Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 16, v. 0.0.9.1

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Rádi, Dániel	2019. május 11.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-05-09	Turing feladat befejezése GNU Public License Norbert Bátfai GNU Public License Dániel Rádi	nbatfai
0.0.5	2019-05-09	Chomsky feladat befejezése GNU Public License Norbert Bátfai GNU Public License Dániel Rádi	raddan
0.0.6	2019-05-09	Turing feladat befejezése GNU Public License Norbert Bátfai GNU Public License Dániel Rádi	raddan

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.0.7	2019-05-11	Mandelbrot feladat befejezése GNU Public License Norbert Bátfai, Dániel Rádi, Ben Hiller, Sully Chen	raddan
0.0.8	2019-05-09	Welch feladat befejezése GNU Public License Norbert Bátfai GNU Public License Dániel Rádi	raddan
0.0.9	2019-05-10	Conway feladat befejezése GNU Public License Norbert Bátfai GNU Public License Dániel Rádi	raddan
0.0.10	2019-05-10	Swarzenegger feladat befejezése. GNU Public License: Norbert Bátfai, Dániel Rádi Referencia: "Dr. Búzáné, Dr. Kis Piroska"	raddan
0.0.11	2019-05-10	Chaitin feladat befejezése. GNU Public License Norbert Bátfai GNU Public License Dániel Rádi	raddan

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. Milyen doksikat olvassak el? . . . 2

1.2. 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 . . . 9

2.6. Helló, Google! . . . 10

2.7. 100 éves a Brun tétel . . . 11

2.8. A Monty Hall probléma . . . 12

3. Helló, Chomsky! 14

3.1. Decimálisból unárisba átváltó Turing gép . . . 14

3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 15

3.3. Hivatkozási nyelv . . . 15

3.4. Saját lexikális elemző . . . 15

3.5. l33t.l . . . 16

3.6. A források olvasása . . . 17

3.7. Logikus . . . 18

3.8. Deklaráció . . . 18

4. Helló, Caesar! 20

4.1. int ** háromszögmátrix . . . 20
4.2. C EXOR titkosító . . . 22
4.3. Java EXOR titkosító . . . 23
4.4. C EXOR törő . . . 24
4.5. Neurális OR, AND és EXOR kapu . . . 26
4.6. Hiba-visszaterjesztéses perceptron . . . 28

5. Helló, Gutenberg! 30

5.1. A kivételkezelés alapfogalmai, kivételkezelési módszerek és eszközrendszerek 11. hét . . . 30
5.2. A programnyelvek I/O eszközei, állománykezelés 9. óra . . . 30
5.3. Blokk. Hatáskörkezelés, láthatóság. Fordítási egység. Absztrakt adattípus. Generikus programozás 8. óra . . . 30
5.4. Paraméterkiértékelés, paraméterátadás 7. óra . . . 31
5.5. Programegységek. Hívási lánc, rekurzió 6. óra . . . 31
5.6. Utasítások 5. óra . . . 31
5.7. Kifejezések 3. és 4. óra . . . 31
5.8. Alapfogalmak 2. óra . . . 32
5.9. 1. óra . . . 32

6. Helló, Mandelbrot! 33

6.1. A Mandelbrot halmaz . . . 33
6.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 35
6.3. Biomorfok . . . 36
6.4. A Mandelbrot halmaz CUDA megvalósítása . . . 38
6.5. Mandelbrot nagyító és utazó C++ nyelven . . . 40

7. Helló, Welch! 44

7.1. Első osztályom . . . 44
7.2. LZW . . . 46
7.3. Fabejárás . . . 46
7.4. Tag a gyökér . . . 46
7.5. Mutató a gyökér . . . 47
7.6. Mozgató szemantika . . . 47
7.7. LZW feladatkör kimeneti kép . . . 48

8. Helló, Conway! 49
8.1. Hangyaszimulációk . . . 49
8.2. Java életjáték . . . 50
8.3. Qt C++ életjáték . . . 51
8.4. BrainB Benchmark . . . 53

9. Helló, Schwarzenegger! 55
9.1. Szoftmax Py MNIST . . . 55
9.2. Mély MNIST . . . 59
9.3. Minecraft MALMÖ . . . 59

10. Helló, Chaitin! 63
10.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 63
10.2. Gimp Scheme Script-fu: króm effekt . . . 63
10.3. Gimp Scheme Script-fu: név mandala . . . 67

III. Második felvonás 69

11. 71
11.1. . . . 71

IV. Irodalomjegyzék 72
11.2. Általános . . . 73
11.3. C . . . 73
11.4. C++ . . . 73
11.5. Lisp . . . 73

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. 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.

1.2. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Ha egy végtelen ciklust futtatunk a mag 100% terhelt lesz, ugyanis végtelenszer ismétli önmagát.

```
        #include <omp.h>

int main()
{
 #pragma omp single
    {for(;;){}}
}
```

Ha egy feladat megáll vagy elaltatjuk a sleep paranccsal, a mag 0% terhelt lesz.

```
        #include <unistd.h>
int main()
{
for(;;){usleep ( 100000 );}
}
```

Ha egy végtelent ciklust paralell futtatunk minden magon, minden mag 100% terhelt lesz. -openmp szükséges hozzá

```
        #include <omp.h>

int main()
{
    #pragma omp parallel
    for(;;){}
}
```

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

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

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

  boolean Lefagy2(Program P)
```

```
  {
     if(Lefagy(P))
      return true;
     else
      for(;;);
  }

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tegyük fel, hogy létezik olyan program (T100), ami el tudja dönteni egy másik programról, hogy van-e benne végtelen ciklus. Ha van, akkor megáll a program, ha nincs, akkor pedig végtelen ciklusba kezd. Létrehozunk egy új programot (T1000) az előzőt (T100) felhasználva és ha a T100 megállt, akkor végtelen ciklusba kezd, ha pedig a T100 kezdett végtelen ciklusba, akkor megáll.

A megállás csak akkor lehetséges, ha a T100 nem áll meg, de ez pedig csak akkor lehet, ha a második argumentumként kapott saját programunk megáll.

Ebből ellentmondásra jutottunk, tehát nem lehet ilyen programot írni.

A program amit oda adunk lefagyó, ekkora a programunk igazat kellene visszaadjon, de nem ad vissza semmit mert lefagy.

A program amit oda adunk nem lefagyó, emiatt a program lefagy és nem tud kilépni.

Mivel ez egy paradoxon ezért nem lehet ilyen programot írni.

Amennyiben a programnak önmagát adjuk meg paraméterként P1(P2(P3(P4))), és a program akkor adna ki eredményt ha a bementi program lefagyó, ebben az esetben a kód soha nem jut el a programig. Ez a végtelenségig is lehet ismételni, az eredmény nem változik.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: Valtozo felcserelesek

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
  int a = 1;
  int b = 2;
  string sel(" ");
  cout << "A kezdo szamok: " << a << sel << b << "\n";

  int c = a;
  a = b;
  b = c;
  cout << "Uj valtozos felcserelessel: " << a << sel << b << "\n";

  a = a+b;
  b = a-b;
  a = a-b;
  cout << "Visszacsereles uj valtozo nelkul: " << a << sel << b << "\n";
}
```

Új változóval: Bevezetünk egy új változót amire "elmentjük" az 'a' változón tárolt információt. Ekkor 'a' változót felülirhatjuk 'b' változóval majd a 'b' változót az új változóval.

Új változó nélkül: Összeadjuk a két változót és magkapjuk az összegüket az egyik változón(itt 'a'). A másik változón(itt 'b') kivonjuk az összegből('a') a 'b' változót, a maradék 'a' eredeti érteke lesz. Ez után az összeget tartalmazó változón('a') kivonuk önmagából a 'b' értéket, ami jelenleg 'a' eredeti érteke, így megkapjuk a 'b' eredeti érteket.

2.4. Labdapattogás

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

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

Megoldás forrása: Labdapattogtatas

Megjegyzes: Minden esetben felvesszük a kódhoz szükseges headert, amit a forrasban meg lehet talalni.

Elkészitjuk az ablakot a terminálból es felvesszuk az adatait.

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

Egy végtelen for ciklusba fog a program futni, ugyanis most nem lesz szükség a labda megállitására.

Felveszunk két változot (a for cikluson kivül) az ablak (x,y) méretének tárolására majd belementjük az érteket minden ciklusban. Ezt a forrásban megfeleztük hogy csak az ablak feléig dolgozzunk.

```
getmaxyx ( ablak, amy , amx );
```

Egyszerü forciklussal megrajzoljuk a határértékeket tetszőleges karakterekekkel, illetve a két maxértekkel a legtávolabbi pontot, az ellentkező sarkot. Ennek csak esztétikai jelentősége van.

```
for(int a = 0;a<amx;a++){mvprintw ( amy, a, "=" );}
for(int b = 0;b<amy;b++){mvprintw ( b, amx, "|" );}
mvprintw ( amy, amx, "/" );
```

Felveszünk két új változót a labda X,Y értekeinek tárolására a végtelen cikluson kivül. Ez a labda pozició kezdő érteke. Felrajzoljuk a labdát a jelenlegi poziciójára majd frissitjük az ablakot, várunk hogy a mozgás emberi szemmel is észlelhető legyen majd eltöröljuk az ablakon megjelenitett elemeket.

```
mvprintw ( y, x, "O" );
refresh ();
usleep (100000);
clear();
```

Felveszünk két új változot a labda mozgasi sebessége meghatározására és hozzáadjuk az érteket az előző pozicióhoz.

```
x = x + xmo;
y = y + ymo;
```

Egyszerű if fügvennyel ellenőrizzuk hogy elérte-e a labda a határérteket. Ha elérte a sebességet reciprokara állitjuk, ezzel a labda mozgásat az ellentkező irányba megforditva.

```
if ( x>=amx-1 ) {xmo = xmo * -1;}
if ( x<=0 ) {xmo = xmo * -1;}
if ( y<=0 ) {ymo = ymo * -1;}
if ( y>=amy-1 ) {ymo = ymo * -1;}
```

Amennyiben if függvény nélkül szeretnénk megoldani, azt egyszerü for loopokra kell átalakitani.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Rádi Dániel GNU General Public License.

Megoldás forrása:

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

A változot a bitshift operátorral eltoljuk amig lehetséges és minden lepést megszámolunk. Az eredmény az értek bit értékű mérete. Ez a változó típusától is függ.

2.6. Helló, Google!

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

Megoldás forrása: Pagerank

Az alábbi funkció kiírja a belevezettet értékeket érkezesi sorrend szerint százados helyiértékig.

```
void kiir (double tomb[], int db)
    {
    for(short i = 0; i < db; ++i)
    printf("%.2f \n",tomb[i]);
    }
```

Az alábbi funkció kiszámolja a PageRank és a határérték távolságát a négy oldal jóságának felhasználásával, majd visszaadja azt double értékkent.

```
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);
}
```

A main-ben felvesszük az oldalak linkjeinek számat mátrixként, illetve a pagerankot és a határértéket vektorkent.

Egy for függvényben elkezdjük a számolást. Ezen belül dupla for fügvénnyel összehasonlítjuk a mátrix minden koordinátájának felhasználásával a page "jóságat".

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

Amennyiben a különbseg kisebb mint 0.00000001 akkor kilépünk a végtelen ciklusból, ha nem PRv-t feltöltjuk PR elemeivel. Ha ciklus véget ér, kiiratjuk az értékeket.

```
if(tavolsag (PR, PRv, 4) < 0.00000001)
      break;
    }
```

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

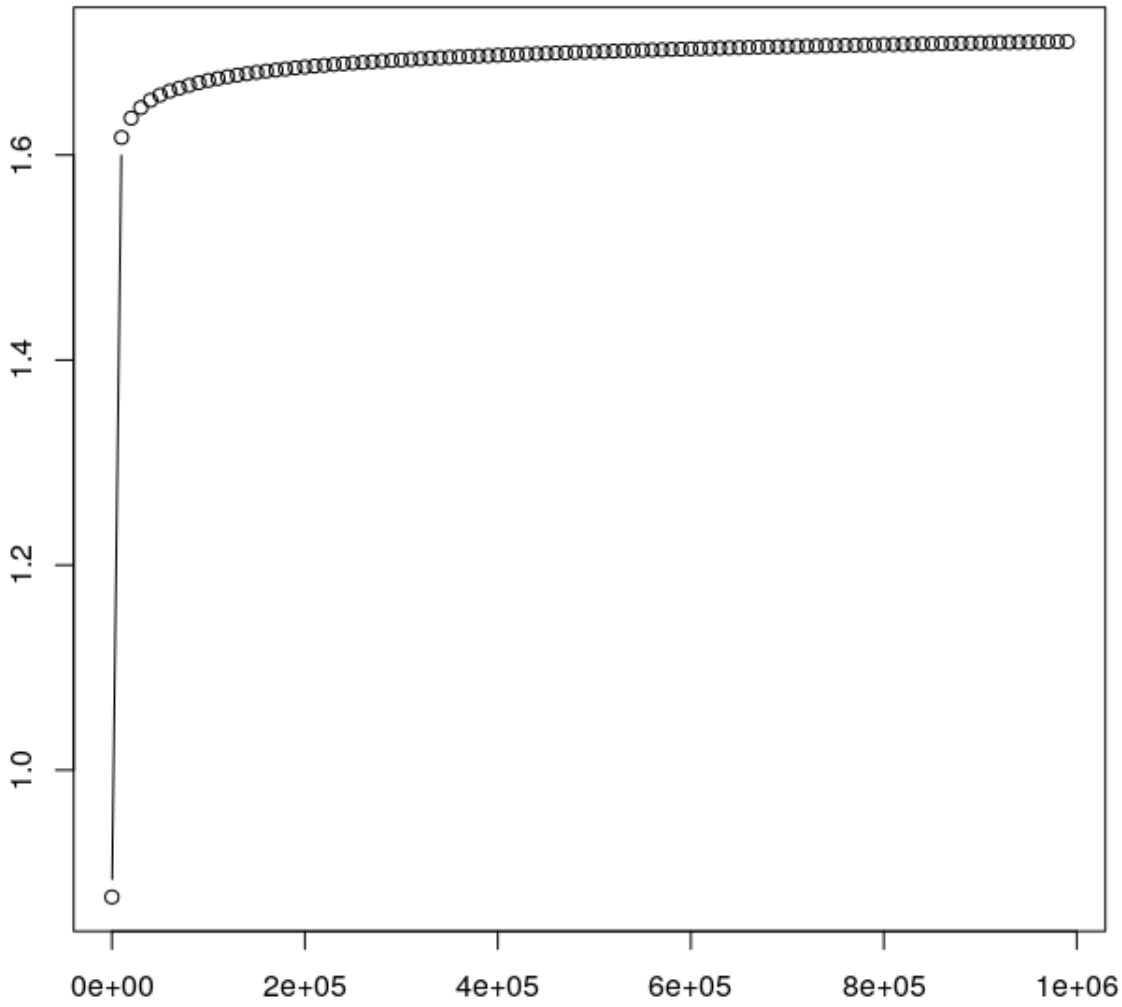
Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R

```
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))
}
```

1. sor kiszámolja a prímeket x-ig.
2. sor az egymást követő prímek különbségét veszi. 3-2, 5-3, 7-5, 11-7, 13-11.
3. sor megnézi hol 2 a különbség. Ahol igaz, azok ikerprim párok.
4. sor a pár első tagját eltárolja, ahol 3 teljesül.
5. sor a második tagot az első+2 vel képzi és tárolja, ahol 4 teljesül.

Vegül reciprokokat képez, összeadja őket majd visszaadja az érteket.

A forrás további részében kirajzoltatjuk és láthatjuk, hogy egy felső határhoz konvergálnak az összegek.



2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

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

for (i in 1:kiserletek_szama) {

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

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

```
    }else{

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

    }

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

}

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

for (i in 1:kiserletek_szama) {

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

}

valtoztatesnyer = which(kiserlet==valtoztat)


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

A program véletlen eseteket generál. Meghatározzuk a kísérletek számat majd a játékos vektorban véletlenszerüen megadjuk mit választ a játékos esetenként. A vektorok mérete a kisérletek számával egyenlő.

Egy for ciklussal végigmegyünk a kisérletek száman és megnézzuk hogy eltalálta-e az eredményt.

Ha nem találta el, a műsorvezető kap pontot.

Ha nyer, kiértékeljük hogy hányszor nyert volna változtatás nélkül. Egyéb esetekben változtatással nyert volna.

Megszámoljuk a vektorokon az eseményeket majd kiiratjuk az eredményeket.

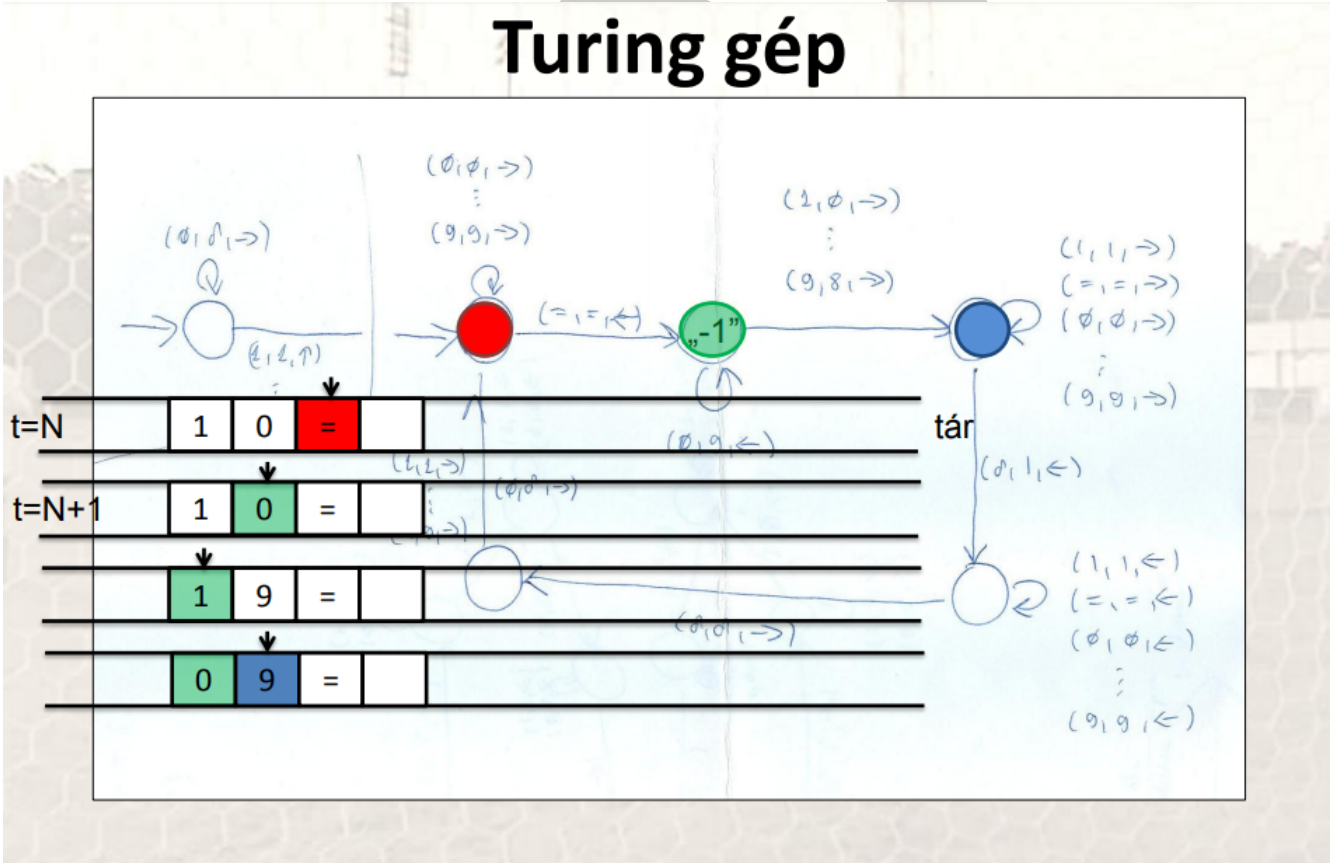
3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Kép forrása: Dr. Bátfai Norbert Prog. 1 előadása.



Unáris szám annyi 1-es karakterből áll, ahány az értéke. Megadunk egy int értéket majd létrehozunk egy üres kimeneti stringet. A for függvény annyiszor fut le ahány a bementi számunk.

```
        for (i;i=0;i--)
```

Minden esetben hozzáad(appendál) egy 1-es karaktert a stringhez. Végül kiiratjuk a stringet.

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

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

A Chomsky féle nyelvhierarhia az a számítógépes értelmezés alapja. E nélkül a program nem érti meg a kontextust és nem képes olyan eredményeket visszaadni amire várunk.

S->abc|A

A->aABc|abc

cB->Bc

bB->bb

_____

S→aSBC|aBC

CB→BC

aB→ab

bB→bb

bC→bc

cC→cc

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.

```
long long int
```

Új adattípus a C99 nyelvben. Explicit specifikáció adattípus komplex számokhoz. 32 bit méret, [-2,147,483,647, +2,147,483,647] közötti értékkel.

Aritmetikai és vezérlő utasítások függvényen belül, blokkban helyezkednek el. A tevékenység pontos megfogalmazása amit nem lehet részletezni, pontos jelentéssel bír.

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!

Felvesszük a megoldásunkat tároló intet 0-ként, hiszen ehhez adunk majd hozzá.

```
                    %{
    #include <stdio.h>
    int realnumbers = 0;
    %}
```

Definiáljuk mi is az amit keresünk.

```
    digit [0-9]
```

Valahányszor valós számot találunk, a (megoldás) darabszámot növeljük eggyel, és a kimeneti stringen a valós számot kiemeli egy "realnum=" -al.

```
                {digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
```

Lehívjuk az yylex-et. A szokásos módon kiírjuk az eredményt standard outputon.

```
    int
    main ()
    {
     yylex ();
     printf("The number of real numbers is %d\n", realnumbers);
     return 0;
    }
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

```
   %{
#include <stdio.h>
%}
leet  [e]
%%
{leet}  {printf("3");}
%%
int
main ()
{
 yylex ();
 return 0;
}
int yywrap(){return(1);}
```

A program valahányszor az "e" karaktert olvass a bemeneten, helyette egy "3"-as karaktert ír ki. Semmilyen más karakter nem változik. A main utáni rész megegyezik az előző felattal.

3.6. A források olvasása

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

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

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

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

i. Ha a SIGINT jel kezelését végrehajtjuk, akkor a jel kezelését a 'jelkezelo' végezze.

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

ii. Ötször lefutó for ciklus.

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

iii. Ugyan az mint az előző.

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

iv.

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

A 'tomb' elemeinek az index értékét adja meg.

v. Egy for ciklus ami addig fut le amíg eléri n-t és d indexértékét s indexértékéhez hasonlítja.

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

vi. Két számot váró kiírás két függvényből.

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

vii. Két számot váró kiírás egy fügvényből és egy változóból.

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

viii. Két számot váró kiírás egy nem nulla függvényből és egy változóból.

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

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

1. minden xre van egy annál nagyobb prímszán.
2. minden xre van egy annál nagyobb ikerprím számpár.
3. Van oylan y aminél minden x prím kisebb.
4. Van olyan y aminél bármely nagyobb szám prímszám.

3.8. Deklaráció

- egész

```
int a;
```

- egészre mutató mutató

```
int *b = &a;
```

- egész referenciája

```
int &r = a;
```

- egészek tömbje

```
int c[5];
```

- egészek tömbjének referenciája (nem az első elemé)

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

- egészre mutató mutatók tömbje

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

- egészre mutató mutatót visszaadó függvény

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

- egészre mutató mutatót visszaadó függvényre mutató mutató

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

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

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

- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

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

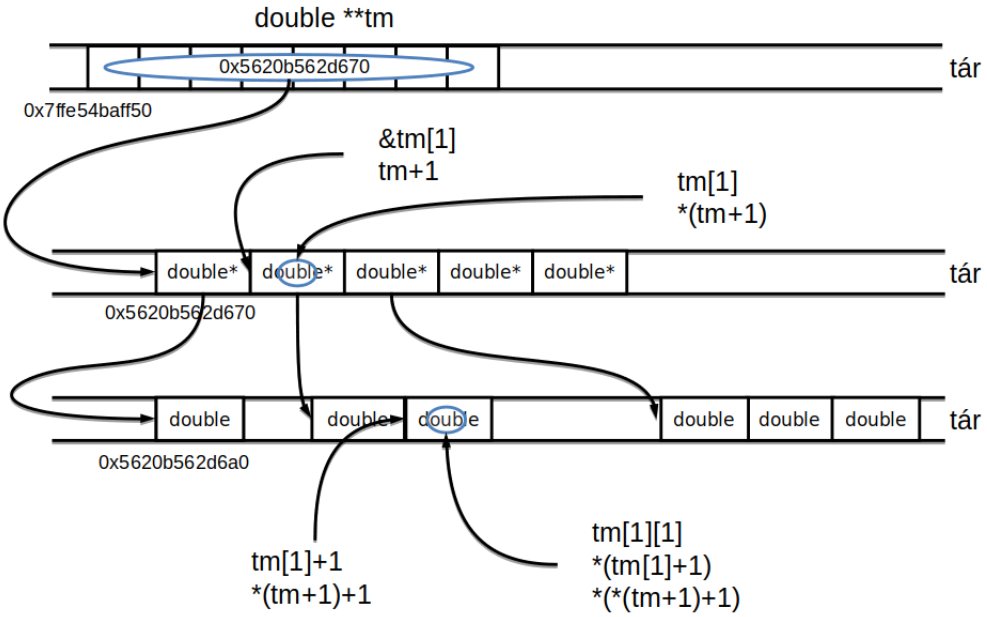
4. fejezet

Helló, Caesar!

4.1. int ** háromszögmátrix

A háromszög mátrix 2 tulajdonsággal rendelkezik, négyzetes, tehát sorai és oszlopai száma megegyezik, illetve, a főátlója alatt(vagy felett) csak nulla szerepel.

Forrás: Bátfai Norbert Repo Kép forrása: Norber Bátfai



Létrehozunk egy egész számot és lefoglaljuk a **tm területét egy mutatóra mutató doubleként. A `malloc` függvény lefglalja a dinamikus területen a paramétereként kapott méretű területet. Ey 8 bitet fog lefoglalni. A kövezekző sor ki fogja írni a tm cimét. A Maloc fügvény azt jelenti, hogy az operációs renszertől kér nemóriát ami jelenleg 8*5 azaz 40 bit. A következő sorban kí iratjuk a tm vátozó értékét. Létrehozunk egy for ciklust ami belselyében tm[i] lefoglalja a megfelelő helyet. Egy for ciklust ban lélvő for ciklussal tele rakjuk a mátrixunkat értékekel. Értéket rendelünk tm-hez. Itt: 42, 43, 44, 55 A free (tm) parancsal pedig felszabaitjuk a pointereket.

```
#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;
}
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás kódja itt.

```
&#xfeff;#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);
```

```
    }
}
```

Az exor titkosítás egy relatív egyszerű folyamat. Az exor kapu a lenti részben fut le.

```
            buffer[i] = buffer[i] ^ kulcs[kulcs_index];
```

A program beolvassa a kulcsot és a bemeni fájlt. Az utóbbit bufferen kezeli hogy a fájl mérete ne okozzon gondot.

A bufferen lévő bájtokat egyessével átvezetjük az exor kapun ami matematikai logikával dönti el hogy meváltozik-e a bit.

Mivel a kulcs több karakteres is lehet, ezt indexeljük és minden bitnél lépegetjük, amennyiben a végéhez érünk a kulcs indexelését az elejéről kezdjük

Valahányszor a buffer végére érünk, kiiratjuk a módosított adatot a kimentre majd a fájl végének elérésig ismételjük a bufferelési és titkosítási processt.

Ez a folyamat fordítva ugyanúgy működik, tehát ismétléskor ugyan az a jelszó megadásával az exor kapu a biteket az ereti állapotukra állítja vissza.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás kódja itt:

```
    public class Titkosito {

    public Titkosito(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 Titkosito(args[0], System.in, System.out);

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Az előző feldat java nyelven megírva. Bemeneti fájl helyett billentyűkombináció lenyomásáig várja az inputot a terminál.

4.4. C EXOR törő

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

A töréshez szükségünk lesz egy módszerre hogy eldöntsük mi is lehet a jó megoldás. Itt a szavak átlagos hosszát és a leggyakoribb magyar szavakat ellenőrizzük.

Az átlagos szóhosszt a szókozök számából és a fájl hosszából állapíthatjuk meg. Ezt egy funkcióként fogjuk használni.

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

  return (double) titkos_meret / sz;
}
```

Egy másik funkcióként lehívjuk ez előzőt majd végrehajtjuk az ellenőrzést. "and" kritériumokként hosszádjuk a leggyakoribb magyar szavak jelenlétét és a jó megoldást visszaadjuk.

```
int
tiszta_lehet (const char *titkos, int titkos_meret)
{
  double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}
```

Bevezetjük fonkcióként a standard exort.

```
            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;

    }

}
```

Bevezetjük az exor törés funkciót ami kezeli az átadott információkat és az alatta lévő funkciók meghívását.

Az exorhoz hasonlóan kezeljük a beolvasási buffert.

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

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

```
    p += olvasott_bajtok;
```

A jelszó töréshez felhasználandó karaktereket itt tároljuk.

```
    char chars[]={'k','*','/','a','g',',','-','%','\0'};
```

Egy mély for ciklus generálja az éppen kipróbált kulcsot.

```
for (int ii = 0; ii < 8; ++ii)
    for (int ji = 0; ji < 8; ++ji)
        for (int ki = 0; ki < 8; ++ki)
            for (int li = 0; li < 8; ++li)
                for (int mi = 0; mi < 8; ++mi)
                    for (int ni = 0; ni < 8; ++ni)
                        for (int oi = 0; oi < 8; ++oi)
                            for (int pi = 0; pi < 8; ++pi)
      {
        kulcs[0] = chars[ii];
        kulcs[1] = chars[ji];
        kulcs[2] = chars[ki];
        kulcs[3] = chars[li];
        kulcs[4] = chars[mi];
        kulcs[5] = chars[ni];
        kulcs[6] = chars[oi];
        kulcs[7] = chars[pi];
```

Meghívjuk az exor törés funkciót és amenyiben találunk megoldást, kiírjuk a megfelelő jelszót és a megoldást.

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

4.5. Neurális OR, AND és EXOR kapu

R

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

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

A neuron az elektromos jelek feldolgozásáért felelős agysejt. Az agy információfeldolgozó kapacitása a neuronok hálózatából alakult ki. Neurális hálónak nevezzük azt a párhuzamos működésre képes információfeldolgozó eszközt, amely nagyszámú, hasonló típusú elem összekapcsolt rendszeréből áll. Jellemzője az is, hogy rendelkezik tanulási algoritmussal és képes használni a megtanult információt.

A programhoz szükségünk van az R nyelvre és a neuralnet package-re.

```
$ more neuralis.r
library(neuralnet)

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

or.data <- data.frame(a1, a2, OR)

nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE,  ↩
   stepmax = 1e+07, threshold = 0.000001)

plot(nn.or)

compute(nn.or, or.data[,1:2])

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])
```

'a1' és 'a2' értékeket tartalmaz, az OR pedig a logikai 'VAGY' műveletet jelöli. A program előre meghatározott szabályok alapján tanul. A `compute` parancs segítségével tudjuk leellenőrizni, hogy a megfelelő eredményeket kaptuk-e. A logikai 'ÉS' művelet (and) ugyan így történik. Az 'EXOR' műveletnél csak többrétegű neuronokkal lehetséges a tanítás (hidden = 2).

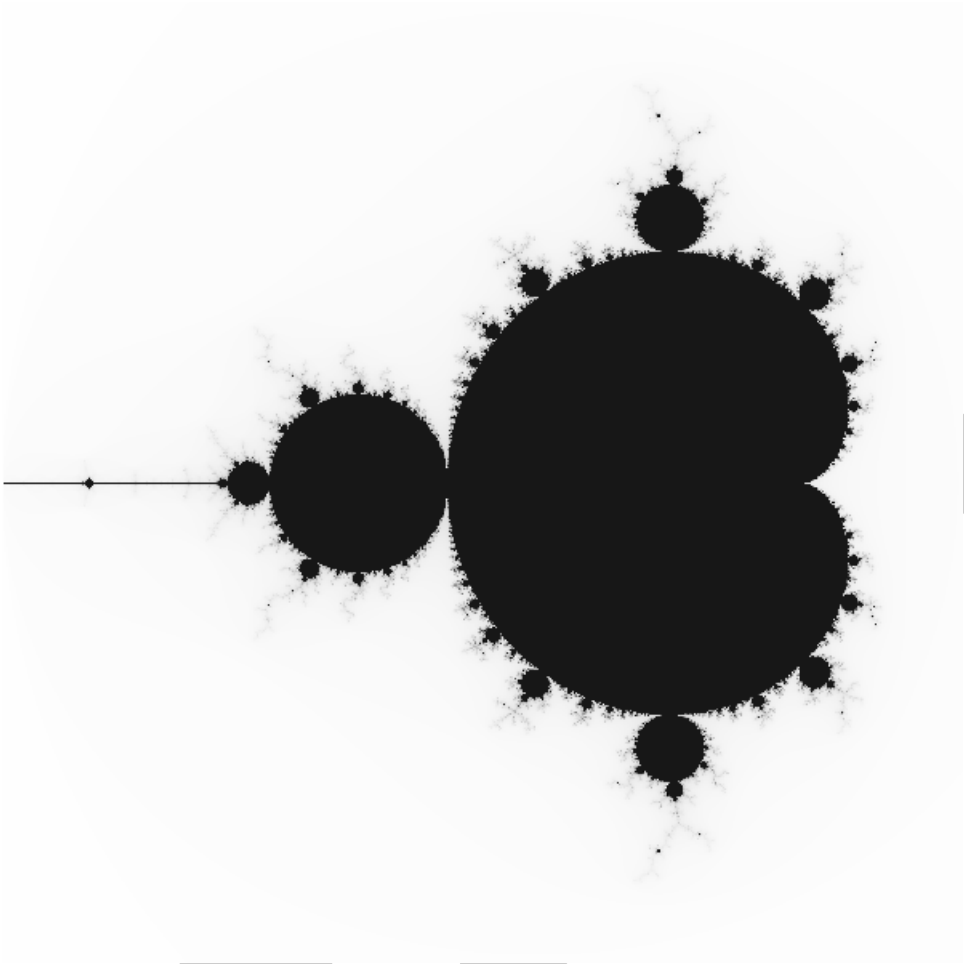
4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: https://gitlab.com/whoisZORZ/bhax/tree/master/attention_raising/Caesar/Perceptron

Perceptron a neurális hálózat hibakezelőjének része. A latin perceptioból syóból ered, észlelést jelent. A perceptron az észlelésre használt eszközök. Gépi tanulás a alakalmazási területe melynek célja hogy magától tanuljon ilyen gyakorlati hálókat a program. Perceprton egy algoritmus mely bináris osztályokat tanít a számítógépnek.

A feladatban neuron hálókról, azon belül perceptronokról beszélünk. Ez egy algoritmus amely a számítógépnek megtanítja a bináris osztályozást. A neuron egy bizonyos pontot elérve , ad egy kimenetet. A perceptronnak adhatunk példákat és képes lesz megmondani a többi elemről, hogy milzen tulajdonságú. Bináris osztáylozásnak nevezzük, mert bináris adatot adunk neki.



5. fejezet

Helló, Gutenberg!

5.1. A kivételkezelés alapfogalmai, kivételkezelési módszerek és eszközrendszerek 11. hét

Különböző szabályos és szabálytalan kilépéseket kezelhetünk amennyiben előre definiáljuk azokat.

5.2. A programnyelvek I/O eszközei, állománykezelés 9. óra

input állomány: a feldolgozás elıtt már léteznie kell, és a feldolgozás során változatlanmarad, csak olvasni lehet belőle.

output állomány: a feldolgozás elıtt nem létezik, a feldolgozás hozza létre, csak írni lehet bele.

input-output állomány: általában létezik a feldolgozás elıtt és létezik a feldolgozás után is, de a tartal ma megváltozik, olvasni és írni is lehet.

C:

Az I/O eszközrendszer nem része a nyelvnek. Standard könyvtári függvények állnak rendelkezésére. Létezik a bináris és a folyamatos módú átvitel, ez utóbbinál egy formátumos és egy szerkesztett átvitel keverékeként. Szeriális szerkezetet kezel fix és változó rekordformátummal. Az I/O függvények minimálisan egy karakter vagy karaktercsoport, illetve egy bájt vagy bájtcsoport írását és olvasását teszik lehetıvé.

5.3. Blokk. Hatáskörkezelés, láthatóság. Fordítási egység. Absztrakt adattípus. Generikus programozás 8. óra

A blokk olyan programegység, amely csak másik programegység belsejében helyezkedhet el, külsı szinten nem állhat.

Formálisan a blokknak van kezdete, törzse és vége. A kezdetet és a véget egy-egy speciális karaktersorozat vagy alapszó jelzi.

Blokkot aktivizálni vagy úgy lehet, hogy szekvenciálisan rákerül a vezérlés, vagy úgy, hogy GOTO-utasítással ráugrunk a kezdetére. Egy blokk befejezıdhet, ha elértük a végét, vagy GOTO-utasítással kilépünk belıle, vagy befejeztetjük az egész programot a blokkban.

A hatáskör a nevekhez kapcsolódó fogalom. Egy név hatásköre alatt értjük a program szövegének azon részét, ahol az adott név ugyanazt a programozási eszközt hivatkozza, tehát jelentése, felhasználási módja, jellemzıi azonosak. A hatáskör szinonimája a láthatóság.

5.4. Paraméterkiértékelés, paraméterátadás 7. óra

érték szerinti - aktuális param értéke a címre másolódik hívott, form. param van címkomponense

cím szerinti - cím másolódik hívott, form. param

eredmény szerinti - aktuális param címe másolódik, de nem használja csak a végén amikor a form param értékét a címre másolja hívott, form. param van címkomponense

érték-eredmény szerinti - aktutális param értéke (kezdőérték lesz a formálisnak) és címe másolódik, de a címet nem használja csak a végén amikor a form param értékét a címre másolja hívott, form. param van címkomponense

5.5. Programegységek. Hívási lánc, rekurzió 6. óra

Az eljárásorientált programnyelvekben a program szövege többé-kevésbé független, szuverén részekre, programegységekre tagolható.

Egy programegység bármikor meghívhat egy másik programegységet, az egy újabb programegységet, és így tovább. Így kialakul egy hívási lánc.

Azt a szituációt, amikor egy aktív alprogramot hívunk meg, rekurziónak nevezzük.

5.6. Utasítások 5. óra

A könyvben az utasítás típusairól, fajtájiról és alkalmazási módjukról olvashatunk. Értelmezzük az if-then, switch-case logikai használatát. Megismerjük a ciklusokat és azoknak feltételtípusait és kezelését. Ezekhez példákat látunk különböző nyelvekben.

5.7. Kifejezések 3. és 4. óra

Megismerjük a kfiejezések típusait, a C nyelvben a precedencia szabályait és megismerjük, majd értelmezzük az operátorok típusait, funkcióit. Kifejezések értelmezése: printf("%d\n", sizeof(char) * 8); printf – azonosító "%d\n"– karakterlánc 8 – állandó sizeof, char – kulcsszó

5.8. Alapfogalmak 2. óra

Alapfogalmak a programozásról, a gépi nyelvről.

– gépi nyelv – assembly szintő nyelv – magas szintő nyelv

Nyelvek típusai - Eljárásorientált nyelvek - Objektumorientált nyelvek

Assembly Gépi nyelvre való fordítás menete.

5.9. 1. óra

Bevezető példák, egyszerű C programok írása. Konstansok és változók használata, adatbevitel, adatok megjelenítése a standard kimeneten. utasítások, cilusok, tömbök.

6. fejezet

Helló, Mandelbrot!

6.1. A Mandelbrot halmaz

Forrás: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Mandelbrot/3.1.2.cpp

A Mandelbrot-halmaz elemei komplex számok, amiken a: x1:=c, c -> komplex szám xn+1:=(xn)^2 +c sorozat, rekurzív, korlátos.

Ennek a halmaznak elemeit ábrázolva a komplex számsíkon (mi esetünkben képfájlba) egy fraktálalakzatot kapunk.

A képlet megadására az alábbi függvényt is használjuk.

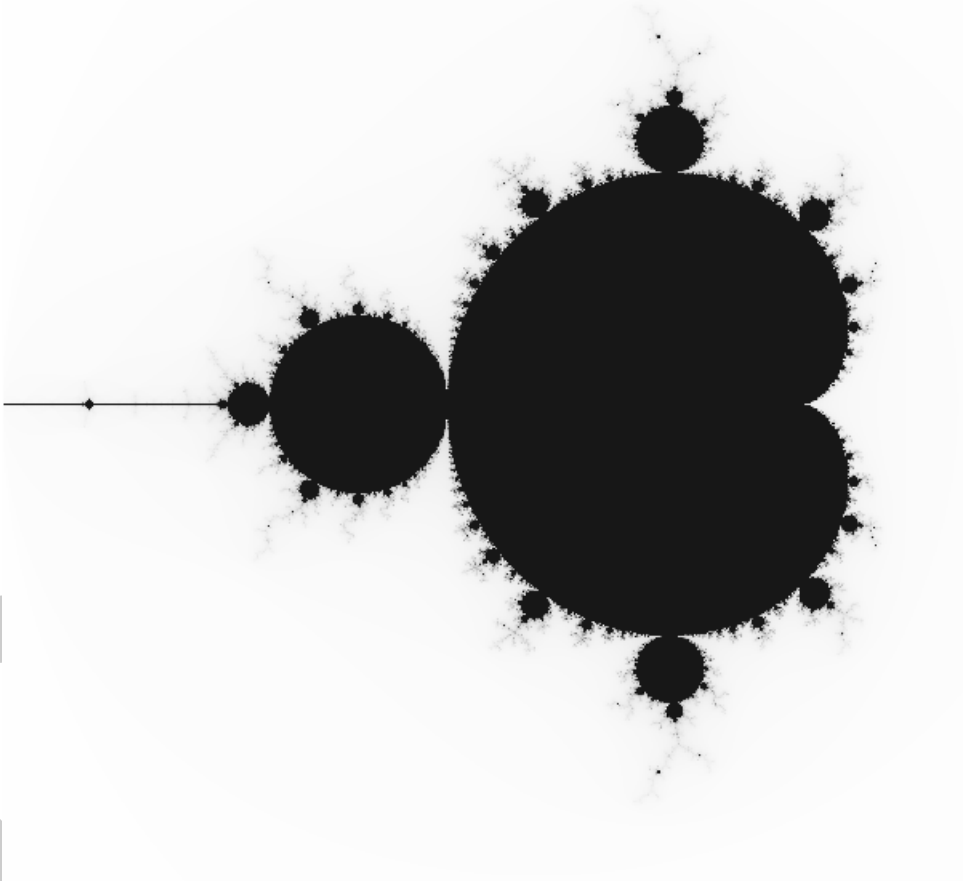
f(z) = z^2 + c

```
#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;

}
```

A mandel egy kétdimenziós tömböt kér be amit egy megadott mátrixba helyezünk bele. Minden pixelre kiszámoljuk hogy része-e a Mandelbrot halmaznak, addig fut a függvény amig el nem érjük az iterációs határt. Miután kiszámoltuk a kép adatait png fájlá alakitjuk.



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

```
#include <complex>
```

Meghivjuk a complex headert.

A mainben deklarálnunk kell a szükséges adatokat és ott kezeljük a fájlt hasonlóan az előző feladathoz. Az 'argc' az argumentumok számát, mig az 'argv' a tömb elemeit fogja tartalmazni. Az 'atoi' függvény a bemenetet egésszé, az 'atof' függvény pedig pointerré alakitja. Az std::complex fogja kezelni a nem valós számokat.

6.3. Biomorfok

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

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

Egy Mandelbrotos-halmazhoz hasonló halmaz, a Júlia-halmaz vizsgálata alapján eloállt algoritmus.

A 2. for-ban számoljuk ki a rácsponthoz a komplex értéket, létrehozzuk 0 kezdértékkel az `iteracio` változót, aztán indul még egy for ciklus, ami 0-tól megy iteraciosHatar -ig.

Itt alkalmazzuk a rekurzív sorozatunkat, majd vizsgáljuk, hogy az eredmény valós vagy képzetes része átlépte-e az `R`-ben megadott értéket, ha igen, akkor az iteracio -ba eltesszük az i-t, és kilépünk a ciklusból.

```
#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;

}
```

For ciklussal számolunk amely az iterációs határig fut.

6.4. A Mandelbrot halmaz CUDA megvalósítása

Forrás: GNU Public License Ben Hiller https://github.com/benhiller/cuda-mandelbrot

Az elező Mandelbrot feladatoknál a CPU-t használtuk fel a számításra, mig most a videókártyánkat fogjuk. A GPU rengeteg számitást végez párhuzamosan amely grafikai problémák megoldására sokkal hatékonyabb. Itt au nvidia CUDA technológiáját fogjuk felhasználni. Eheu kell a CUDA toolkit linuxunkra.

Létrehozunk egy megadott szélességü rácsot és mendegyiket egy külön CUDA maggal számoljuk ki párhuzamosan. A cudaMalloc függvény foglalja nekünk le a vramot GPU-nkból. A forrásfájlt az nvcc-vel tudjuk lefuttatni. Ez a folyamat másodpercek alatt lefut és ugyan azt az eredményt adja.

```
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>
#include <sys/times.h>
#include <iostream>
#define SIZE 600
#define ITERATION_LIMIT 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 width = SIZE, height = SIZE, iterationLimit = ITERATION_LIMIT;
// a számítás
float dx = (b - a) / width;
float dy = (d - c) / height;
float reC, imC, reZ, imZ, ujreZ, ujimZ;
// Hány iterációt csináltunk?
int iteration = 0;
// c = (reC, imC) a rács csomópontjainak
// megfelel˝o komplex szám
reC = a + k * dx;
imC = d - j * dy;
// z_0 = 0 = (reZ, imZ)
reZ = 0.0;
imZ = 0.0;
iteration = 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 && iteration < iterationLimit)
    {
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;
    ++iteration;
    }
return iteration;
}
/*
__global__ void
mandelkernel (int *buffer)
{
int j = blockIdx.x;
int k = blockIdx.y;
buffer[j + k * SIZE] = mandel (j, k);
}
*/
__global__ void
mandelkernel (int *buffer)
{
int tj = threadIdx.x;
int tk = threadIdx.y;
int j = blockIdx.x * 10 + tj;
int k = blockIdx.y * 10 + tk;
buffer[j + k * SIZE] = mandel (j, k);
}
void
cudamandel (int buffer[SIZE][SIZE])
{
int *deviceImageBuffer;
cudaMalloc ((void **) &deviceImageBuffer, SIZE * SIZE * sizeof (int));
// dim3 grid (SIZE, SIZE);
// mandelkernel <<< grid, 1 >>> (deviceImageBuffer);
dim3 grid (SIZE / 10, SIZE / 10);
dim3 tgrid (10, 10);
mandelkernel <<< grid, tgrid >>> (deviceImageBuffer);
cudaMemcpy (buffer, deviceImageBuffer,
SIZE * SIZE * sizeof (int), cudaMemcpyDeviceToHost);
cudaFree (deviceImageBuffer);
}
```

```
int
main (int argc, char *argv[])
{
// Mérünk id˝ot (PP 64)
clock_t delta = clock ();
// Mérünk id˝ot (PP 66)
struct tms tmsbuf1, tmsbuf2;
times (&tmsbuf1);
if (argc != 2)
    {
    std::cout << "Hasznalat: ./mandelpngc fajlnev";
    return -1;
    }
int buffer[SIZE][SIZE];
cudamandel (buffer);
png::image < png::rgb_pixel > image (SIZE, SIZE);
for (int j = 0; j < SIZE; ++j)
    {
    //sor = j;
    for (int k = 0; k < SIZE; ++k)
        {
        image.set_pixel (k, j,
        png::rgb_pixel (255 -
        (255 * buffer[j][k]) / ITERATION_LIMIT,
        255 -
        (255 * buffer[j][k]) / ITERATION_LIMIT,
        255 -
        (255 * buffer[j][k]) / ITERATION_LIMIT));
        }
    }
image.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;
}
```

6.5. 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 komplex számokat!

Forrás: GNU Public License Sully Chen https://github.com/SullyChen/Mandelbrot-Set-Plotter/blob/master/-MandelbrotSetPlotter.cpp

A program beolvassa a fájlt és képes azt az ablakban kezelni. Ay egér kattintására annak pozicióját lekéri és mozgatja az ablakot.

```
                #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++)
        {
            //scale the pixel location to the complex plane for ↩
               calculations
            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; //keep track of the number of iterations
            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;
            }
            //color pixel based on the number of iterations
            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();
        }

        //zoom into area that is left clicked
        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;
}
```

7. fejezet

Helló, Welch!

7.1. 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 forrás: https://github.com/raddanfea/prog1/blob/master/polt

Létrehozzuk a random osztályt, hoyyá konstuktort és destrutort. Felvesszük privát és public értékeinket és fügvényeinket. Két számot generálunk, egyiket tároljuk valueben, másikat a get() függvény visszaadja. Végén 10 random számot generálunk.

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>



class Random {

  public:
    Random();
    ~Random(){}
    double get();

  private:
    bool exist;
    double value;
};


Random::Random() {
  exist = false;
  std::srand (std::time(NULL));
```

```
};


double Random::get() {
  if (!exist)
  {
  double u1, u2, v1, v2, w;

    do{
      u1 = std::rand () / (RAND_MAX + 10.0);
      u2 = std::rand () / (RAND_MAX + 1.0);
      v1 = 2 * u1 - 1;
      v2 = 2 * u2 - 1;
      w = v1 * v1 + v2 * v2;
        }
    while (w > 1);

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

    value = r * v2;
    exist = !exist;
    return r * v1;
  }
  else
  {
    return value;
    exist = !exist;
  }
};
int main()
{
  Random rnd;

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

}
```

Az objektumorientált programozás (OOP) olyan módszert nyújt a programozók számára, amely lehetővé teszi a programok bonyolultságának csökkentését, a megbízhatóság és a hatékonyság növelését. Objektumokból, a valós világ elemeinek programozási modelljeiből építi fel a programot.

Osztályokatat definiálhatjuk változókkal és funkciókkal amelyet konstruktorként meghivhatunk létrehozva egy példányt az osztály mintájára. Megszabhatjuk hogy honna érhető el egy funkció vagy változó az osztályban. Lehet protected, private vagy public.

7.2. LZW

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

Megoldás forrás: https://github.com/raddanfea/prog1/blob/master/lzw_v9

Az LZW (Lempel-Ziv-Welch) tömörítési algoritmust Terry Welch amerikai informatikus publikálta 1984-ben. A kiírásra kerülô kódok általában 12 bitesek, így a sztringtáblába 4096 bejegyzés fér. Az elsô 256 bejegyzésbe elôzôleg az egyszerû ASCII karakterek, 256-tól 4095-ig pedig a tömörítés során a byte-csoportok kerülnek.

Szükség lesz egy rekordra, amely a fának az elemeit tartalmazza. Lesznek bal oldali gyermeikei, amelyekben 0-k fognak szerepelni, illetve lesznek jobb oldaliak, azokban pedig 1-esek. Objektum-orientált programozási nyelvekben erre a célra classt használunk. A C nyelvben erre a célra 'struktúrákat' használunk.

Szükség lesz egy olyan függvényre, amely képes új elemek létrehozására, a gyökér, és az egyes és nullás elemek számára.

Az 'uj_elem()' függvény által vissza adott mutató által címzett helyre kerül ezáltal, értéke a speciális gyökér érték lesz. Ezt követően beállítjuk 'nullpointer'-re a gyökér által mutatott egyes, valamint nullás gyermeket. Végül a fa mutató a gyökérre fog mutatni.

7.3. Fabejárás

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

Megoldás forrás: https://github.com/raddanfea/prog1/blob/master/lzw_v9

Preorder, inorder, és posztorder bejárások léteznek attól függően, hogy a gyökér-elem kiírása hol történik meg.

A preorder: először gyökér, majd egyes aztán nullás elemek.

A posztorder: először egyes, majd nullás, aztán gyökér.

7.4. 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 forrás: https://github.com/raddanfea/prog1/blob/master/lzw_v9

A kiFile << binFa azt jelenti tagfüggvényként, hogy a kiFile valamilyen std::ostream stream osztály forrásába kellene beleírni ezt a tagfüggvényt, amely ismeri a mi LZW binfánkat (kiFile.operator<<(binFa)). Globális függvényként pedig: operator<<(kiFile, binFa). A bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen írjuk.

7.5. Mutató a gyökér

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

Megoldás forrás: https://github.com/raddanfea/prog1/blob/master/lzw_v9

A famutatót ráállítjuk a gyökérre a konstruktorban, amit alul hozunk létre. A gyökér is mutató lesz, ezért az előző forrásban mindenhol, ahol referenciaként adtuk át a gyökeret a famutatónak, referencia nélkül tesszük. A konstruktorban a gyökének helyet foglalunk a memóriában és erre állítjuk a fát rá. A szabadításkor pont helyett nyilat használunk, ha mutató mutatóit kérjük.

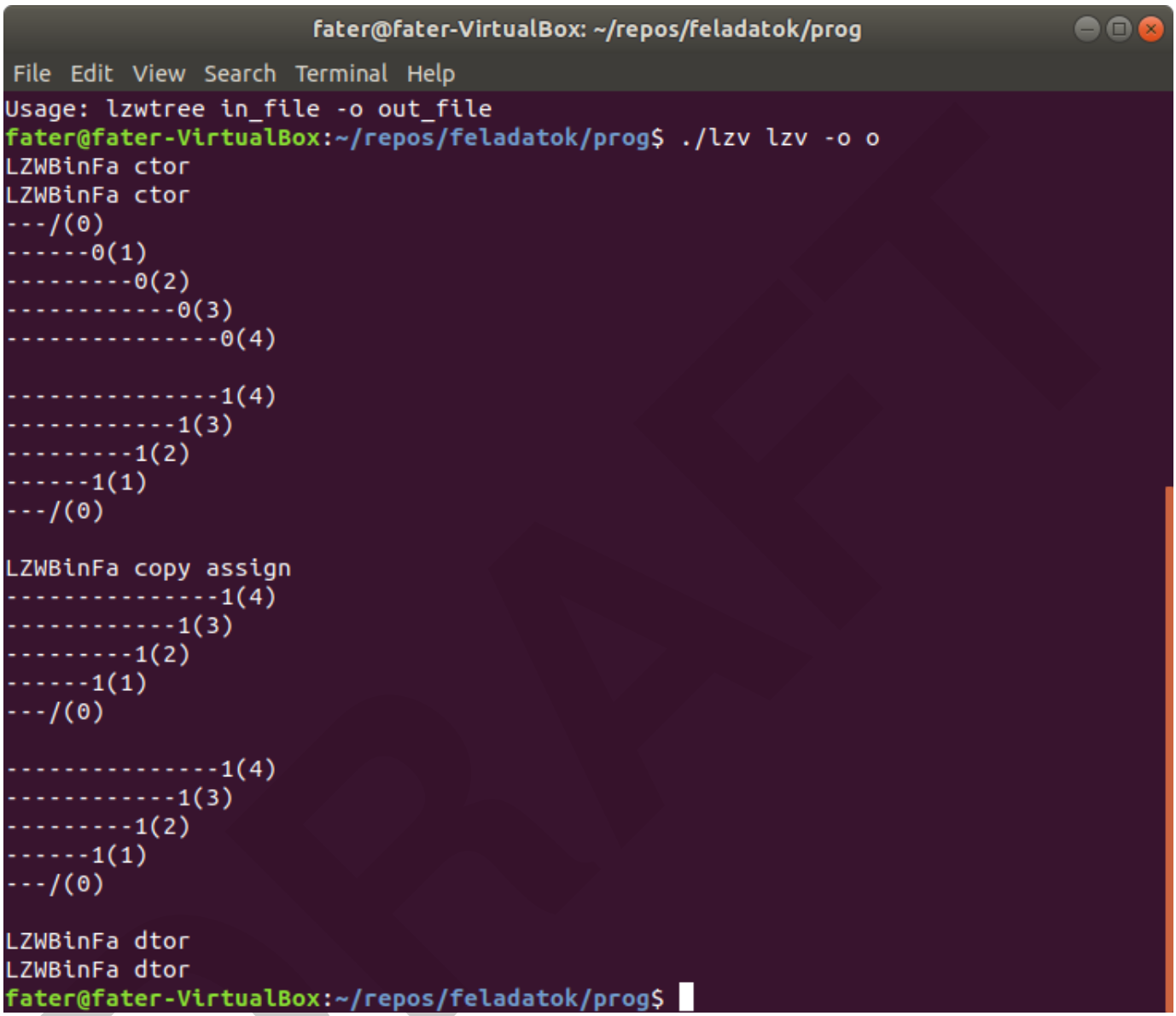
7.6. Mozgató szemantika

Megoldás forrás: https://github.com/raddanfea/prog1/blob/master/lzw_v9

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

A paraméterként átadott fa gyökerének az elemeit átadjuk az üres fának, majd a mozgatott fa elemeit kinullázzuk. Ezután az `std::move` függvénnyel átmozgatjuk az egyes elemeket, és végül kiírjuk az új fát. Az eredeti binFát már nem tudnánk kinyomni, mert azt nulláztuk.

7.7. LZW feladatkör kimeneti kép



8. fejezet

Helló, Conway!

8.1. 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

A forrásban megadott több fájlból áll. egy .pro összefoglaló fájl ami megmutatja nekünk hogy 3 forrásunk és 3 header fájlunk van.

ant.h:

```
#ifndef ANT_H
#define ANT_H

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;

#endif
```

Definiáljuk a hangyát mint objektum és adunk neki poziciót és irányt véletlenszerüen. typedef-el megadjuk hogy az objektum vektora az "ants" legyen.

`antthread.h`

Az ant.h és qtread felhasználásával létrehozzuk anz Anttread osztályt. Egy osztály egy threadnek felel meg.

Létrehozzuk a finish, pause és az isrunning funkciót a kezeléséhez.

Deklaráljuk a `timeDevel`, `newDir`, `detDirs`, `moveAnts`, `sumNbhs`, `setPheromones` funkciókat.

`antwin.h`

Megadjuk az AntWin osztályt, ami a QMainWindow gyermeke. Deklaráljuk a konstruktort, destruktort, és a `paintEvent` függvényt.

Meghatrozzuk a `closeEvent` és `keyPressEvent` függvényeket. Az ablak bezárásakor leállítja a szálat, a Q(kilépés) és P(szünet) valamint az Esc(kilépés) gombok lenyomását kezelik.

`antthread.cpp függvényei:`

Konstruktor:

Értéket ad a változóknak, és elhelyezi a hangyákat a rácson.

sumNbhs() Megszámolja a cella szomszédait.

newDir() Ha új irányt kell adni egy hangyának akkor azt ez a függvény kezeli.

detDirs() A hangya iránya alapján megadja, hogy merre megy.

moveAnts() Ez a függvény mozgat egy adott hangyát.

timeDevel() Az összes hangyát mozgatja.

setPheromone() Beállítja a feromonokat az adott cellán. `timeDevel()` használja.

run() A foylamatos loop itt valósul meg. Itt lesz a szünet és a megállás is kezelve.

Destruktor: Elpusztítja a hangyákat és a tömböt amiben tároltuk őket.

`antwin.cpp` függvényei: Kezeli az ablakot és a rajzoló threadet.

8.2. Java életjáték

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

```
public Sejtautomata(int szélesség, int magasság) {
        this.szélesség = szélesség;
        this.magasság = magasság;
        rácsok[0] = new boolean[magasság][szélesség];
        rácsok[1] = new boolean[magasság][szélesség];
        rácsIndex = 0;
        rács = rácsok[rácsIndex];
        for(int i=0; i<rács.length; ++i)
            for(int j=0; j<rács[0].length; ++j)
                rács[i][j] = HALOTT;
        siklóKilövő(rács, 5, 60);
```

```
                addWindowListener(new java.awt.event.WindowAdapter() {
                    public void windowClosing(java.awt.event. ↩
                       WindowEvent e) {
                        setVisible(false);
                        System.exit(0);
                }
            });
```

Létre kell hoznunk egy KeyListenert, amely a billentyűzetről kapott leütéseket figyeli és kezeli le és kezeli a funkciókat.

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

8.3. Qt C++ életjáték

Most Qt C++-ban!

```
SejtAblak::SejtAblak(int szelesseg, int magassag, QWidget *parent)
    : QMainWindow(parent)
{
    setWindowTitle("A John Horton Conway-féle életjáték");

    this->magassag = magassag;
    this->szelesseg = szelesseg;

    cellaSzelesseg = 6;
    cellaMagassag = 6;

    setFixedSize(QSize(szelesseg*cellaSzelesseg, magassag*cellaMagassag));

    racsok = new bool**[2];
    racsok[0] = new bool*[magassag];
    for(int i=0; i<magassag; ++i)
        racsok[0][i] = new bool [szelesseg];
    racsok[1] = new bool*[magassag];
    for(int i=0; i<magassag; ++i)
        racsok[1][i] = new bool [szelesseg];

    racsIndex = 0;
    racs = racsok[racsIndex];

    for(int i=0; i<magassag; ++i)
        for(int j=0; j<szelesseg; ++j)
            racs[i][j] = HALOTT;
```

Létrehozzuk az ablakot méretre, legenráljuk ohzzá a mátrixot és kezeljük a sejteket melyek alapból halottak.

```
    siklo(racs, 2, 2);
    sikloKilovo(racs, 5, 60);


    eletjatek = new SejtSzal(racsok, szelesseg, magassag, 120, this);
    eletjatek->start();

}
```

Meghatározunk pár élő sejtet és elindutjuk a szimulációt.

Végül létrehozzuk a paintEvent() függvényt, mely a rácsot és a sejteket rajzolja ki. Ha az adott sejt élő akkor feketével, ha halott akkor fehérrel jelöljük.

```
        void SejtAblak::paintEvent(QPaintEvent*) {
            QPainter qpainter(this);


            bool **racs = racsok[racsIndex];

            for(int i=0; i<magassag; ++i) { // végig lépked a sorokon
```

```
                for(int j=0; j<szelesseg; ++j) { // s az oszlopok
                    // Sejt cella kirajzolása
                    if(racs[i][j] == ELO)
                        qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag ↩
                            ,
                                          cellaSzelesseg, cellaMagassag, Qt ↩
                                             ::black);
                    else
                        qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag ↩
                            ,
                                          cellaSzelesseg, cellaMagassag, Qt ↩
                                             ::white);
                    qpainter.setPen(QPen(Qt::gray, 1));

                    qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
                                      cellaSzelesseg, cellaMagassag);
                }
            }

            qpainter.end();
        }
```

8.4. BrainB Benchmark

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

Az ablakban megjelenik egy objektum (Samu Entropy) és rajta kell tartani az egeret, míg az mozog. Ennek nehezítésére újabb objektumok is megjelennek az ablakban. A mozgatást x és y mentén is úgy oldjuk meg, hogy az előző értékhez hozzáadunk egy számot, mely egy random és az objektumra jellemző "agility" szorzata. Így érjük el a kiszámíthatatlan mozgást, ami az objektum mozgékonyságától is függ. Továbbá megszorítást is kell kötni, miszerint ha nem érjük el az ablak szélét, csak abban az esetben alkalmazzuk a mozgást.

```
    void move ( int maxx, int maxy, int env ) {

        int newx = x+ ( ( ( double ) agility*1.0 ) * ( double ) ( std::rand ↩
            () / ( RAND_MAX+1.0 ) )-agility/2 ) ;
        if ( newx-env > 0 && newx+env < maxx ) {
            x = newx;
        }
        int newy = y+ ( ( ( double ) agility*1.0 ) * ( double ) ( std::rand ↩
            () / ( RAND_MAX+1.0 ) )-agility/2 );
        if ( newy-env > 0 && newy+env < maxy ) {
            y = newy;
        }

    }
```

```
};
```

Ez a BrainBThread osztály. Ez a benchmrkért felelős.

```
void BrainBThread::run()
{
        while ( time < endTime ) {

                QThread::msleep ( delay );
                if ( !paused ) {
                        ++time;
                        devel();
                }
                draw();
        }
        emit endAndStats ( endTime );
}
void BrainBThread::pause()
{
        paused = !paused;
        if ( paused ) {
                ++nofPaused;
        }
}

void BrainBThread::set_paused ( bool p )
{
        if ( !paused && p ) {
                ++nofPaused;
        }
        paused = p;
}
```

9. fejezet

Helló, Schwarzenegger!

9.1. Szoftmax Py MNIST

Forrás: "Dr. Búzáné, Dr. Kis Piroska"

A TensorFlow egy szoftverkönyvtár gépi tanulási algoritmusok leírására és végrehajtására. Flexibilis, széles körű algoritmusok megvalósítására alkalmas, például a beszédfelismerésben, a robotikában, az információ kinyerésben, a számítógépek elleni támadások felderítésében és az agykutatásban.

A TensorFlow-t a Python programozási nyelv pip package managerével telepítjük. Első lépésként a Python fejlesztői környezete:

```
$ sudo apt update
$ sudo apt install python3-dev python3-pip
$ sudo apt-get install python3-matplotlib
$ sudo pip3 install -U virtualenv
```

A Python virtuális környezetének létrehozása és a TensorFlow pip package telepítése:

```
$ virtualenv --system-site-packages -p python3
$ source ./venv/bin/activate
$ pip install --upgrade pip
$ pip install --upgrade tensorflow
$ python -c "import tensorflow as tf; tf.enable_eager_execution ↩
   (); print(tf.reduce_sum(tf.random_normal([1000, 1000])))"
$ deactivate
```

Az `mnist_softmax_UDPROG61.py` fájl segítségével tehát a 28x28 pixeles png képeken lévő, kézzel rajzolt számjegyek felismerésére tanítjuk a gépi modellt. A használat előtt importálnunk kell a TensorFlow-t, illetve a korábban már említett MatPlotLib könyvtárat. A modell létrehozásához definiáljuk x és y változókat. A modellünkhöz szükség lesz súlyokra és bias értékekre is (W és b változókban tároljuk ezeket).

A keresztentrópia implementálásához szükségünk van egy y_ placeholderre is a helyes válasz bevitelére. A modellünket interaktív session-be rakjuk.

```
import argparse

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

```
import tensorflow as tf

import matplotlib.pyplot

FLAGS = None

def readimg():
    file = tf.read_file("sajat8a.png")
    img = tf.image.decode_png(file)
    return img

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

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

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

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

  sess = tf.InteractiveSession()

  tf.initialize_all_variables().run()
```

Kiiratjuk a pontosság értékét amelyet megadja a programunk jelenlegi pontosságát arányban. Kiiratjuk a tesztképet hogy mi is ellenőrizhessük.

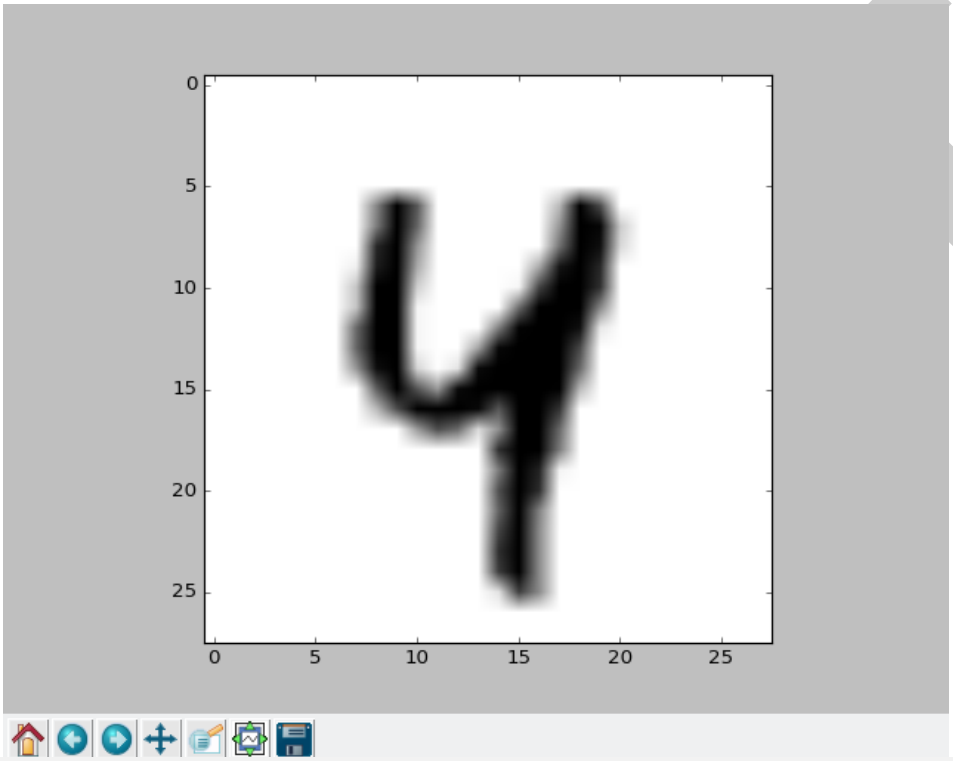
```
print("-- Learning")
  for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
      print(i/10, "%")
  print("------------------------------------------------------------")

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

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

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

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

  print("-- Identified number: ", classification[0])
  print("----------------------------------------------------------")

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

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

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

  print("-- Identified number: ", classification[0])
  print("----------------------------------------------------------")

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

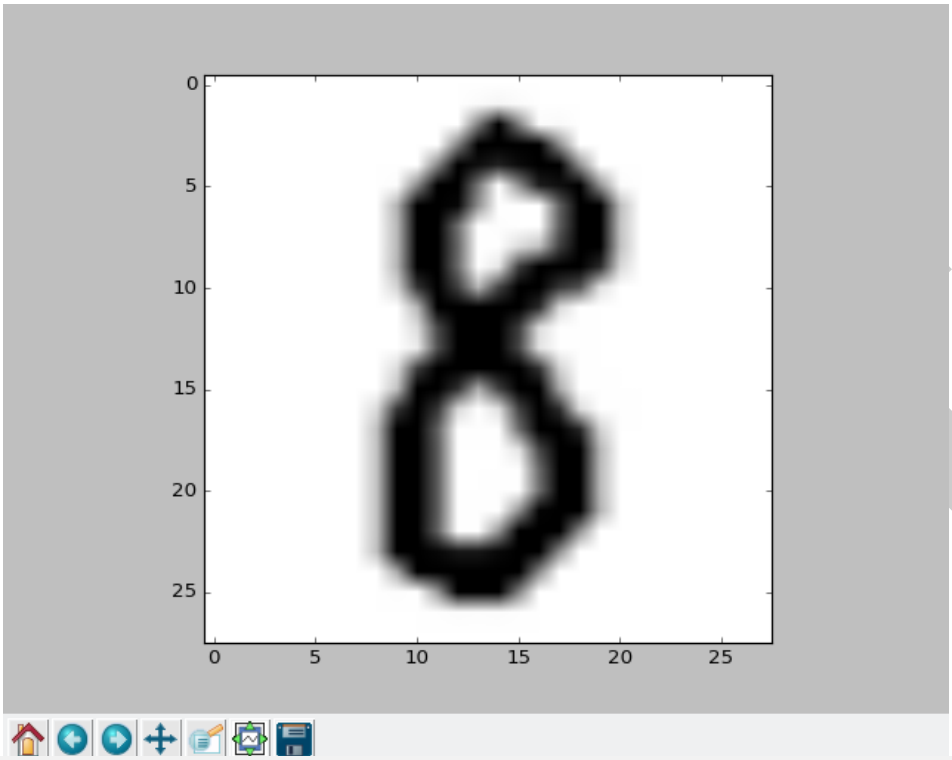
Igy fog kinézni a program helyes futtatása:

```
$ python mnist_softmax_UDPROG61.py
Extracting /tmp/tensorflow/mnist/input_data/train-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/train-labels-idx1-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-images-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-labels-idx1-ubyte.gz
-- Learning
0.0 %
10.0 %
20.0 %
30.0 %
40.0 %
50.0 %
60.0 %
70.0 %
```

```
80.0 %
90.0 %
------------------------------------------------------------
-- Testing
-- Accuracy:  0.9175
------------------------------------------------------------
```




```
 -- Identified number:  4
------------------------------------------------------------
```




```
-- Identified number:  8
```

9.2. Mély MNIST

Ezt a feladatot eldobtam.

9.3. Minecraft MALMÖ

Forrás: "Microsoft Project Malmo"

A Project Malmo egy olyan platform, amely segítségével kutatásokat végezhetünk a mesterséges intelligencia világában, méghozzá a népszerű játék, a Minecraft használatával. Python ágenst készítünk, amely egy alapértelmezett biomban gazdag világban az akadályokat legyőzve, jobbra-balra tartva felfedezi a világot.

A my_mission = MalmoPython.MissionSpec() parancs létrehoz egy XML stringet, amelyet a Minecraft képes értelmezni, így el is tud indulni a küldetésünk. A missionXML részben röviden összegezhetjük a mission-t, időlimitet állíthatunk (ms-ben megadva), illetve játékmódot is itt tudunk beállítani.

Utasítanunk kell az ágenst arra, hogy haladjon előre teljes sebességgel. Továbbá deklarálnunk kell néhány változót: a pozíció koordinátákat a stevex, stevey és stevez változók, az irányt a steveyaw változó, a nézetet pedig a stevepitch változó fogja tárolni.

```
        # Loop until mission starts:
        print("Waiting for the mission to start ", end=' ')
        world_state = agent_host.getWorldState()
        while not world_state.has_mission_begun:
            print(".", end="")
```

```
            time.sleep(0.1)
            world_state = agent_host.getWorldState()
            for error in world_state.errors:
                print("Error:",error.text)

        print()
        print("Mission running ", end=' ')

            agent_host.sendCommand( "move 1" )

        stevex = 0
        stevez = 0
        stevey = 0
        steveyaw = 0
        stevepitch = 0
        elotteidx = 0
        elotteidxj = 0
        elotteidxb = 0
        akadaly = 0
```

A yaw értéke a Minecraft koordinátarendszeréhez igazodik: észak 180, dél 0, nyugat 90, kelet pedig -90. Ezt hivatott szemléltetni a következő ábra is:

Miközben Steve mozog, blokkok veszik körül, tehát akadályokkal kell szembenéznie. Hogy ezt elkerüljük, a json függvény segítségével információt szerzünk az őt körülvevő blokkokról, ezeknek típusát ki is tudjuk iratni a terminálra. 3x3-as, blokkokból álló griddel dolgozunk.

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

  if "Yaw" in observations:
    steveyaw = observations["Yaw"]
  if "Pitch" in observations:
    stevepitch = observations["Pitch"]
  if "XPos" in observations:
    stevex = observations["XPos"]
  if "ZPos" in observations:
    stevez = observations["ZPos"]
  if "YPos" in observations:
    stevey = observations["YPos"]

  print ("Pozicio koordinatak: ", stevex, stevez, stevey)
  print ("Irany: ", steveyaw)
  print ("Nezet: ", stevepitch)

  if steveyaw >= 180-22.5 and steveyaw <= 180+22.5 :
    elotteidx = 1
    elotteidxj = 2
```

```
    elotteidxb = 0

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

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

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

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

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

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

  if steveyaw >= 90+22.5 and steveyaw <= 180-22.5 :
    elotteidx = 0
    elotteidxj = 1
    elotteidxb = 3

  print ("racsindex", elotteidx)
```

Ha Steve számára épp nem szabad az út akkor reagáljon és megnöveljük az akadályokat számláló változó értékét eggyel. Ha szabad az út, nem kell reagálnia, csak előre haladnia teljes sebességgel. Az akadályokat ugrálással is kikerülheti, így tud kiugrani a vízből vagy a lyukakból.

```
    if nbr[elotteidx+9]!="air" or nbr[elotteidxj+9]!="air" or nbr[ ↩
       elotteidxb+9]!="air":
                    print ("Nincs szabad utam, elottem: ", nbr[elotteidx ↩
                       +9])
                    agent_host.sendCommand ("turn" + str(turn))
                    akadaly = akadaly + 1
            else:
```

```
                    print ("Szabad az ut!")
                    agent_host.sendCommand ("turn 0")
                    agent_host.sendCommand ("jump 0")
                    agent_host.sendCommand ("attack 0")
                    akadaly = 0

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

            lepes = lepes + 1
            if lepes > 100:
            lepes = 0
            if tav < 20:
                    prevstevex = stevex
                    prevstevez = stevez
                    prevstevey = stevey
                    turn = turn * -1
                    agent_host.sendCommand ("attack 1")

            time.sleep(1)

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

10. fejezet

Helló, Chaitin!

10.1. Iteratív és rekurzív faktoriális Lisp-ben

Iteratív:

```
(define (fact n)
(do ((i 1 (+ 1 i))
(number 1 (* i number)))
((> i n) number
```

Deklaráljuk 'i'-t 1 értékkel. Deklaráljuk 'number'-t 1 értékkel majd szorozzuk 'i'-vel. Ezt egy loopba helyezzük és fordulásonként növeljük 'i'-t 1-el.

Rekurzív:

```
(define (fakt n ) (if (< n 1) 1 (* n (fakt (- n 1)))))
```

Itt először definiálunk, majd amig n nagyobb mint 1 szorozzuk (n-1)-el amely minden lépésnél csökkenti az n-t.

10.2. 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 kóddal intiutivan elnevezett operátorokkal kezelhetjük a gimp programot.

Példák:

```
        set! //gyorsparancs beállitásra
```

```
        text-width x // szélesség beállitása
```

```
        layer2 //kép layer váltása
```

```
        gimp-drawable-fill layer  FILL-FOREGROUND // layer kitöltése
```

És sok más ezken kívül. https://www.gimp.org/tutorials/Basic_Scheme/

Ezen kívül Lisp-es funkciókat is dekralálhatunk.

```
; bhax_chrome3.scm
;
; BHAX-Chrome creates a chrome effect on a given text.
; Copyright (C) 2019
; Norbert Bátfai, batfai.norbert@inf.unideb.hu
; Nándor Bátfai, batfai.nandi@gmail.com
;
;    This program is free software: you can redistribute it and/or modify
;    it under the terms of the GNU General Public License as published by
;    the Free Software Foundation, either version 3 of the License, or
;    (at your option) any later version.
;
;    This program is distributed in the hope that it will be useful,
;    but WITHOUT ANY WARRANTY; without even the implied warranty of
;    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
;    GNU General Public License for more details.
;
;    You should have received a copy of the GNU General Public License
;    along with this program.  If not, see <https://www.gnu.org/licenses/>.
;
; Version history
;
;  This Scheme code is partially based on the Gimp tutorial
;  http://penguinpetes.com/b2evo/index.php?p=351
;  (the interactive steps of this tutorial are written in Scheme)
;
;  https://bhaxor.blog.hu/2019/01/10/ ↩
   a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv
;

(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"
)
```

10.3. 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

```
        (image (car (gimp-image-new width height 0)))
```

A képhez hasonló módon réteget hozunk létre majd aztt hozzá kell adnunk a képhez és beállítjuk a hátteret zöldre. A szöveget elforgatjuk hogy mandala legyen belőle.

```
        (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100  ↩
           LAYER-MODE-NORMAL-LEGACY)))
        (gimp-image-insert-layer image layer 0 0)
        (gimp-context-set-foreground '(0 255 0))
        (gimp-drawable-fill layer FILL-FOREGROUND)
```

```
(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"
```

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

11.1.

IV. rész

Irodalomjegyzék

11.2. Általános

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

11.3. C

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

11.4. C++

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

11.5. 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.