0, j, 1000, j);
    }
    for(int i=0;i<1000;i+=diff)
    {
        for(int j=0;j<1000;j+=diff)
```

```
                {
                    if(entities.get(i/diff).get(j/diff))
                    {
                        g.setColor(Color.BLACK);

                    }
                    else
                    {
                        g.setColor(Color.WHITE);
                    }

                    g.fillRect(i+2, j+2, diff-3, diff-3);
                }
            }
        }



        private static final long serialVersionUID = 1L;

    }

    private static final long serialVersionUID = 1L;
    public static void main(String args[])
    {
        game_of_life gol = new game_of_life();
        while(gol.ra.running)
        {
            if(!gol.ra.edit_mode)gol.update();
            try{Thread.sleep(200);}
            catch(Exception ex)
            {

            }
            gol.ra.update(gol.ra.getGraphics());
        }
        gol.dispose();
    }
}
```

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/sejtautomata/

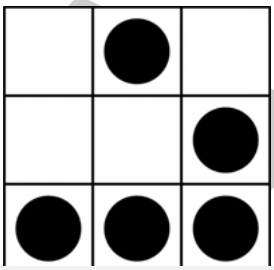
Ez a program ugyan az mint az egyel fentebbi csak itt Qt-ben kell megírnunk, ami azt jelenti hogy a qmake -project parancs kell ahhoz hogy legeneráljuk az Eletjatek.pro fájlunkat. Amint ezt a parancsot beütöttük kapunk egy Makefile-t amit make parancsal tudunk lefordítani:

```
$ qmake -project $ qmake Eletjatek.pro $ make $ ./Eletjatek
```

A fenti linkben elérhető az összes file ami szükséges a programunk futtatásához ezért nem linkelem be mindet egyessével, csak a main-t hogy megmutassam hogy a megszokottakhoz eltérően milyen rövid.

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

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

    return a.exec();
}
```

Ez pedig a siklókilövő mely egy sikló alakzatot indít el ami x generáció után ugyan azt a formát nyeri vissza, ez a sikló lett később a hackerek szimbóluma.



```
        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;

}
```

## 7.4. BrainB Benchmark

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

Rengeteg felméréssel, tesztel és benchmark programmal találkozhatunk az interneten amik feltérképezik és megmutatják a képességeinket. Találhatunk aim fejlesztő vagy épp reakcióidő fejlesztő programokat is. A BrainB Benchmark is ilyen csak ez azt mutatja meg hogy mennyire vagyunk képesek követni a saját karakterünket egy játékban. A program elindításakor pár négyzet jelenik meg, bennük egy-egy kék kör-rel a közepükön. Az első feladatunk hogy a Samu nevű négyzetet megkeressük es a kurzort ráhúzzuk a benne lévő kék kör-re és rajta kell tartanunk a kurzorunkat úgy hogy közbe az egeret lenyomva tartjuk. A felmérés 10 percig tart, ezalatt újabb négyzetek jelennek meg és mindegyik mozog összevissza, a Samu nevű is. Az elején könnyen tudjuk követni, ám miután több négyzet is megjelenik, azután már egyre nehezebb dolgunk van. A program azt méri hogy mennyi ideig tartottuk a kurzort rajta a kék kör-ön és ha elveszítettük akkor

mennyi idő alatt találtuk meg. A mérés 10 percig tart és ha végeztünk az eredményt egy új fájlba kapjuk meg.

A forrásban megtaláljuk a scripteket és a hozzá tartozó header file-kat is. A main.cpp:

```
#include <QApplication>
#include <QTextStream>
#include <QtWidgets>
#include "BrainBWin.h"

int main ( int argc, char **argv )
{
        QApplication app ( argc, argv );

        QTextStream qout ( stdout );
        qout.setCodec ( "UTF-8" );

        qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( "  ↩
           Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;

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

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

        qout << QString::fromUtf8 ( "Ez a program szabad szoftver;  ↩
           terjeszthető illetve módosítható a Free Software" ) << endl;
        qout << QString::fromUtf8 ( "Foundation által kiadott GNU General  ↩
           Public License dokumentumában leírtak;" ) << endl;
        qout << QString::fromUtf8 ( "akár a licenc 3-as, akár (tetszőleges) ↩
            későbbi változata szerint.\n" ) << endl;

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

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

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

# 8. fejezet

# Helló, Schwarzenegger!

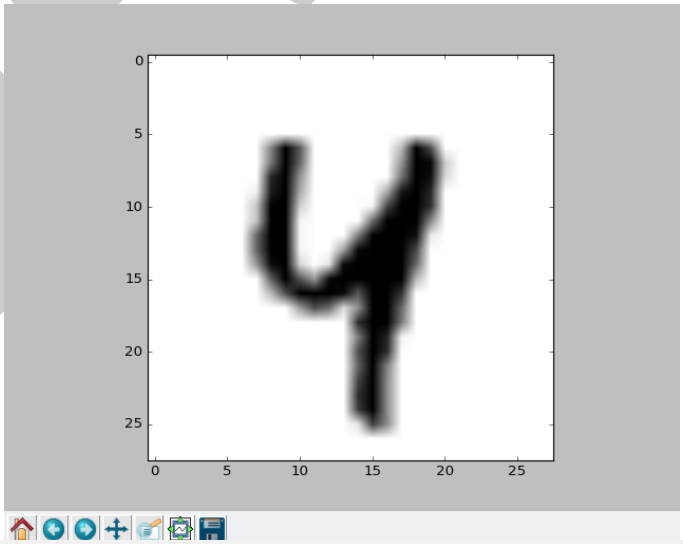
## 8.1. Szoftmax Py MNIST

Python

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

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

Megint egy új programmal fogunk megismerkedni, mégpedig a Tensor flow-al amit a google hozott létre a tanulás fejlesztése érdekébe. Tanulási algoritmusok leírására használ, de az informatika egyre több területén használják. A jelenlegi programunk lényege és hogy a kézírást felismerje, tehát rajzolunk egy felületre egy betűt vagy akármit és a programunknak fel kell/kéne ismernie hogy mi az. De mind ezt úgy hogy előtte úgymond "tanítjuk" a programunkat tesztköröket futtatunk. Az első lépésünk az hogy importáljuk a tensorflow-t az import tensorflow as tf parancsal. Majd az x_train, y_train, x_test és y_test parancsokkal a lefuttatott tesztképeket eltároljuk amit később át fogounk alakítani float típusra. Ezután pedig beállítjuk a szinek értékét az x_train/= 255 és az x_test/= 255 parancsokkal.




```
          import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
x_train.shape

# Reshaping the array to 4-dims so that it can work with the Keras API
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)
# Making sure that the values are float so that we can get decimal points ↩
   after division
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
# Normalizing the RGB codes by dividing it to the max RGB value.
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print('Number of images in x_train', x_train.shape[0])
print('Number of images in x_test', x_test.shape[0])

# Importing the required Keras modules containing model and layers
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Conv2D(28, kernel_size=(3,3), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(128, activation=tf.nn.relu))
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x=x_train,y=y_train, epochs=10)

model.evaluate(x_test, y_test)

image_index = 4444
plt.imshow(x_test[image_index].reshape(28, 28),cmap='Greys')
pred = model.predict(x_test[image_index].reshape(1, 28, 28, 1))
print(pred.argmax())
print(y_test[image_index])
```

## 8.2. Mély MNIST

Python

Megoldás forrása:https://github.com/tensorflow/tensorflow/tree/r1.4/tensorflow/examples/tutorials/mnist

A helyzet szinte ugyan az mint az előző feladatunknál, először importálnunk kell itt is a tensorflow-t a from tensorflow.examples.tutorials.mnist import input_data parancsal, majd mint a kódunkba is látjuk 4 függvényt kell definiálnunk a define parancsal. Viszont annyiban eltér az előző feladattól hogy itt 2 réteggel dolgozunk, az első ami a számításokat végzi, a második pedig az első által elvégzett számításokkal dolgozik így pontosabb a programunk. Létrehozzuk a neurális hálónkat, amiből 5 van. És amint ezeket a hálókat megcsináltuk megadjuk hogy mit is akarunk kapni végeredményként. Majd a Training algorithm parancsal megadjuk hogy mit is tanítunk a programunknak.

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

import argparse
import sys
import tempfile

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

FLAGS = None


def deepnn(x):
  """deepnn builds the graph for a deep net for classifying digits.
  Args:
    x: an input tensor with the dimensions (N_examples, 784), where 784 is ↩
       the
    number of pixels in a standard MNIST image.
  Returns:
    A tuple (y, keep_prob). y is a tensor of shape (N_examples, 10), with ↩
       values
    equal to the logits of classifying the digit into one of 10 classes ( ↩
       the
    digits 0-9). keep_prob is a scalar placeholder for the probability of
    dropout.
  """
  # Reshape to use within a convolutional neural net.
  # Last dimension is for "features" - there is only one here, since images ↩
     are
  # grayscale -- it would be 3 for an RGB image, 4 for RGBA, etc.
  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)

  # 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)

  # Fully connected layer 1 -- after 2 round of downsampling, our 28x28 ↩
     image
  # is down to 7x7x64 feature maps -- maps this to 1024 features.
  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)

  # Dropout - controls the complexity of the model, prevents co-adaptation ↩
     of
  # features.
  with tf.name_scope('dropout'):
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

  # Map the 1024 features to 10 classes, one for each digit
  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
  return y_conv, keep_prob


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')
```

```
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)


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

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])

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

  # Build the graph for the deep net
  y_conv, keep_prob = deepnn(x)

  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}))

if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument('--data_dir', type=str,
                      default='/tmp/tensorflow/mnist/input_data',
                      help='Directory for storing input data')
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

## 8.3. Minecraft-MALMÖ

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

Megoldás forrása:https://github.com/Microsoft/malmo

```
from __future__ import print_function
#  ↩
   ------------------------------------------------------------------------------------------------

# Copyright (c) 2016 Microsoft Corporation
#
# Permission is hereby granted, free of charge, to any person obtaining a  ↩
   copy of this software and
# associated documentation files (the "Software"), to deal in the Software  ↩
   without restriction,
# including without limitation the rights to use, copy, modify, merge,  ↩
   publish, distribute,
# sublicense, and/or sell copies of the Software, and to permit persons to  ↩
   whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included  ↩
   in all copies or
# substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS  ↩
   OR IMPLIED, INCLUDING BUT
# NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A  ↩
   PARTICULAR PURPOSE AND
```

```
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE  ↩
   LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR  ↩
   OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN ↩
    THE SOFTWARE.
#  ↩
   ------------------------------------------------------------------------------------------------




from builtins import range
import sys
import os
import random
import time
import uuid
#from PIL import Image
import json
import math

# Allow MalmoPython to be imported both from an installed
# malmo module and (as an override) separately as a native library.
try:
    import MalmoPython
    import malmoutils
except ImportError:
    import malmo.MalmoPython as MalmoPython
    import malmo.malmoutils as malmoutils


class MissionTimeoutException(Exception):
    pass


def restart_minecraft(world_state, agent_host, client_info, message):
    """Attempt to quit mission if running and kill the client"""
    if world_state.is_mission_running:
        agent_host.sendCommand("quit")
        time.sleep(10)
    agent_host.killClient(client_info)
    raise MissionTimeoutException(message)


def run(argv=['']):
    if "MALMO_XSD_PATH" not in os.environ:
        print("Please set the MALMO_XSD_PATH environment variable.")
        return
    #forceReset="true"
```

```
missionXML='''<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
        <Mission xmlns="http://ProjectMalmo.microsoft.com" xmlns:xsi=" ↩
           http://www.w3.org/2001/XMLSchema-instance">

          <About>
            <Summary>Hello world!</Summary>
          </About>

          <ServerSection>
            <ServerHandlers>
              <DefaultWorldGenerator forceReset="true" />
              <ServerQuitFromTimeUp timeLimitMs="30000"/>
              <ServerQuitWhenAnyAgentFinishes/>
            </ServerHandlers>
          </ServerSection>


          <AgentSection mode="Survival">
            <Name>MalmoTutorialBot</Name>
            <AgentStart>
                <Inventory>
                    <InventoryItem slot="8" type="diamond_pickaxe"/>
                </Inventory>
            </AgentStart>
            <AgentHandlers>
                <ObservationFromFullStats/>
                <ObservationFromGrid>
                    <Grid name="all_the_blocks" >
                        <min x="-1" y="-1" z="-1"/>
                        <max x="1" y="2" z="1"/>
                    </Grid>
                </ObservationFromGrid>
                <ContinuousMovementCommands turnSpeedDegs="180"/>
            </AgentHandlers>
          </AgentSection>
        </Mission>'''

malmoutils.fix_print()

#agent_host = MalmoPython.AgentHost()
agent_host = MalmoPython.AgentHost()
malmoutils.parse_command_line(agent_host, argv)

my_mission = MalmoPython.MissionSpec(missionXML, True)
my_mission.timeLimitInSeconds( 300 )
my_mission.requestVideo( 640, 480 )

#my_mission.rewardForReachingPosition( 19.5, 0.0, 19.5, 100.0, 1.1 )
```

```
    my_mission_record = malmoutils.get_default_recording_object(agent_host, ↩
        "saved_data")

    # client_info = MalmoPython.ClientInfo('localhost', 10000)
    client_info = MalmoPython.ClientInfo('127.0.0.1', 10000)
    pool = MalmoPython.ClientPool()
    pool.add(client_info)

    experiment_id = str(uuid.uuid1())
    print("experiment id " + experiment_id)

    max_retries = 3
    max_response_time = 60  # seconds

    for retry in range(max_retries):
        try:
            agent_host.startMission(my_mission, pool, my_mission_record, 0, ↩
                experiment_id)
            break
        except RuntimeError as e:
            if retry == max_retries - 1:
                print("Error starting mission:",e)
                exit(1)
            else:
                time.sleep(2)

    print("Waiting for the mission to start", end=' ')
    start_time = time.time()
    world_state = agent_host.getWorldState()
    while not world_state.has_mission_begun:
        print(".", end="")
        time.sleep(0.1)
        if time.time() - start_time > max_response_time:
            print("Max delay exceeded for mission to begin")
            restart_minecraft(world_state, agent_host, client_info, "begin  ↩
               mission")
        world_state = agent_host.getWorldState()
        for error in world_state.errors:
            print("Error:",error.text)
    print()

    last_delta = time.time()



    # main loop:
    #agent_host.sendCommand( "jump 1")
    TURN = 0
    TURN2 = 0
    JUMP = 0
```

```
    while world_state.is_mission_running:
        print("New Iteration")

        if JUMP > 0:
            JUMP = JUMP - 1
        if JUMP == 0:
            agent_host.sendCommand( "jump 0" )
            JUMP = JUMP - 1
        agent_host.sendCommand( "move 1" )
        if math.sin(TURN)/3 >= 0:
            agent_host.sendCommand( "turn 0.15")
        else:
            agent_host.sendCommand( "turn -0.2")
        print(TURN," ",math.sin(TURN))
        TURN = TURN + 0.3

        #agent_host.sendCommand( "jump 1" )
        time.sleep(0.5)
        world_state = agent_host.getWorldState()
        y = json.loads(world_state.observations[-1].text)

        #print(y["all_the_blocks"])
        dir = ""
        if y["Yaw"] + 180 < 90:
            dir="S"
            print("Facing South")
        elif y["Yaw"] < 180:
            dir="W"
            print("Facing West")
        elif y["Yaw"] < 270:
            dir="N"
            print("Facing North")
        else:
            dir="E"
            print("Facing East")

        blocks = [[],[],[],[]]
        i=0
        for x in y["all_the_blocks"]:
            blocks[math.floor(i/9)].append(x)
            i = i+1




        if dir=="S":
            willjump = False

            for j in range(0,3):
```

```
                if blocks[1][j] !="air":
                    willjump = True
                print(j,blocks[1][j],willjump)
            if willjump :
                JUMP = 2
                agent_host.sendCommand( "jump 1" )
        elif dir=="W":
            willjump = False

            for j in range(0,3):
                if blocks[1][j*3] !="air":
                    willjump = True
                print(j*3,blocks[1][j*3],willjump)
            if willjump :
                JUMP = 2
                agent_host.sendCommand( "jump 1" )
        elif dir=="E":
            willjump = False

            for j in range(1,4):
                if blocks[1][j*3-1] !="air":
                    willjump = True
                print(j*3-1,blocks[1][j*3-1],willjump)
            if willjump :
                JUMP = 2
                agent_host.sendCommand( "jump 1" )
        elif dir=="N":
            willjump = False

            for j in range(0,3):
                if blocks[1][j] !="air":
                    willjump = True
                print(j,blocks[1][j+6],willjump)
            if willjump :
                JUMP = 2
                agent_host.sendCommand( "jump 1" )

        if (blocks[1][2] !="air" and blocks[2][2] !="air" or blocks[1][4] ↩
           !="air" and blocks[2][4] !="air" or
            blocks[1][2] !="air" and blocks[2][2] !="air" or blocks[1][4] ↩
               !="air" and blocks[2][4] !="air"):
            TURN2 = 2

        if TURN2 >= 0:
            agent_host.sendCommand( "turn 1" )
            TURN2 = TURN2 - 1


        '''if blocks[1][5] != "air" or  blocks[1][5] != "grass" or blocks ↩
           [1][5] != "tallgrass" :
```

```
            JUMP = 2
            agent_host.sendCommand( "jump 1" )
            print()
            print(blocks[1][5])'''

        #print(len(blocks))
        #print(blocks)

        if (world_state.number_of_video_frames_since_last_state > 0 or
           world_state.number_of_observations_since_last_state > 0 or
           world_state.number_of_rewards_since_last_state > 0):
            last_delta = time.time()
        else:
            if time.time() - last_delta > max_response_time:
                print("Max delay exceeded for world state change")
                restart_minecraft(world_state, agent_host, client_info, " ↩
                   world state change")
        for reward in world_state.rewards:
            print("Summed reward:",reward.getValue())
        for error in world_state.errors:
            print("Error:",error.text)
        for frame in world_state.video_frames:
            print()
            #print("Frame:",frame.width,'x',frame.height,':',frame.channels ↩
               ,'channels')
            #image = Image.frombytes('RGB', (frame.width, frame.height),  ↩
               bytes(frame.pixels) ) # to convert to a PIL image
    print("Mission has stopped.")


if __name__ == "__main__":
run(sys.argv)
```

A világ egyik leghíresebb játékáról lesz most szó, a Minecraftról. A microsoft által készített program lényege az hogy a karakterünk akit Steve-nek hívnak egy megadott időlimiten belül ne menjen neki semminek és ne akadjon el semmiben. Tehát vagy kikerülje azokat vagy elrombolja. Steve körül 26 kockát vizsgálunk. A karakterünk elindul és ha akadályba ütközik először megpróbálja kikerülni, ha ez nem lehetséges akkor az ugrást próbálja meg. Az akadályok lehetnek különféle növényzet vagy akár víz vagy láva is. A rombolás meg csak akkor jön létre ha 2 blocknál magasabb az adott akadály hisz a minecraftba már 2 block magasra nem tudunk ugrani, így a karakterünk kénytelen elrombolni azokat a blockokat a továbbjutáshoz.

# 9. fejezet

# Helló, Chaitin!

## 9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Ez az utolsó fejezetünk ebben a félévben, itt a Lisp programozási nyelvről lesz szó amit először egy sima programozási nyelvként használtak, ám később rájöttek hogy a mesterséges intelligencia kutatásaihoz is tökéletesek. A neve a List Processing kifejezésből ered. Kicsit eltér az eddig használt nyelvektől, itt az operátorok is máshogy működnek. Itt a változók elé kerülnek, pl: - x y És a kifejezéseket prefix alakban kéri. Meg kell adnunk a rekurzív és az iteratív alakot is.

```
        rekurzív:
            (define (fakt n) (if(< n 1) 1 (* n (fakt(- n 1)))))
            iteratív:
            (define (factorial n)(define (iter product counter)(if (>  ↩
               counter n)product(iter (* counter product) (+ counter 1))))( ↩
               iter 1 1))
```

A programunk ugye faktoriálist számít rekurzív módon ami úgy működik hogy a define-al meghatározzuk a függvényünket, majd egyszerű matematika következik. Azaz n-1 szorozva n-el. Így megkapjuk az eredményünket.

```
        (defun fact(x)
(setq sum 1)
(loop while (> x 1)
    do (
        setq sum (* sum x)
        x (- x 1)
        )
    )
    sum
)
(setq f (read))
(format t "~D! ~D" f (fact f))
```

Itt pedig az iteratív alak látható annyi kölönbséggel hogy itt nem a define-t használjuk, hanem helyette a fact-ot. Amivel ugyan úgy definiáljuk a függvényünket. Maga a számolás a loop while-on belül található meg ami addig megy még a megadott függvényünk kisebb nem lesz a bemenetnél.

## 9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

A Script-fu a GIMP-en belül található, ez egy olyan program ami úgy működik mint a photoshop csak leírva. A kódunk legelején a define-al megadjuk az alapértékeket,méreteket,betűtípust és az ilyen alapbeállításokat. A második define-vel pedig magát a képünket hozzuk létre, itt is megadjuk a méreteket: szélesség, magasság. És ez bármikor szerkeszthető. Hogy átláthatóbb és egyszerűbb legyen a kódunk a step parancsal elválasztjuk a lépéseket. Az első lépésben létrehozunk egy fehér hátteret (gimp-context-set-foreground) majd kiszinezzük feketére szintén az előző parancs segítségével csak a nullák helyére 255-öt írunk, majd írunk rá egy fehér szöveget(set! textfs) amit középre helyezünk (offsets textfs). A második lépésben egy Gauss elmosódást készítünk a gauss-iir parancsal. A harmadik lépésben a képünk minőségével játszunk, beállítjuk a minimumot és a maximumot hogy minél élesebb legyen a képünk/szövegünk. A negyedik lépésben ismét egy Gauss elmosást használunk. Majd az ötödik lépésben a set-color parancs segítségével kijelöljük a fekete layert és a selection -invert-el invertáljuk, így a begépelt szövegünk köré egy keretet kapunk. A hatodik lépésben létrehozunk egy új átlátszó layert a layer-new-image parancsal. A hetedik lépésben pedig a set-gradient parancsal egy szürke átmenetet kapunk amit az egerünk segítségével tudunk módosítani. Nyolcadik lépés, itt a térhatást próbáljuk elérni a szövegünkön a run-moninteractive parancsal. Végül az utolsó lépésben a curves-spline parancsal a szövegünk fémességét tudjuk variálni. Ha valami nem érthető a videóban mindenre válasz kapunk!

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

;(color-curve)

(define (elem x lista)
```

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

)

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

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

    (list text-width text-height)
    )
)

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

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

    ;step 1
    (gimp-image-insert-layer image layer 0 0)
    (gimp-context-set-foreground '(0 0 0))
    (gimp-drawable-fill layer  FILL-FOREGROUND )
    (gimp-context-set-foreground '(255 255 255))

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

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

```
    ;step 2
    (plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)

    ;step 3
    (gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)

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

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

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

    ;step 7
  (gimp-context-set-gradient gradient)
  (gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ↩
     GRADIENT-LINEAR 100 0 REPEAT-NONE
        FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ↩
           3)))

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

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

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

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 '(255 0 0) " ↩
   Crown molding")

(script-fu-register "script-fu-bhax-chrome"
    "Chrome3"
    "Creates a chrome effect on a given text."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 19, 2019"
    ""
    SF-STRING       "Text"      "Bátf41 Haxor"
    SF-FONT         "Font"      "Sans"
    SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
```

```
    SF-VALUE        "Width"     "1000"
    SF-VALUE        "Height"    "1000"
    SF-COLOR        "Color"     '(255 0 0)
    SF-GRADIENT     "Gradient"  "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
    "<Image>/File/Create/BHAX"
)
```


## 9.3. Gimp Scheme Script-fu: név mandala

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

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

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

A feladat igazából ugyan az mint az előző, egy képet kell csinálnunk/szerkesztenünk a Script-fu segítségével, csak itt annyival lesz másabb a helyzet hogy nem egy szöveget szerkesztünk hanem egy kör alakú képet. Ahol a forgatáson lesz a hangsúly mivel a mandala azokból a szövegekből áll amit megadunk, majd ezeket elforgatjuk és keresztezik egymás útját igy kapunk egy szép szimetrikus, szabályos formát. Itt nem lesznek a lépések pontról pontra szétszedve és levezetve, a videón úgyis ott van minden. Csak a fontosabb parancsokat írom le, ilyen például a gimp-image-new width height 0 amellyel létrehozzuk a képünket és megadjuk neki a magasságot a szélességet és a színt is. Majd a front layer színét állítjuk be a set-foreground parancsal és a gimp- draw-fill-el kitöltjük azt. Majd a betűméretet adjuk meg pixelben, aztán jön az a bizonyos forgatás amin a hangsúly van ezt a gimp-item-transform-rotate parancsal érhetjük el. 4 fokozata van ennek a forgatásnak, a 180 fok a pí/2, a pí/4 és a pí/6.

```
   (define (elem x lista)

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

)

(define (text-width text font fontsize)
(let*
```

```
    (
        (text-width 1)
    )
    (set! text-width (car (gimp-text-get-extents-fontname text fontsize  ↩
       PIXELS font)))

    text-width
    )
)

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

    (list text-width text-height)
    )
)


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

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

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

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

  (gimp-context-set-foreground color)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(script-fu-register "script-fu-bhax-mandala"
    "Mandala9"
    "Creates a mandala from a text box."
```

```
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 9, 2019"
    ""
    SF-STRING       "Text"      "Bátf41 Haxor"
    SF-STRING       "Text2"     "BHAX"
    SF-FONT         "Font"      "Sans"
    SF-ADJUSTMENT   "Font size" '(100 1 1000 1 10 0 1)
    SF-VALUE        "Width"     "1000"
    SF-VALUE        "Height"    "1000"
    SF-COLOR        "Color"     '(255 0 0)
    SF-GRADIENT     "Gradient"  "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
    "<Image>/File/Create/BHAX"
)
```

# 10. fejezet

# Helló, Gutenberg!

## 10.1. Juhász István - Magas szintű programozási nyelvek

Programozni megtanulni csakis úgy lehet ha nagyon sokat gyakorlunk, folyamatosan megállás nélkül csináljuk. Próbálgatjuk az újabb és újabb forráskódokat és kíváncsian vágunk bele új, saját, általunk készített feladatba is. Mindenki nagyra álmodik és jó programozó akar lenni, de ehez tenni kell. Az első ilyen teendő az hogy megtanuljuk az alapfogalmakat amik a programozáshoz kapcsolódnak. Így egyszerűbb lesz megértenünk mindent. Vágjunk is bele. Tehát a programozásnak 3 féle nyelvcsaládját különbüztetjük meg. Az első ilyen nyelv a gépi nyelv, az Assembly nyelv és a Magas szintű nyelvek. Az utolsó nyelvcsaládhoz forráskód szükséges amit megírunk és ehez egy fordító program mely lefordítja a forráskódunkat és beviszi és lefutattja azt a számítógépünkön. Egy ilyen fordítóprogram 4 lépésen megy keresztül ilyenkor mikor megkap egy forráskódot. Először is lexikálisan elemzi, majd miután ez megvan szintaktikai elemzést is végrehajt, majd szemantikai elemzést, és legvégül egy kódgenerálást. Tehát először a forráskódot lexikálisan elemeire bontja majd utána szintaktikailag elemzi azokat az elemekt hogy minden stimmel-e, és helyesek e és teljeseülnek -e az adott nyelven megírt forráskód szabályai. Ezenbelül a programnyelveket még 2 külön fajtába osztjuk. Az első ilyen Fajta az imperatív nyelvek, amelyek algoritmikusok által működnek. Egy programozó megír egy ilyen algoritmust és ez működteti a processzort, az általa írt utasítások és szabályok szerint, emellett ennek a nyelvnek vannak még alcsoportjai is, például az eljárásorientált nyelvek és az objektumorientált nyelvek. Majd jön a másik fajta, amely a deklaratív nyelv Ezekkel csak problémát írunk le, melyekre magába a nyelvbe beépített megoldásokat keressük. Ennek a nyelvnek is vannak alcsoportjai, például a fukncionális nyelvek és a logikai nyelvek. A könyv következő részében a kifezezésekről volt szó, Tehát a kifejezések egyfajta szintaktikai eszközök melyek a programunk egy pontján egy megadott értékből egy új, másik értéket hoz létre, ennek két összetevője van: az érték és a típus. Egy kifejezés összesen 3 részből áll. Az első ilyen rész az Operandusok melyek a kifejezésben az értéket képviselik, például:konstans, vagy változó. A második ilyen összetevő az Operátorok melyek műveleti jelek, nem kell itt olyan bonyolult dolgokra gondolni. Sima összeadás kivonás szorzás osztás amit ovodás korunk óta használunk. A harmadik összetevő pedig a zárójelek Melyek segítenek elkülöníteni és rendezni a kifejezéseinket, és emellett a műveletek sorrendjét is ezzel tudjuk megadni. Ha alak szempontból vizsgáljuk őket akkor 3 féle van..mindig 3..valahogy nagyon különleges szám ez az informatikában! Tehát visszatérve lehet prefix alakú egy ilyen kifejezés, ami úgy néz ki hogy először a zárójel majd az operátor aztán a számok ( + 5 6), van az infix alak, ami így néz ki ( 3 + 15 ) és legvégül van a postfix ami a prefix fordított alakja, tehát egy a zárójel után egy számmal kezdünk majd az operátor a sor legvégén helyezkedik el. Pontosan így ( 8 5 -). Visszatérünnk a nyelvekhez, mégpedig az eljárásorientált nyelvekhez. Egy adattípust na kitaláljátok hány dolog határoz

meg? igen 3..A tartomány ami a felvehető értékek halmaza. A műveletek ami a tartomány elemein végrehajtható műveleteket tárolja, és a reprezentáció: a tartományba tartozó értékeket jeleníti meg. A nevesített konstantoknak 3 fajtája van. A neve, a típusa és az értéke. Ezek közül mindet egytől egyig deklarálnunk kell. Ezek a nevesített konstansok azt teszik nekünk lehetővé hogy a sokszor előfordulő értékeket elnevezzük ezzel utalva a konstas szerepére. Csoda történt. A változóknak nem 3 hanem 4 komponense van. Az első ilyen komponens a Név Ami a programunkban a változó nevével jelenik meg. Aztán ott vannak az Attribútumok amik olyanfajta jellemzők amelyek a változók viselkedését figyeli írja le és határozza meg. A harmadik ilyen típus a Cím, itt helyezkedik el a változónk értéke. A negyedik és egyben utolsó ilyen komponensünk az érték. A kifejezések meghatározására két típusról fogunk tanulni: az egyik típus az a típusegyenértékűség, a másik pedig a típuskényszerítés. Típusegyenértékűségnél azt mondjuk hogy adott egy két operandusu operátorunk, és neki csak ugyna ilyen típusu operandusa lehet. Ilyenkor a kimenetünk, vagyis az eredményünk vagy olyan lesz mint a két operandus vagy pedig olyan mint az operátoré. A másik esetben a típuskészerítésnél pedig az előzőhöz ellentétes tehát a kétoperanduusú operátornak lehet két különböző típusú operandusa. Utasításokkal adjuk meg az algoritmusainkat, tehát mi adjuk meg hogy mi hogy működik, hogy mennyi ideig tartson vagy mennyi ideig csináljon ezt azt. Ezeknek két csoportja van, A deklarációs utasítások és a végrehajtó utasítások. A deklarációs utasítások az igazából a fordítóprogramnak szolnak hogy mit csináljon. Ezeket az utasításokat még több alcsoportra osztjuk: például: értékadó,üres,hívó,elágazó és még sok más utasítás csoport létezik. Egy üres utasítás esetén a gépünk ezt az üres utasítást hajtra vége a processzor segítségével. Az elágaztató utasítás pedig mint a nevébe is benne van, abba segít nekünk hogy két dolog kozott, jelen esetben tevékenység közül válasszunk. Ilyenek például az IF vagy a THEN vagy az ELSE Az IF azért jó utasítás és azért használják nagyon sokat a programozók mert egymásba fűzhetőek. Az elset magába nem nagyon használjuk, inkább az IF-el együtt hogy IF ... ELSE ... . A ciklusszervező utasításokkal azt tudjuk elérni hogy egy adott tevékenységet akár többször is elvégezhetünk. A feltételes ciklusnál van egy fajta szabályozás, ami az ismétlődést szabályozza és ez pedig a feltétel értékén múlik. A kezdőfeltételes ciklus pedig először megkapja az eredményt majd addig ismételi magát és addig csinálja a megadott utasításokat amíg az az utasítás hamis nem lesz. Aztán jönnek a végfeltételes ciklusok amelyek először lefutnak majd utána értékelődnek ki és ha az utasítás és a feltétel igaz akkor ujra lefutnak ha pedig hamis akkor nem értékelődnek ki. Térjünk át a Vezérlő utasításokra: itt is 3 fajta van. A break: ami a ciklusmagban található és 'lefékezi' azaz megállítja és befejezi a ciklust majd kilép belőle. A return szintén befejezi a ciklust ám ő átadja a vezérlést a hívónak. Majd legvégül a CONTINUE ami szintén a breakhez hasonlóan a ciklusmagban található. Ő az utasításokat nem hatja végre csak vizsgálja a feltételeket.

## 10.2. Kerninghan és Richie

Ebben a konyvben A c programozási nyelvről fogunk olvasni. Magáról a nyelvről tudni kell hogy egy általános célú programozási nyelv, de magasszintűnek egyáltalán nem mondhatjuk, de van aki mégis oda sorolja, hisz vannak olyan dolgai és featurei amik kényelmesebbé, egyszerűbbé ezáltal szerethetőbbé teszi magát mint néhány magas szintű programozási nyelv. Például nincsennek benne bonyolult, összetett műveletek. Például karakterláncok vagy halmazok vagy hasonló. Ha adattípusok szerint vizsgáljuk meg a C nyelvet akkor azt kell mondanunk hogy nagyon keveset használ Pl. double, float, int. Viszont cserébe úgynevezett minősítőket használhatunk amelyek segítik a programozó életét. Vannak állandók is amelyek szintén rendelkeznek a fenti típusokkal és minősítőkkel. Emellett léteznek olyan állandók amikkel karaktereket adunk meg pl kódszámokat. De léteznek karaktersorozatok is, amelyek szintén lehetnek állandók de vannak olyanok is amelyek nem lehetnek állandóak csakis escape sorozattal írhatóak le. C nyelvben nem lehetnek a nyelvek kulcsszavai változónevek és a változó nevek első karakterének mindig muszáj betűnek

lennie! Elágaztató utasításokra az if-else párost használjuk. De az else simán elhagyható. Találkozhatunk a csellengő if problémával is mikor két if-ünk van és egy elsénk és sem a program sem mi nemtudjuk hogy most épp melyik if-hez tartozik az a fránya else. Ezért is van másfajta módszer a többirányú elágazásra. A SWITCH, ami összehasonlítja az értékeket majd végre hajtja az utasításokat. Ciklusszervezés a WHILE és a FOR utasításokkal működik. mintkettőben ott van az ismétlődés. De a For lehet végtelen is akár for (; ; ) ; és a while is ugyan úgy while(1). Két ciklust ha kedvünk van felserélhetünk. Visszatérve a nyelvre hogy mennyire jó is, arra nem tudunk választ adni. Hisz ez mind emberfüggő, aki mondjuk épp C-ben tanult meg programozni annak ez lesz a legjobb programozási nyelv amit mindig szeretni fog és isteníteni, viszont aki már több megtanult programozási nyelv után tér rá erre annak lehet hogy nem fog eggyezni az előzővel a véleménye. Negatívumnak mondanám még azt hogy találhatunk a C nyelvhez egy könyvet amiben ugymond az utasítások featurok vannak leírva Ez mind szép és jó, ám mikor behoznak egy új szabványt akkor az tény hogy több mindenre lesz lehetőségünk. De visszafele nem fognak működni a szabványok. Tehát ha a C99-es szabványba behoztak valami új dolgot, akkor azt a C89-es szabványban nem fogjuk tudni alkalmazni, hisz visszafele nemtudnak működni.

## 10.3. BME: Szoftverfejlesztés C++ nyelven / Benedek Zoltán, Levendovszky Tihamér

Az utolsó könyvünkhöz érkeztünk, amiben a C++ programozási nyelvről fogunk olvasni. Ez a programozási nyelv A C programozási nyelvnek az objektum-orientált típusa. Melyet Bjarne Stroustrup fejlesztett ki. Ez a nyelv is általános célú ami a C nyelvre hajlik és alapoz. Ezért is hasonlítanak nagyon egymásra. Találkozhatunk olyan programokkal forráskódokkal amik szinte egy az egyben ugyan úgy néznek ki és ugyan úgy működnek. De emellett rengetek különbség is van a kétnyelv között, hisz ha nem lenne mi értelme lenne egy új programozási nyelvet megtanulni a C után ami szinte ugyan az. Például a függvényparamétereknél vélhetjük felfedezni ez első ilyen nagyobb kölünbséget. C-ben ha definiálni szeretnénk egy függvényt ami üres paraméterlistával rendelkezik akkor simán megtehetjük akárhány paraméterrel. Viszont ez a C++-ban egy void parancsal egyenlő. Amit megtudunk úgy irni hogy akárhány paraméterrel legyen meghívható, annyi a dolgunk hogy a void-ot kicsit átalakítjuk, méghozzá így : ( void f (...) {}. C-ben a függvényvisszatérési értékét nem adhatjuk meg tehát alapértelmezett lesz ami egy tök jó dolog. Viszont C++-ban nem találkozhatunk ilyen alapértékekkel, itt ehelyett egy hibaüzenet fogunk kapni. C++-ban jobban elengedik a kezünket, szabadabbak vagyunk. Ezt úgy értem hogy itt példál lehetőségünk van arra hogy ha szeretnénk akkor tudunk dekralálni változókat is, amik helyén amugy egy utasításnak kéne állnia. Jó példa még a két nyelv közötti kölünbségre a függvénytúlterhelés. C-ben ilyenre nem volt lehetőségünk, ám C++ annál inkább. C-ben az átadás érték szerint történik, ami azt jelenti hogy kénytelenek vagyunk pointereket használni, hogyha változtatni szeretnénk egy változó már megadott értékein. C++-ban pedig már van lehetőségünk cím szerinti átadásra. Lehetőségünk van kivételkezelésre is, amely olyan parancs, szebb szóval mechanizmus amely segít abba hogy kivételes eseteket is tudjunk kezelni. Például ha van egy programunk és fontos számunkra az hogy a bemenetünk ne 0 legyen akkor használhatjuk ezt. Egy try-catch blokkot kell használnunk. És még rengeteg új és jó tulajdonsága van a C++-nak de nekem ennyire volt időm elolvasni a könyvet, így újabb és hasznosabb dolgot nemtudnék írni vagy mondani. De a C++-t ajánlom mindenkinek, kezdőknek is, és azoknak is akik a C nyelv tanulásánál huzták a szájukat vagy nem szimpatizáltak annyira azzal a nyelvel, mert ezzel tuti fognak!

# III. rész

# Második felvonás

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

# 11. fejezet

# Helló, Arroway!

## 11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 11.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

# IV. rész

# Irodalomjegyzék 2.0

## 11.3. Általános

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

## 11.4. C

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

## 11.5. C++

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

## 11.6. Lisp

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

Köszönet illeti a NEMESPOR, https://groups.google.com/forum/#!forum/nemespor, az UDPROG tanulószoba, https://www.facebook.com/groups/udprog, a DEAC-Hackers előszoba, https://www.facebook.com/-groups/DEACHackers (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.