Univerzális programozás

Így neveld a programozód!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert Ã©s Turi, Gergő Marcell	2019. szeptember 19.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai

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

Ábrák jegyzéke

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?

Egy eszköz a programozó kezében, ugyan olyan eszköz, mint festő kezében az ecset, vagy író kezében a toll.

1.2. A Könyv adatai

Passzolások száma: 4 Passzolt feladatok: 2.2 - Lefagyott, nem fagyott akkor most mi van? 4.6 - Hibavisszaterjesztéses perceptron 8.2 - Mély MNIST 8.3 - Minecraft - MALMÖ Tutorált emberek száma: 2

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: https://youtu.be/lvmi6tyz-nI

Több módon is létre lehet hozni végtelen ciklusokat. Ahoz hogy egy cput-t száz százalékban dolgoztassunk egyszerűen folyamatosan dolgoztatjuk egy processzen. Így egy egyszerű végtelen ciklussal el is érhetjük hogy az egyik mag száz százalékban legyen dolgoztatva. Forrás: vegtelen_egy.c

```
//vegtelen_egy.c

int main()
{
  while(1)

    {

    }

  return 0;
}
```

```
ps aux | grep vegtelen
wyndrel+  4190 99.0  0.0   4376   716 pts/1    R+   13:19    ↩
   0:20 ./vegtelen_egy
wyndrel+  4197  0.0  0.0  24120  1004 pts/2    S+   13:19    ↩
   0:00 grep --color=auto vegtelen
```

Láthatjuk a harmadik oszlopban hogy a processz közel száz százalékban terheli a cpu-t.

Ahhoz hogy egy mag nulla százalékban, pontosabban közel nulla százalékban legyen dolgoztatva, ugyancsak egy egyszerű végtelen ciklust kell alkotnunk amiben van szünet, így bár nem teljesen nulla százalékban van dolgoztatva, olyan kis mértékben használjuk a mag erőforrásait hogy tekinthetjük nulla százaléknak. Forrás: vegtelen_nulla.c

```
              //vegtelen_nulla.c

int main()
{

  while(1)
  {
    sleep(1);
  }

  return 0;
}
```

```
            ps aux | grep vegtelen
            wyndrel+  4261  0.0  0.0   4376   796 pts/2    S+   13:22    ↩
               0:00 ./vegtelen_nulla
            wyndrel+  4266  0.0  0.0  24120  1148 pts/1    S+   13:23    ↩
               0:00 grep --color=auto vegtelen
```

Itt pedig láthatjuk hogy a processz nulla százalékban terheli a cput.

A legegyszerűbb megoldás arra hogy minden processz száz százalékban terheljünk az az, hogy az első, tehát a 'vegtelen_egy' processzet annyiszor futtatjuk ahány magunk van.

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

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

Megoldás videó:

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

```
Program T100
{

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

```
      else
       return false;
  }

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tanulságok, tapasztalatok, magyarázat...

SKIP - 1

2.3. Változók értékének felcserélése

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

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

valtozo_csere.c

```
            int main()

{
  printf("Első Szám ->  ");
  int a;
  scanf("%d", &a);
  printf("\nMásodik Szám ->  ");
  int b;
  scanf("%d", &b);
  printf("\n-------------\n");

  printf("Első Szám ->  %d\n", a);
  printf("Második Szám ->  %d\n", b);
  printf("\n----- Csere -----\n");
  a = a + b;
  b = a - b;
  a = a - b;


  printf("Első Szám ->  %d\n", a);
  printf("Második Szám ->  %d\n", b);


  return 0;


}
```

Fontos tudni, hogy bár ezzel a megoldással megspórolunk egy változót, a mai számítógépek gyorsasága, illetve memóriakapacitására tekintettel nézve, az extra változó használata véleményem szerint egy luxus amit megengedhetünk magunknak. Fontos kiemelni azt is, hogy bár ez a megoldás integer alapú adatokkal működik, más objektumokkal már nem valószínű. (pl.: char, string, stb).

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!

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

labda.c

```
            //labda.c

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


void DrawMap(int r, int c, int cr, int cc)
{

    for (int i = 0; i<(r+2); i++)
    {

        for (int j = 0; j < (c+2); j++)
        {

            if (i == (cr+1) && j == (cc+1))
            {
                printf("o");
            }
            else
            {
                printf(" ");
            }

            if (i == 0 || j == 0 || i == (r+1) || j == (c+1) )
            {
                printf(".");
            }

        }
```

```
        printf("\n");
    }

}
```

Ezen metódus rajzolja le a pályát minden iterációjában. Megkapja értékül a pálya méreteit amit a program elején mi adunk meg, hogy tudja pontosan mekkora pályát kell rajzolnia, ezen felül megkapja a lapda jelenlegi pozícióját is, hogy tudja hova kell berajzolnia.

```
int main()
{

    int row;
    int coloumn;

    printf("number of rows -> "); scanf("%d",&row); printf("\n");
    printf("number of coloumns -> "); scanf("%d",&coloumn); printf("\n");

    int yn = 1;


    int CurrentRow = row /2;
    int CurrentColoumn = coloumn/2;
    int prevRow = CurrentRow - 1;
    int prevCol = CurrentColoumn - 1;
```

Deklaráljuk a fő változókat, illetve felvesszük a pálya méreteit. Ezen felül 'beállítjuk' a labdát a pálya közepére, ileltve megadjuk neki a kezdő irányt, hogy merre 'pattanjon'.

```
    while (1)
    {

    DrawMap(row, coloumn, CurrentRow,CurrentColoumn);

    sleep(1);
```

Lerajzoljuk a pályát, illetve várunk egy másodpercet. Ha ezt nem tennénk meg a program túl gyorsan rajzolná a pálya egymás utáni iterációit, így nem látnánk mi is történik valójában.

```
    if ( CurrentRow == 0)
    {
        prevRow = -1;
    }
    if (CurrentRow == row-1)
```

```
    {
        prevRow = row+1;
    }
    if (CurrentColoumn == 0)
    {
        prevCol = -1;
    }
    if (CurrentColoumn == coloumn-1)
    {
        prevCol = coloumn+1;
    }



    if (CurrentRow < prevRow)
    {
        CurrentRow--;
    }
    else { CurrentRow++; }
    if (CurrentColoumn < prevCol)
    {
        CurrentColoumn--;
    } else {CurrentColoumn++;}
```

Megadjuk hogy a pálya következő iterációjában hol fog elhelyezkedni a labda.

```
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    printf("\n");
    }









    return 0;
}
```

Végül pedig beteszek néhány sort hogy elválaszthatóak legyenek a külön iterációk. Ez egy elég primitív megoldás, de a feladat leírásának eleget tesz.

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!

Tutoráltak: Ratku Dániel/

szohossz.c

```
    //szohossz.c


            int main(void)
{
    int h = 0;
    int n = 0x01;
    do{ ++h;}
    while (n<<=1);
    printf("Szóhossz -> "); printf("%d",h); printf(" bit" ); printf("\n");
    return 0;

}
```

Egy egyszerű program mely biteltolással határozza meg hogy pontosan hány számjegy 'fér el' az intben, ezzel meghatározva hány bites a szóhossz.

BogoMIPS.c

```
    //BogoMIPS.c

#include <time.h>
#include <stdio.h>
void
delay (unsigned long long int loops)
{
unsigned long long int i;
for (i = 0; i < loops; i++);
}
int main (void)
{
unsigned long long int loops_per_sec = 1;
unsigned long long int ticks;
printf ("Calibrating delay loop..");
fflush (stdout);
```

```
while ((loops_per_sec <<= 1))
{
ticks = clock ();
delay (loops_per_sec);
ticks = clock () - ticks;
printf ("%llu %llu\n", ticks, loops_per_sec);
if (ticks >= CLOCKS_PER_SEC)
{
loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;
printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
(loops_per_sec / 5000) % 100);
return 0;
}
}
printf ("failed\n");
return -1;
}
```

```
           ./BogoMIPS
Calibrating delay loop..4 2
2 4
2 8
3 16
3 32
3 64
4 128
5 256
7 512
13 1024
23 2048
43 4096
85 8192
169 16384
334 32768
909 65536
1597 131072
903 262144
2374 524288
3372 1048576
8161 2097152
14339 4194304
25820 8388608
48024 16777216
91928 33554432
176512 67108864
```

```
366557 134217728
737364 268435456
1482899 536870912
ok - 724.00 BogoMIPS
```

A kimenet jobb oldali oszlopában a kettő hatványait láthatjuk, míg a bal oldali oszlopában azt figyelhetjük meg hogy az eredmény kiszámolásáig hányszor tickelt a cpu. Használata: A BogoMIPS használatával próbálják belőni a cpu sebességét Linux kernelben.

2.6. Helló, Google!

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

pagerank.c

```
    //pagerank.c

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

void
kiir (double tomb[], int db)
{
int i;
for (i=0; i<db; i++)
printf("PageRank [%d]: %lf\n", i, tomb[i]);
}

double tavolsag(double pagerank[],double pagerank_temp[],int db)
{
double tav = 0.0;
int i;
for(i=0;i<db;i++)
tav +=abs(pagerank[i] - pagerank_temp[i]);
return tav;
}

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

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

long int i,j,h;
i=0; j=0; h=5;

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

if ( tavolsag(PR,PRv, 4) < 0.00001)
break;
}
kiir (PR,4);
return 0;

}
```

A PageRank elsősorban a Google-hoz kötődik. Célja a jó minőségű oldalak felderítése. A kimenő linkeket veszi alapul, feltételezve, hogy a kimenő linkek az adott oldal minőségéről tanúskodnak. Tehát ha van A és B oldal, és A oldalon van egy hiperlink amely B oldalra mutat, akkor A veszít az értékéből, B értéke pedig nő. Fontos kiemelni, hogy minden oldalnak kezdetben 1-es az értéke.

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Tutoráltak: Ratku Dániel/

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

monty_hall.r

```
            //monty_hall.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 játék lényege a következő: A játékos előtt három ajtó áll. A három ajtó közül az egyikben értékes nyeremény van, a másik kettőben pedig értéktelen. A játékos célje ebből adódóan az értékes nyeremény megszerzése. A játékos kiválaszt egy ajtót, azonban ekkor a műsorvezető kinyit a másik két ajtó közül egyet, amiben biztosan az egyik értéktelen nyeremény van. A játékos ekkor választhat: Marad az eredetileg kiválasztott ajtónál és kinyitja, vagy átvált a harmadik, még ki nem nyitott ajtóra. Bár lehet hogy elsőre úgy tűnhet, hogy a nyeremény megszerzésére körülbelül 33% esélye van a játékosnak, ez messzebb sem állhatna az igaszságtól. Egy ügyes játékosnak ugyanis 66% esélye van. Abban az esetben ha a játékos

úgy dönt hogy mindig átvált a másik, még ki nem nyitott ajtóra, akkor ha odafigyelünk láthatjuk hogy a fő célja a játékosnak megtalálni az egyik értéktelen ajtót (hiszen ha kiválaszt egy értéktelen ajtót, a második automatikusan kiesik, hisz a műsorvezető kinyitja azt, így ha ajtót vált garantáltan a nyer). Mivel két értéktelen ajtó van,a háromból, így abban az esetben ha a játékos mindig ajtót vált, 66% esélye van a győzelemre.

Az R szimuláció 100 próbálkozás után:

```
[1] "Kiserletek szama: 100"
> length(nemvaltoztatesnyer)
[1] 36
> length(valtoztatesnyer)
[1] 64
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.5625
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 100
>
```

Az R szimuláció 100 000 próbálkozás után:

```
[1] "Kiserletek szama: 100000"
> length(nemvaltoztatesnyer)
[1] 33292
> length(valtoztatesnyer)
[1] 66708
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.4990706
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 100000
>
```

Az R szimuláció 1 000 000 próbálkozás után:

```
[1] "Kiserletek szama: 1000000"
> length(nemvaltoztatesnyer)
[1] 333805
> length(valtoztatesnyer)
[1] 666195
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.501062
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 1000000
>
```

2.8. 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

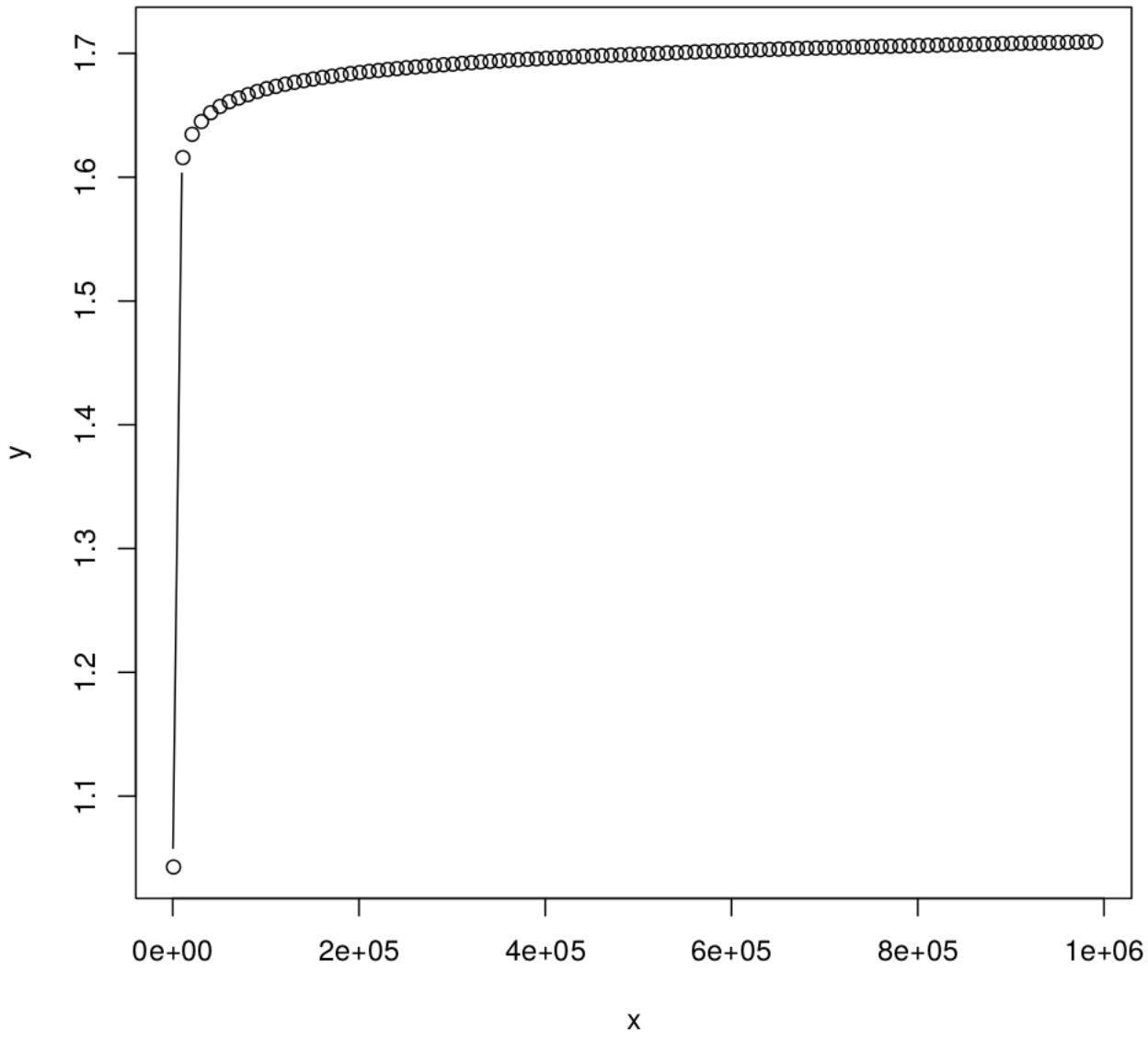
brun.r

```
            //brun.r


library(matlab)

stp <- function(x){

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

x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

A Brun tétel: Az ikerprímek reciprokösszegük egy konstanshoz konvergálnak. Ezt a konstanst napjainkban még mindig csak megbecsülni tudják, megközelítő értéke: 1.902160583104 Ezen feladatban a Brun konstansot kell megpróbálni megközelíteni, tehát leszimulálni.

A fentebbi R kód bár eleget tesz a feladatnak, maga a megoldása elég egyszerű. A program egy megadott határértékig megkeresi az összes prímet, majd ezen prímek közül megkeresi azokat, amelyek ikerprímeknek felelnek meg. Végül pedig ezen príem reciprokjait összeadja.

```
>
> x=seq(13, 1000000, by=10000)
> y=sapply(x, FUN = stp)
> plot(x,y,type="b")
>
```



2.1. ábra. A B_2 konstans közelítése

Werkfilm

- https://youtu.be/VkMFrgBhN1g
- https://youtu.be/aF4YK6mBwf4

3. fejezet

Helló, Chomsky!

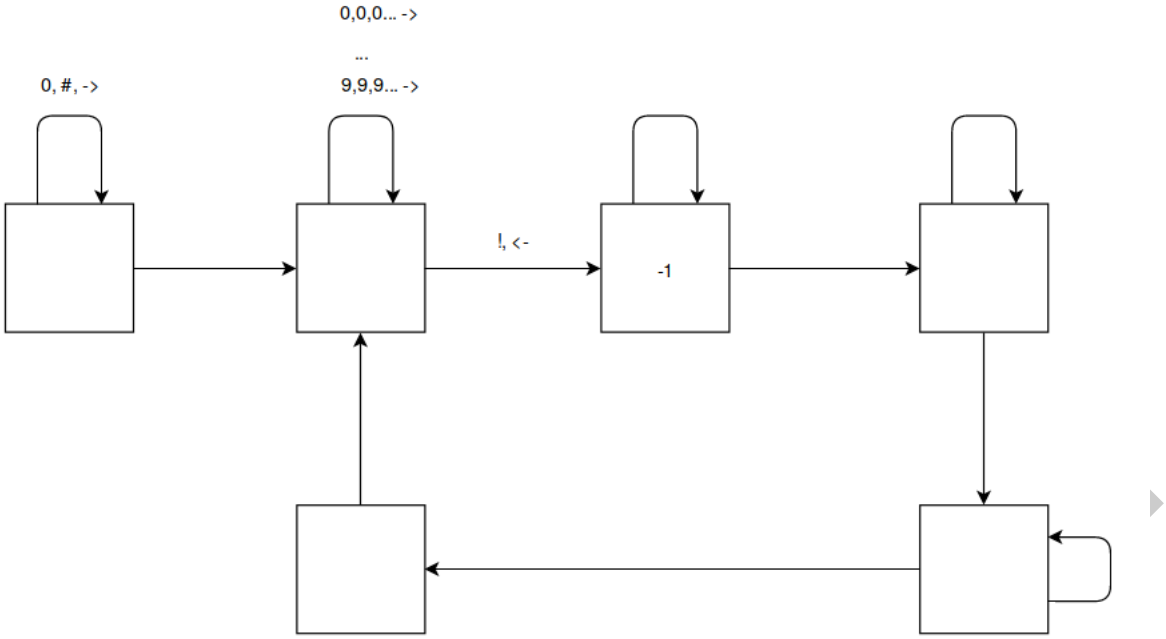
3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Elkezdjük beolvasni a decimális értéket. Üreset írunk vissza nulla és üres karakter esetén, és addig lépünk jobbra, amíg el nem érjük az első igazi karaktert.

A beolvasott karaktereket visszaírjuk egészen addig amíg el nem érjük a karakterlánc végét (!). Itt csökkentjük egyel a beolvasott decimális szám értékét, kivéve ha nullára végződik, ebben az esetben az utolsó kettő számot csökkentjük. Az olvasott értéket ezután visszaírjuk és jobbra lépünk.

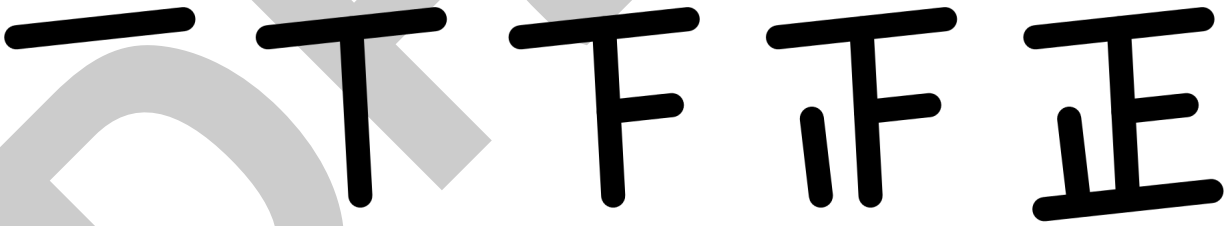
Jobbra léptetve írunk és olvasunk amíg üres helyhez nem érünk. Minden üres helyre egyet helyettesítünk és lépünk tovább. Amíg van mit olvasni, a ciklus folytatódik.



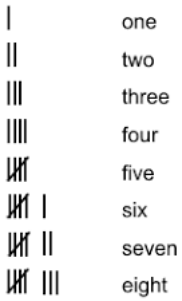
3.1. ábra. Turing gép decimálisból unárisba

Leegyszerűsítve tehát: A beolvasott decimális számból addig vonunk le egyeket, amíg nulla nem lesz, és minden egyest kiírunk, így átírva a decimális számot unárisba.

Néhány péda unáris számrendszerre:



3.2. ábra. Japánban, Kínában, és Koreában használt



3.3. ábra.

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

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

A generatív nyelvtan négy alaprésze: - Terminális szimbólumok: ábécé betűi - Nem terminális szimbólumok: generálási segédeszközként jelenlévő ábécé. - Kezdőszimbólum: egy kitüntetett szimbólum. - Helyettesítési szabályok

Adott egy kezdő szó, melyet ebben a feladatban az 'S' jelöl. Ezt a szót az általunk lefektetett helyettesítési szabályok mentén addig alakítjuk amíg csak terminális szombólumok nem maradnak. A feladatban megadott nyelv környezetfüggő, ami az jelenti hogy nem lehet

```
H -> h
```

hozzárendelési szabállyal legenerálni.

```
Első gramatika szabályok:


S -> abBc

Változók cseréje
Bc -> bcc
A -> aa
bB -> bbc
abB -> AbB
```

Tehát:

```
S (S -> abBc)
abBc (abB -> AbB)
AbBc (A -> aa)
aabBc (Bc -> bcc)
aabbcc
```

```
Második gramatika szabályok:
```

```
S -> aabXcD

Változók cseréje
X -> bD
D -> cc
bX -> bbX
aabX -> aaabX
```

Tehát:

```
S ( S -> aabXcD )
aabXcD ( D -> cc)
aabXccc ( aabX -> aaabX)
aaabXccc ( aabX -> aaabX)
aaaabXccc (aabX -> aaabX)
aaaaabXccc (bX -> bbX)
aaaaabbXccc (bx -> bbx)
aaaaabbbXccc (bx -> bbx)
aaaaabbbbXccc (X -> bD)
aaaaabbbbbDccc (D -> cc)
aaaaabbbbbccccc
```

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.

BNF: Backus-Naur-Form egy környezetfüggetlen nyelvtanok leírásában használt metaszintaxis, mellyel formális nyelvek írhatóak le. Leegyszerűsítva: nyelvtanokat definiáló nyelvtan.

Utasítások:

```
<utasítás>::= <címkézett utasítás>,<kifejezés utasítás>, < ↩
   összetett utasítás>, <kiválasztó utasítás>, <ismétlődő  ↩
   utasítás>, <jumo utasítás>

<címkézett utasítás>::= <azonosító>:: <utasítás> | case < ↩
   állandó kifejezés>: <utasítás>| default: <utasítás>

<kifejezés utasítás>::= <kifejezés opc> //opc: opcionális,  ↩
   abban az esetben ha hiányzik, ez egy null utasítás

<kiválasztó utasítás>::= if (<kifejezés>) <utasítás> | if (< ↩
   kifejezés>) <utasítás> else <utasítás> | switch (<kifejezés ↩
   >) <utasítás>

<ismétlődő utasítások>:: = while (<kifejezés>) <utasítás> | do  ↩
   <utasítás> while (<kifejezés>); | for (<kifejezés opc> < ↩
   kifejezés opc>; <kifejezés opc> <utasítás>
```

```
            <jumo utasítás>::= goto <azonosító>; | continue; | break; |  ↩
               return <kifejezés opc>
```

Kódcsipet, mely nem fordul le C89-ben:

```
            //old.c
            int main()
{
  printf("Ez nem fog lefordulni C89-ben");
  //mivel abban a verzióban még nem lehetett így kommentelni.
}
```

```
                gcc old.c -o old -std=c89
old.c: In function ‘main’:
old.c:3:2: warning: incompatible implicit declaration of built-in function  ↩
   ‘printf’
  printf("Ez nem fog lefordulni C89-ben");
  ^~~~~~
old.c:3:2: note: include ‘<stdio.h>’ or provide a declaration of ‘printf’
old.c:4:2: error: C++ style comments are not allowed in ISO C90
  //mivel abban a verzióban még nem lehetett így kommentelni.
  ^
old.c:4:2: error: (this will be reported only once per input file)
```

3.4. Saját lexikális elemző

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU (15:01-től).

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.l

```
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
%%
{digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
```

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

A lexer analízist végez az inputon bejövő karaktersorozatokra, tokenekké átalakítva azt. Az általunk használt lexer C forráskódot fog legenerálni.

```
        #include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
```

Itt számoltatjuk vele a bejövő számokat, melyek [0-9] lehetnek.

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

Itt mondjuk meg a programnak a tokenek szabályait. Ezen szabály szerint a program feljegyez akármilyen numerikus karaktert, melyet követ még valamilyen akrakter. Abban az esetben ha talál ilyet a megnő a realnumbers változó értéke egyel. (itt számolja a valós számokat), melyet kiír.

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

Végül pedig kiírja a valós számok darabszámát.

Lefuttatás menete:

```
            lex -o realnumber.c realnumber.l
```

```
            gcc realnumber.c -o realnumber -lfl
```

```
            ./realnumber
iknvsin3453nbs bu23bvs73
iknvsin[realnum=3453 3453.000000]nbs bu[realnum=23 23.000000]bvs[realnum=73 ↩
    73.000000]
```

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás videó: https://youtu.be/06C_PqDpD_k

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.l

Az interneten számos szleng alakult ki bizonyos szavakra. A számtalan rövidítések mellett megjelentek olyan szavak is, melyek a kifejezésekben található betűket cserélnek le más, de hasonló karakterekre, ezzel 'hangsúlyt' adva a szavaknak. Ez a program is ezt fogja csinálni, megadott karaktereket fog lecserélni, ezzel szlengesítve az inputot.

```
%{
  #include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <ctype.h>

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

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

A kód első sorában adjuk meg a fő struktúrát. Például *leet[4]-ben fogjuk megadni hogy egyes karakterekhez milyen karakter típust lehet hozzárendelni.

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

  {'0', {"D", "0", "D", "0"}},
```

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

Itt határozzuk meg, hogy milyen karaktert milyen karakterre lehet lecserélni.

```
// https://simple.wikipedia.org/wiki/Leet
  };

%}
%%
. {

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

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

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

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

        found = 1;
        break;
      }
```

Itt pedig randomizáljuk azt, hogy melyik karaktert fogja választani a megadottak közül.

```
    }

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

  }
```

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

A program futtatásának lépései:

```
            lex -o l33t.c l33t.l
```

```
            gcc l33t.c -o l33t
```

Futtatva pedig a következőt kapuk:

```
            ./l33t
let us try this program to see if it works
|_3t us tr th1s pr0gr4m t0 533 1f 17 w0rks
```

Ne felejtsük el hogy az angol ábécé betűit adtuk meg neki, így ha esetleg ékezetes betűt adnánk meg neki akkor azok nem fognak változni

```
             ./l33t
Az ékezetes betűk nem fognak változni
4z ék3z3t3s b3tűk /V3m f0gn4k vált0zn1
```

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.)

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

Ha a 'SIGINT' nincs ignorálva, akkor a 'jelkezelo' kezelheti a 'SIGINT'-et

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

Egy 5-ször végrehajtandó for ciklus mely minden ciklusában 1-es inkrementációval hajtódik végre. Az i-nek az értéke minden inkrementálás végén (i+1).

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

Egy 5-ször végrehajtandó for ciklus, mely minden ciklusában 1-es inkrementácival hajtódik végre. Az i-nek az értéke minden inkrementálásnál egyel növekszik.

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

Egy for ciklus, mely folyamán egy tömb elemeit a következőnek adja meg:

```
tomb[0] = 0;
tomb[1] = 1;
tomb[2] = 2
tomb[3] = 3
tomb[4] = 4
```

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

Egy for ciklus, melynek minden ciklusában az i értéke növekszik egyel. Ezen felül a ciklus véget ér, ha i értéke kisebb n-nél, illetve amíg d és s pointerek ugyanarra mutatnak.

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

Kiírja az ’f(a, ++a)’ függvény értékét, majd kiírja az ’f(++a, a)’ függvény értékét. Mindkét érték integer.

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

Kiírja az f(a) függvény értékét, majd az a változó értékét.

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

Kiírja a f(&a) függvény értékét, illetve az a változó értékét. Mindkét esetben integer alapú érték kerül kiírásra.

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

Végtelen prímszám létezik

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

Végtelen ikerprímszám létezik

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

Véges sok prímszám létezik

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

Véges sok prímszám létezik.

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

```
                //declaration.cpp

                int *getPointers(int* array);
typedef int(*KillMe) (int, int);
int PleaseLoL(int a, int b);
KillMe SmokeWeed(int);



int main()
{
  int a = 1; /* egész */

  int *b = &a; /* egészre mutató mutató */

  int &c = *b; /* egész referenciája */

  int d[5] = {0,1,2,3,4}; /* egészek tömbje */

  int (&e)[5] = d; /* egészek tömbjének referenciája */

  int *f[5]; /* egészre mutató mutatók tömbje */

  int *g = getPointers(d); /* egészre mutató mutatót visszaadó függvény */

  int* (*asd)(int*) = getPointers; /* egészre mutató mutatót visszaadó ↩
     függvényre mutató mutató*/

  KillMe EndMyMisery = SmokeWeed(420); /* egészet visszaadó és két egészet ↩
     kapó függvényre mutató mutatót
visszaadó, egészet kapó függvény */

  return 0;
}

int* getPointers(int* array)
{
  return array;
}


int PleaseLoL(int a, int b)
{
  return 420;
}
```

```
KillMe SmokeWeed(int everyday)
{
  return PleaseLoL;
}
```

- ```
  int a;
  ```

  Egy a nevű egész változót
- ```
  int *b = &a;
  ```

  Egy egész címére mutató mutatót.
- ```
  int &r = a;
  ```

  Egy egészre mutató referencia.
- ```
  int c[5];
  ```

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

  Egy öt elemű egészekből álló tömbre mutató mutatót, ami c-re mutat.
- ```
  int *d[5];
  ```

  Egy öt elemű egészekre mutató mutatókból álló tömb.
- ```
  int *h ();
  ```

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

  Egy mutatóval visszatérő függvénymutató.
- ```
  int (*v (int c)) (int a, int b)
  ```

  Egy egésszel visszatérő két egészet váró függvényre mutató mutató.
- ```
  int (*(*z) (int)) (int, int);
  ```

  Egy egésszel visszatérő két egészet váró függvényre mutató egészet váró függvényre mutató mutató.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c

Mielőtt belekezdenénk a feladatba, fontos megjegyezni hogy mik a háromszögmátrixok. Két fajta háromszögmátrix létezik: Alsó- és felső háromszögmátrix. Ha alsó háromszögmátrixról beszélünk akkor, arra gondolunk, hogy a mátrix főátlója felett, csak nullások szerepelnek, felső háromszög esetén pedig a főátló alatt szerepelnek csak nullások. Fontos azt is megjegyezni hogy minden háromszögmátrix oszlopainak és sorainak száma megegyezik.

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

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

            printf("tm memóriacím (double** memóriacíme)\n");
            printf("%p\n", &tm);
```

Először is felvesszük a fő paramétereinket. Az 'nr' változó fogja jelölni a sorok számát, a 'tm' pedig maga a háromszögmátrix lesz.

```
    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
```

```
        return -1;
    }

            printf("double* memóriacíme\n");
            printf("%p\n", tm);
```

Abban az esetben ha nem tudunk elég helyet felszabadítani (allokálni) a tm-nek a malloc paranccsal (memory allocation), akkor a program leáll return -1-el.

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

    }

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

Abban az esetben ha nem sikerül elég helyet felszabadítani a memóriában azokra a mutatókra amikre a tm for mutatni, akkor a program leáll.

```
    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");
    }
```

Feltöltjük a háromszögmátrixot elemekkel majd kiírjuk azokat.

```
    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;
    *(tm[3] + 2) = 44.0;
    *(*(tm + 3) + 3) = 45.0;
```

Értékeket adunk adott helyekre a mátrixon.

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

Kiírjuk a mátrix elemeit.

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

    free (tm);

    return 0;

}
```

Lefuttatva a programot láthatjuk hogy minden rendben történt a memóriafoglalással.

```
     ./tm
tm memóriacím (double** memóriacíme)
0x7fff48341150
double* memóriacíme
0x5557ccd06670
double memóriacíme
0x5557ccd066a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
```

A végén pedig felszabadítjuk a helyet amit lefoglaltunk a program elején.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Az XOR titkosító egy elveiben nagyon egyszerű titkosító. A lényege, hogy a inputba kap egy kulcsot, illetve egy hosszabb szöveget, mindkét inputot bitjeiben veszi, és a bejövő szöveg bitjeit az XOR müvelet

alapján 'titkosítja' a kulcs bitjeivel. Az aki tudja a kulcsot ugyanazzal az XOR művelettel vissza tudja kapni az eredeti kulcsot.

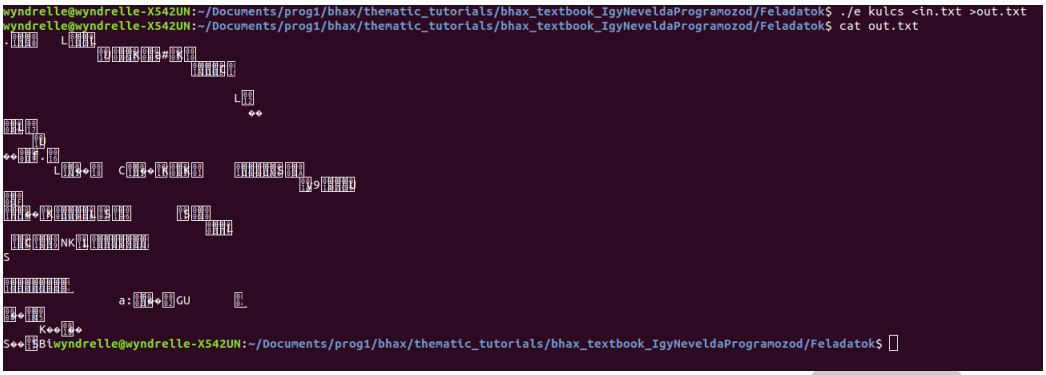
```
           //e.c

           #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);
```

Megadjuk a fő változóinkat.

```
  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);

    }

               }
```

Buffereljük az inputszöveget amíg tudjuk, minden bitjét pedig a kulccsal xorozzuk. Végül pedig kiírjuk a szöveget, mely mehet a standar outputra, vagy fájlba is attól függően hogy hogyan adtuk meg a parancssori argumentumokat.

Ez hogy néz ki:



4.1. ábra.

Tanulságok, tapasztalatok, magyarázat...

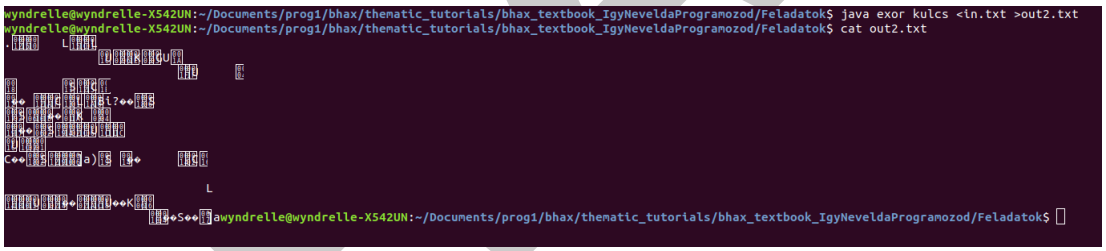
4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

```
            //exor.java
            public class exor {


public class exor {

    public exor(String keyString,
            java.io.InputStream inputStream,
            java.io.OutputStream outputStream)
            throws java.io.IOException {

        byte [] kulcs = keyString.getBytes();
        byte [] buffer = new byte[256];
        int keyIndex = 0;
        int readBytes = 0;

        while((readBytes =
                inputStream.read(buffer)) != -1) {

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

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

            }
```

```
            outputStream.write(buffer, 0, readBytes);

        }

    }

    public static void main(String[] args) {

        try {

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

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Működése ugyanolyan mint a c-beli változata, a különbség annyi hogy itt Java-ban van megírva.



4.2. ábra.

4.4. C EXOR törő

Ez a program úgy nevezett brute force-al töri fel a titkosított szöveget. Nem valami hatásos, rengeteg erőforrást igényel, mivel minden létező kulcson végig kell menne, azonban mindenféleképpen hatásos. Egy másik nagy hátránya, hogy a kódtörőnek tudnia kell hány karakteres a kulcs.

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

#include <stdio.h>
```

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

include-oljuk a szükséges dolgokat. Itt definiáljuk a kulcs méretét is, mely itt ebben a programban 8. Ezen felül definiálásra kerül még a buffer mérete is.

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

  return (double) titkos_meret / sz;
}

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

  double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}
```

Ez a szekció azért felelős hogy a program felismerje ha tiszta szövegre bukkan. Mivel a számítógép nyilvánvalóan nem tud értelmezni szöveget, ezért megadunk neki néhány gyakrabban használt szót melyek ebben az esetben a 'hogy', 'nem', 'az', 'ha'. Ha ezekre rábukkan akkor kiírja a kulcsot melynek használatával rábukkant erre, illetve kiírja az xoros szöveget is.

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

  int kulcs_index = 0;

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

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

```
      kulcs_index = (kulcs_index + 1) % kulcs_meret;

    }

}

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

  exor (kulcs, kulcs_meret, titkos, titkos_meret);

  return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

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

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

  // maradek hely nullazasa a titkos bufferben
  for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';

  // osszes kulcs eloallitasa
  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);
      }

  return 0;
}
```

A program utolsó fázisában láthatjuk ahogy egyesével végigmegy az előállítható kulcsokon, és ellenőrzi hogy megjelentek-e a gyakori szavak.

lefuttatva így néz ki:

```
            ./e 00000001 <in.txt > out.txt
            ./t <out.txt
Kulcs: [00000001]
Tiszta szoveg: [az Egyszer volt hol nem volt, volt egyszer egy magányos txt ↩
    file.
Léte abba merült ki hogy gazdája random programok parancssori argumentumába ↩
    írta bele.

Ez valóban egy igencsak magányos és szomorú lét, hogy boldog volt-e? nem

ha azonban lefordulna ez a cucc annak nagyon örülnék.
]
```

Tanulságok, tapasztalatok, magyarázat...

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

Neurális hálókat legfőképp gépi tanuláshoz használnak. A neuronok, melyeket legkönnyebben egy nagy gráf elemeiként tudunk elképzelni kommunikálnak egymással. Ezek a neuronok legalább három rétegben vannak elhelyezve. A bemeneti rétegben nem történik semmi, azonban a rejtett rétegekben különféle műveletek hajtódnak végre, melyek a neurális háló gerincét képzeik.

A következő program lefuttatása után láthatjuk ahogy ez az R nyelvben íródott program megtanulja az OR, AND és EXOR kapukat.

```
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)
```

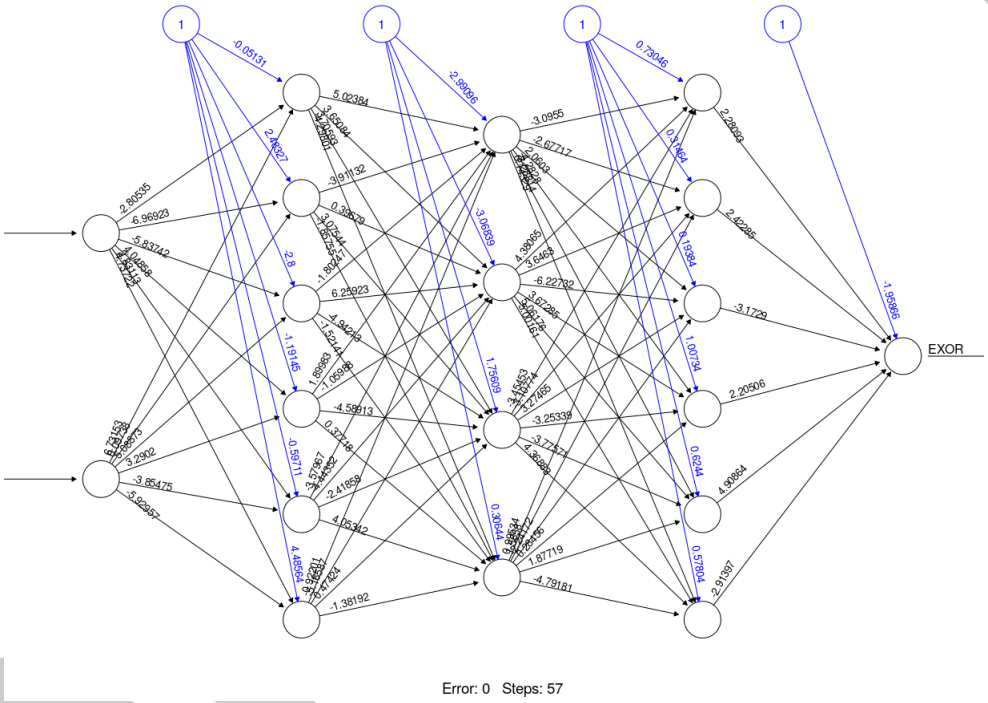
```
plot(nn.exor)

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])
```

Miután letöltöttük a neuralnet csomagot, a programot futtathatjuk is.

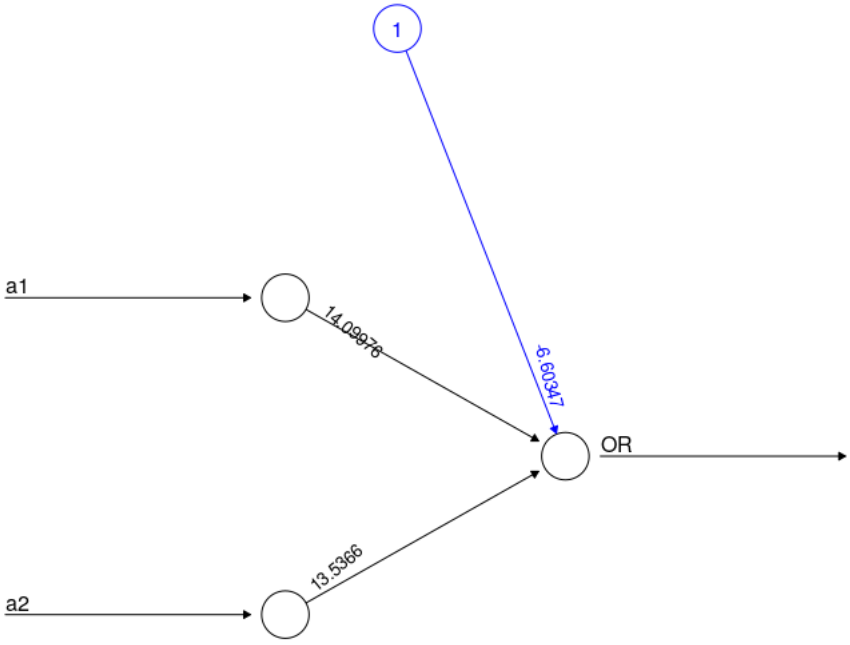
```
                   $neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1

$neurons[[2]]
     [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
[1,]    1 0.61826996 0.47173771 0.36548935 0.91902536 0.86546616 0.80702504
[2,]    1 0.99216843 0.02119818 0.97865534 0.06052892 0.99895668 0.09329873
[3,]    1 0.02954135 0.97322920 0.01030473 0.99963950 0.06385189 0.99842321
[4,]    1 0.70423361 0.46855755 0.45318772 0.94026710 0.91032743 0.93968626

$neurons[[3]]
     [,1]         [,2]         [,3]        [,4]       [,5]
[1,]    1 1.177180e-03 2.429896e-03 0.998396000 0.99682623
[2,]    1 9.982752e-01 1.214920e-07 0.003284555 0.99999920
[3,]    1 2.212019e-06 9.943069e-01 0.999998730 0.01409462
[4,]    1 1.275600e-03 9.200916e-04 0.998455350 0.99911935

$neurons[[4]]
     [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
[1,]    1 0.99739291 0.98437949 0.04652966 0.98731380 0.01096734 0.01191996
[2,]    1 0.99999293 0.04148374 0.46562351 0.02323602 0.98358954 0.98375911
[3,]    1 0.03192529 0.20509612 0.98744873 0.29616955 0.93373216 0.75232819
[4,]    1 0.99744281 0.98452897 0.04595664 0.98741608 0.01083823 0.01181926
```

```
$net.result
             [,1]
[1,] 0.0006787719
[2,] 0.9996485352
[3,] 0.9997862854
[4,] 0.0006762063

>
```

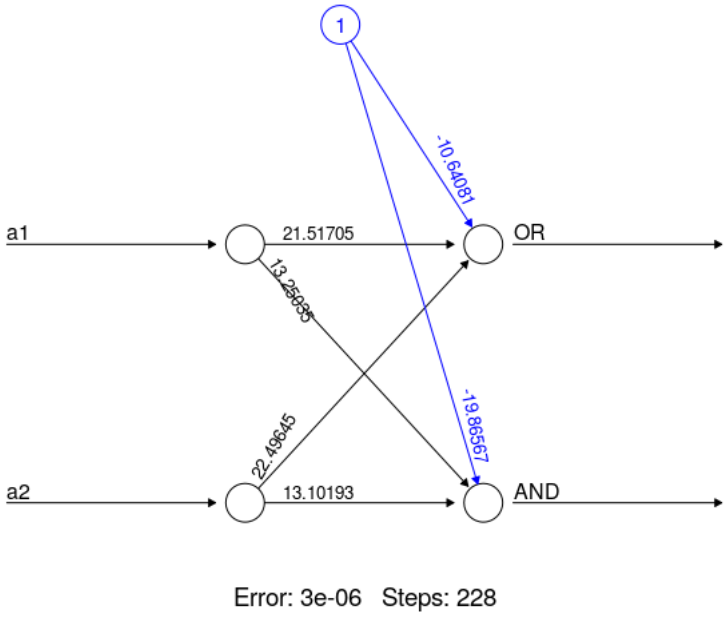
Láthatjuk ahogy a program 'megtanulta' hogy melyik logikai kapu milyen bemenettel milyen értéket fog visszaadni.



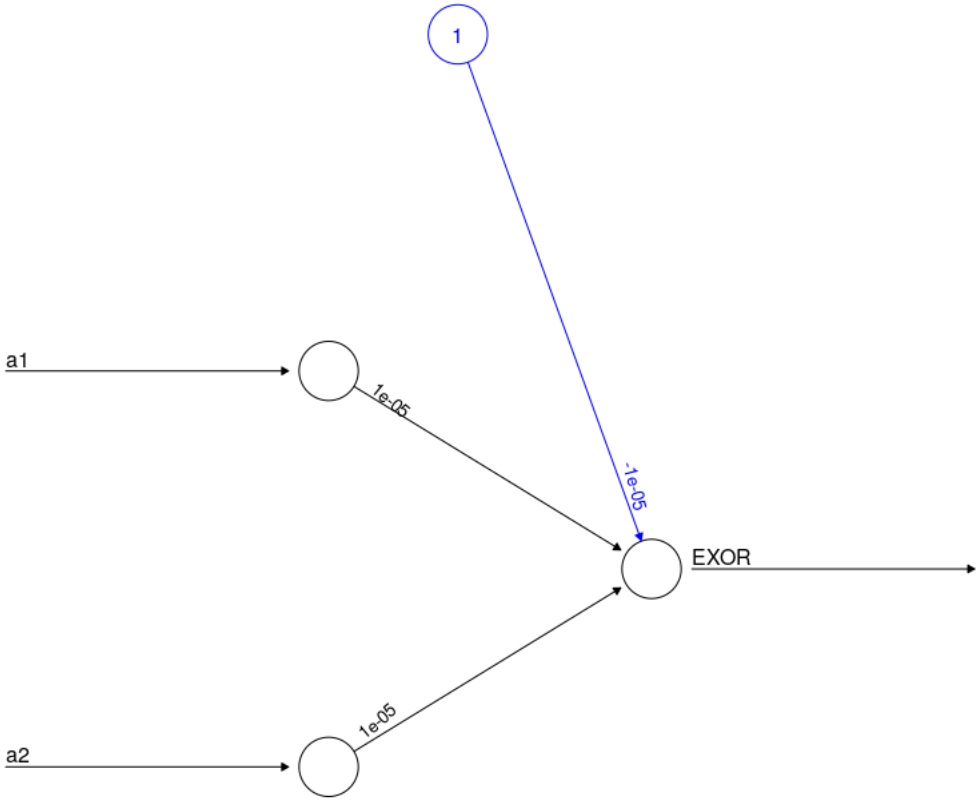
4.3. ábra. EXOR



4.4. ábra. OR



4.5. ábra. AND OR



4.6. ábra. EXOR

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: https://youtu.be/XpBnR31BRJY

Megoldás forrása: https://github.com/nbatfai/nahshon/blob/master/ql.hpp#L64

Tanulságok, tapasztalatok, magyarázat...

SKIP - 2

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

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

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

Még mielőtt nekikezdenénk a feladatnak, fontosnak tartom hogy tisztázzuk mi is az a Mandelbrot halmaz. A Mandelbrot halmaz egy fraktál-minta, mely a virtuálisan a komplex síkon vizualizálható. Fontos információk: Komplex szám. Bizonyára mindenki találkozott már négyzetszámokkal, így mindenki meg tudja mondani hogy mi 25-nek a gyöke. Ez 5, viszont az hogy mi -25-nek a négyzetgyöke már egy érdekesebb téma. Mivel ilyen szám nem létezik, a matematikában a -1-nek a négyzetgyökét elnevezték 'imaginary number'-nek, azaz kitalált számnak, melyet 'i'-vel jelölünk. Ezen kitalált szám segítségével már meg tudjuk mondani a -25-nek a gyökét ami 5i-lesz, egy komplex szám. A fraktál-mintára nincs elfogadott definíció a matematikában.

A kód lefuttatásához szükségünk van png++-ra amit a következő linken tudunk elérni: png++ A részletes telepítési útmutató is megtalálható ezen az oldalon.

mandelpngt.c++

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

#define MERET 600
#define ITER_HAT 32000

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


    clock_t delta = clock ();

    struct tms tmsbuf1, tmsbuf2;
```

```
    times (&tmsbuf1);
```

Felveszünk két változót, mellyel az időt fogjuk mérni. Ez azért fog kelleni hogy láthassuk mennyi idő alatt futott le a program.

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

A számítás adatai. Ezek a változók lesznek szükségesek ahhoz hogy 'meg tudjuk rajzolni' a mandelbrot halmazt, illetve az annak kiszámítására fognak szolgálni.

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

A fő számítás a már fentebb deklarált és értékkel hozzáfűzött változókkal itt történik meg.

```
    int iteracio = 0;

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

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

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

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

            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;

                ++iteracio;

            }
```

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

255 iteráció alatt végigmegyünk a rácson, kiszámítva minden pontjára a mandelbrot halmazt. Technikailag itt történik a 'rajzolás'.

```
    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;

}
```

Kiírjuk az időt, hogy tudjuk mindez mennyi időbe tellett.

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

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

Ha esetleg rosszul használná a felhasználó a fájlt, egy rövid emlékeztetőt írunk ki neki standard outputra, hogy tudja hogyan kell lefuttatni.

```
    int kepadat[MERET][MERET];

    mandel(kepadat);

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

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

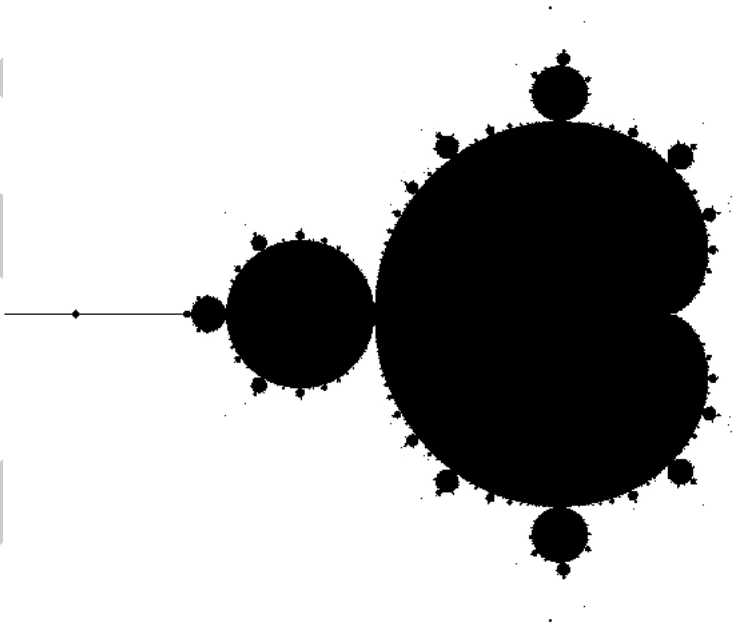
```
                                                255 -
                                                (255 * kepadat[j][k]) / ITER_HAT ↩
                                                   ,
                                                255 -
                                                (255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

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

}
```

A futtatásának módja:

```
            g++ mandelpngt.c++ -lpng16 -o mandel
```

```
            ./mandel mandel.png
```

És íme a végeredmény:



5.1. ábra. Mandelbrot halmaz

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

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása:

A feladat ugyanaz mint az előzőben: meg kell jeleníteni vizuálisan a mandelbrot halmazt, azonban ezúttal most C-ben. A feladat jellege miatt, maga a program nem fog sokban különbözni az előzőtől, mivel mindkettő ugyanazt hajtja végre, csupán különböző nyelven. Mivel C++-ban kicsit több eszközünk van dolgozni, itt több mindent meg kell adni, vagy máshogy kell megadni hog minden működjön.

3.1.2.cpp

```
//3.1.2.cpp
#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;

}
```

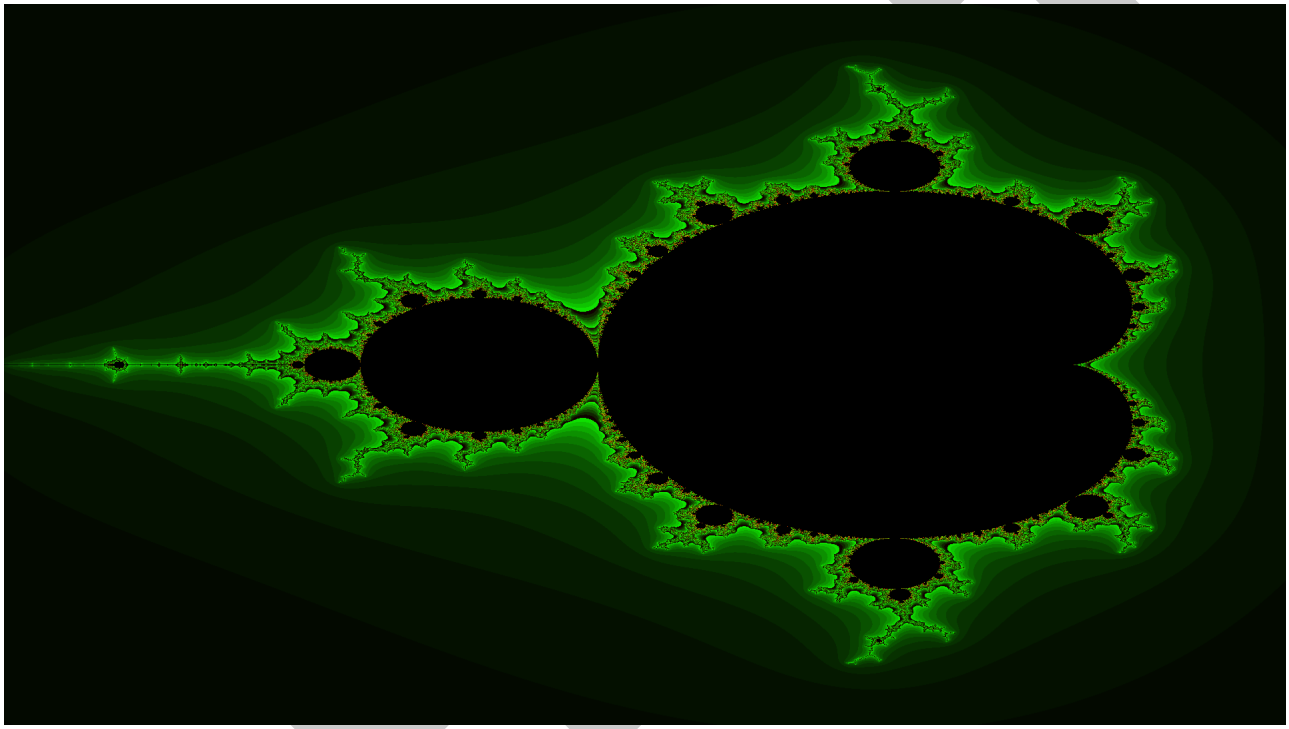
Futtatása:

```
g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
```

```
./3.1.2 mandel.png hosszúság magasság n a b c d
```

```
./3.1.2 mandel.png 1920 1080 255 -2 0.7 -1.35 1.35
Szamitas
mandel.png mentve.
```

A végeredmény:



5.2. ábra. Mandelbrot halmaz

Láthatjuk hogy a Mandelbrot halmaz itt is kirajzolódid, igaz, más színkompzícióval.

5.3. Biomorfok

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

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

A lényeges különbség e között és az előző két feladatt között a 'c', ami a Mandelbrot halmazos feladatokban változó, itt, a Júlia halmaz vizsgálatánál, kirajzolásánál állandó lesz. Kialakulása egy bugnak köszönhető amire Clifford bukkant rá amikor a Júlia halmaz elemeit próbálta megközelíteni. A bug miatt a képek melyeket a kód generált egysejtű organizmusokra kezdtek hasonlítani, mely miatt Clifford úgy hitte rábukkant a természet egy törvényére.

A kód:

biomorf.cpp

```
                                        //biomorf.cpp

#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;
    }
```

Inicializásioc lépések. A programban deklaráljuk a fő változókat, illetve a parancssori argumentumokból értéket is rendelünk hozzájuk. Fontos kiemelni hogy abban az esetbe ha hiányzik egy argumentum a program automatikusan kiírja hogy hogyan kell futtatni.

```
    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 ));
        }
```

A program ezen szakaszában történnek meg a tényleges műveletek. Végigmegyünk az összes soron és oszlopon, minden rácspontra egyesével kiszámítva, és minden rácspontra a képen hozzárendeljük a megfelelő értéket.

```
        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;

}
```

Végül pedig egy visszajelzést írunk ki a standard outpura, hogy tudjuk, lefutott a program. Feltűnhet még hogy kiírja a program hogy éppen hol tart a futtatásában, ezt százalékban teszi meg, így láthatjuk ha esetleg nem fut, elakad, vagy valami hiba folytán nem végzdik el a processz, nem kell várnunk a végtelenségig hogy lássuk valami nincs rendben.

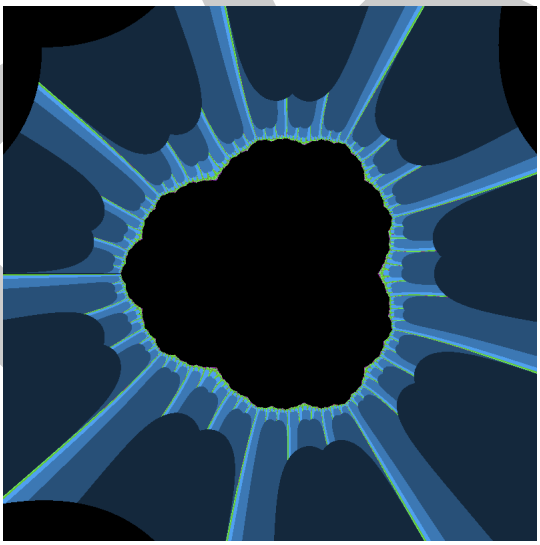
A program futtatása:

```
g++ biomorf.cpp -lpng -O3 -o biomorfok
```

```
./biomorfok bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
```

```
 Szamitas
bmorf.png mentve.
```

A végeredmény pedig így néz ki:



5.3. ábra. Biomorfok

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: `bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu` nevű állománya.

Megfigyelhető volt mind a három előző feladatban, hogy a program lefutási ideje igencsak hosszadalmas. Ez, ha nagyobb felbontású, vagy több képet akarunk készíteni nem tűrhető. Mivel rengeteg számítás megy végbe érthető hogy a lefutási idő nem kevesebb egy másodpercnél, azonban erre van megoldás. Optimalizálni kell a kódot. Bár tapasztalt és hozzáértő programozók megpróbálhatják kézzel is azaz manuálisan optimalizálni a kódot, ez a legtöbb esetben nem szokott előnyös lenni, hisz ez is okot adhat hibára, illetve nem feltétlenül jelenti azt hogy a legjobb mértékben optimalizálta a kódot. Erre használjuk a CUDA párhuzamosítást. A grafikus kártya vezérlő egységét használjuk fel, hiszen a hardver egyébként is sok, nagy erőforrást-igénylő számítások elvégzésére lett kitalálva, így ez nekünk pont kézre esik.

Fontos tudni hogy a program futtatásához szükséges számunka az 'nvidia-cuda-toolkit' telepítése.

```
sudo apt install nvidia-cuda-toolkit
```

A lentebbi kód ismerős lehet, hisz hasonlót használtunk a Mandelbortos c++-os megjlenenítésben.

```
//mandelpngc_60x60_100.cu
#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)
{



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


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

  int iteracio = 0;

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

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

  while (reZ * reZ + imZ * imZ < 4 && iteracio < 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 / 10, MERET / 10);
  dim3 tgrid (10, 10);
  mandelkernel <<< grid, tgrid >>> (device_kepadat);

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

}

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

  clock_t delta = clock ();

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

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

  int kepadat[MERET][MERET];

  cudamandel (kepadat);

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

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

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

  times (&tmsbuf2);
```

```
  std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
    + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

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

}
```

A program futtatása:

```
nvcc mandelpngc_60x60_100.cu -lpng -O3 -o mandelCuda
```

```
./mandelCuda mandelCuda.png
```

```
mandelCuda.png mentve
2
0.027182 sec
```

Láthatjuk hogy nagyságrendekkel gyorsabban fut le a program ha párhuzamosítjuk. Ezzel rengeteg idő megspórolva. Míg az én gépemen az eredeti, nem CUDA-s 17 másodpercig futott, addig ez, a másodperc töredéke alatt. Hogy egészen pontos legyek az én gépemen 566-szor gyorsabban futott le CUDA-val mint nélküle.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

források:

A program futtatásához qt keretrendszer szükséges!

A program három fő részből áll. Ezek a következők: Main Frakablak Frakszal Ezeket külön külön fogom kitárgyalni.

Main

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

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

    FrakAblak w1;
    w1.show();
```

```
    return a.exec();
}
```

Ezen a részben hozzuk létre magát a GUI-t, a keretrendszer adta lehetőségekből.

frakablak

```
#include "frakablak.h"

FrakAblak::FrakAblak(double a, double b, double c, double d,
                     int szelesseg, int iteraciosHatar, QWidget *parent)
                         : QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");

    szamitasFut = true;
    x = y = mx = my = 0;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    magassag = (int)(szelesseg * ((d-c)/(b-a)));

    setFixedSize(QSize(szelesseg, magassag));
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);

    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag,  ↩
       iteraciosHatar, this);
    mandelbrot->start();
}

FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}

void FrakAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);

    }
    qpainter.end();
}

void FrakAblak::mousePressEvent(QMouseEvent* event) {
```

```
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;

    update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event) {


    mx = event->x() - x;
    my = mx;

    update();
}

void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {

    if(szamitasFut)
        return;

    szamitasFut = true;

    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;

    double a = this->a+x*dx;
    double b = this->a+x*dx+mx*dx;
    double c = this->d-y*dy-my*dy;
    double d = this->d-y*dy;

    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;

    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag,  ↩
       iteraciosHatar, this);
    mandelbrot->start();

    update();
}

void FrakAblak::keyPressEvent(QKeyEvent *event)
{

    if(szamitasFut)
```

```
        return;

    if (event->key() == Qt::Key_N)
        iteraciosHatar *= 2;
    szamitasFut = true;

    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag,  ↩
       iteraciosHatar, this);
    mandelbrot->start();

}


void FrakAblak::vissza(int magassag, int *sor, int meret)
{
    for(int i=0; i<meret; ++i) {
        QRgb szin = qRgb(0, 255-sor[i], 0);
        fraktal->setPixel(i, magassag, szin);
    }
    update();
}

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

A program ezen részében találhatjuk meg a fő konstruktort. Itt lehet beállítani a Mandelbrot halmaz intervallumait. Ugyancsak ebben a részben zajlanak az eseménykezelések, egérkattintások stb.

frakszal

```
#include "frakszal.h"

FrakSzal::FrakSzal(double a, double b, double c, double d,
                   int szelesseg, int magassag, int iteraciosHatar,  ↩
                      FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;

    egySor = new int[szelesseg];
```

```
}

FrakSzal::~FrakSzal()
{
    delete[] egySor;
}

void FrakSzal::run()
{

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

    int iteracio = 0;

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

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

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

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

            while(reZ*reZ + imZ*imZ < 4 && iteracio < iteraciosHatar) {


                ujreZ = reZ*reZ - imZ*imZ + reC;
                ujimZ = 2*reZ*imZ + imC;

                reZ = ujreZ;
                imZ = ujimZ;

                ++iteracio;

            }


            iteracio %= 256;


            egySor[k] = iteracio;
        }

        frakAblak->vissza(j, egySor, szelesseg);
    }
    frakAblak->vissza();
```

```
}
```

Az egész-akciók kezelése. Az hogy hogyan értelmezi a program az egérkattintást, az egérrel való húzást, mind itt történik meg.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: https://youtu.be/Ui3B6IJnssY, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518

```
        public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {

    private int x, y;

    private int mx, my;

    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
            int szélesség, int iterációsHatár) {

        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");

        addMouseListener(new java.awt.event.MouseAdapter() {

            public void mousePressed(java.awt.event.MouseEvent m) {

                x = m.getX();
                y = m.getY();
                mx = 0;
                my = 0;
                repaint();
            }
```

Felvesszük a fő változókat, illetve felveszünk egy MouseListenert, ami az egérről való inputokat fogja kezelni.

```
            public void mouseReleased(java.awt.event.MouseEvent m) {
                double dx = (MandelbrotHalmazNagyító.this.b
                        - MandelbrotHalmazNagyító.this.a)
                        /MandelbrotHalmazNagyító.this.szélesség;
                double dy = (MandelbrotHalmazNagyító.this.d
```

```
                        - MandelbrotHalmazNagyító.this.c)
                        /MandelbrotHalmazNagyító.this.magasság;

                new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+ ↩
                    x*dx,
                        MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
                        MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
                        MandelbrotHalmazNagyító.this.d-y*dy,
                        600,
                        MandelbrotHalmazNagyító.this.iterációsHatár);
            }
        });

        addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {

            public void mouseDragged(java.awt.event.MouseEvent m) {

                mx = m.getX() - x;
                my = m.getY() - y;
                repaint();
            }
        });
    }
```

Az egér kezelésének gerince, itt van kezelve az egérrel való húzás is.

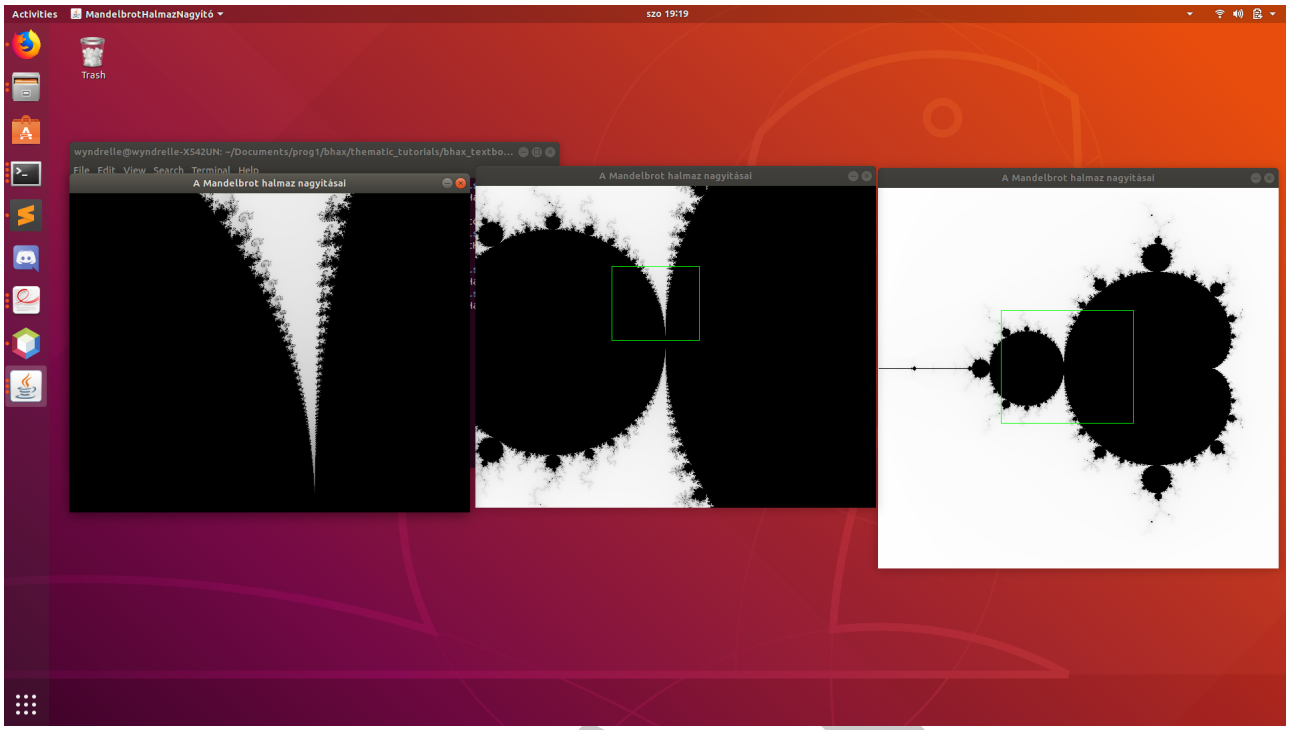
```
        public void pillanatfelvétel() {

        java.awt.image.BufferedImage mentKép =
                new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        java.awt.Graphics g = mentKép.getGraphics();
        g.drawImage(kép, 0, 0, this);
        g.setColor(java.awt.Color.BLUE);
        g.drawString("a=" + a, 10, 15);
        g.drawString("b=" + b, 10, 30);
        g.drawString("c=" + c, 10, 45);
        g.drawString("d=" + d, 10, 60);
        g.drawString("n=" + iterációsHatár, 10, 75);
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
        g.dispose();

        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("MandelbrotHalmazNagyitas_");
```

```
        sb.append(++pillanatfelvételSzámláló);
        sb.append("_");

        sb.append(a);
        sb.append("_");
        sb.append(b);
        sb.append("_");
        sb.append(c);
        sb.append("_");
        sb.append(d);
        sb.append(".png");
```

Itt veszi fel a program az új 'koordinátáit' az újonnan kijelölt területnek, és rendeli hozzá a változókhoz. Itt kap új elnevezést is az itt létrehozott képfájl.

```
         try {
            javax.imageio.ImageIO.write(mentKép, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }

    public void paint(java.awt.Graphics g) {

         g.drawImage(kép, 0, 0, this);

        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }

        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
    }

    public static void main(String[] args) {

        new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```

A Mandelbrot halmaz kiszámítása és rajzolása zajlik ezen a részen.



5.4. ábra. Mandelbrot halmaz nagyító

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.

C++

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

class PolarGen
{

  public:

    PolarGen();
        double next();
        ~PolarGen(){}

  private:
    bool notStored;
    double storedValue;
};
```

Létrehozunk egy PolarGen class-t. Ebben találhatjuk a fő konstruktort, és destroktort, ezen felül láthatjuk hogy két double-t tartalmaz és egy bool. A 'storedValue'-ban van a szám amit raktározunk.

```
PolarGen::PolarGen()
{
  notStored = true;
```

```
  std::srand (std::time(NULL));
};


double PolarGen::next()
{
  if (notStored)
{
    double u1, u2, v1, v2, w;

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

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

    storedValue = r * v2;
    notStored = !notStored;

    return r * v1;
  }

  else
  {
    notStored = !notStored;
    return storedValue;
  }
};
```

Itt történik a random szám generálása különböző műveletekkel

```
int main()
{

  PolarGen rnd;

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

}
```

És kiírunk 10 random számot, amivel megnézhetjük hogy az algoritmusunk működik-e. Mivel ugyanazt az algoritmust használtuk az összes szám generálásához, ha nem ugyanazok akkor feltételezhetően működnek.

Futtatása:

```
    g++ Random.c -o RandomC
```

```
    ./RandomC
-0.224994
0.572746
-0.376624
0.293692
-0.0369429
0.185147
0.00934692
-0.0551615
1.29168
0.531338
```

Ahogy észrevehetjük, mind a 10 szám különböző, tehát az algoritmus működik.

Java

```
                        public class PolarGenerator
{
    boolean notStored = true;
    double storedValue;

    public PolarGenerator()
    {
        notStored = true;
    }
```

A fő classt láthatjuk itt. Fontos megjegyezni hogy a két kód struktúrájában megegyezik. A Fő classban láthatjuk a storedValue-t ami magát az értéket tartalmazza és egy booleant ami jelzi hogy van-e értéke az objektumnak vagy nem. Mikor deklaráljuk az objektumot adunk neki értéket.

```
                                            public double next()
    {
        if(notStored)
        {
            double u1,u2,v1,v2,w;
            do
            {

                u1 = Math.random();
                u2 = Math.random();
                v1 = 2 * u1 - 1;
                v2 = 2 * u2 -1;
```

```
                w = v1*v1 + v2*v2;

            } while(w>1);

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

Itt történik meg a random szám generálása, ugyanolyan műveletekkel mint a c++-os változatában.Itt használjuk a Java Math.random() függvényét is, így biztosan random számokat fogunk kapni.

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

Végül pedig kiírunk a standard outputra 10 számot.

A program futtatása:

```
                java PolarGenerator
-1.042235345103159
0.27841844673218763
-0.38977233542862694
-1.065713636736024
-1.4692309400600783
-0.26718802269506126
-1.6687565664903936
0.6424669277862723
-0.042469546910690825
0.17745378636975095
```

Láthatjuk hogy a 10 szám random, így sikeresen teljesítettük a feladatot.

6.2. LZW

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

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

typedef struct binfa
{
  int ertek;
  struct binfa *bal_nulla;
  struct binfa *jobb_egy;

} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
  BINFA_PTR p;

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

extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);
```

Csinálunk egy új struktúrát, meny tartalmaz egy integer alapú értéket és két pointert, mely a két alatta lévő gyermekére mutat, jobb és bal oldali gyermkére. Ezen felül csinálunk neki konstruktórt és inicializáljuk a destruktort is, emellett pedig a kiir függvény is inicializálásra kerül. Ez a struktúra pointer alapú lesz.

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

```
  BINFA_PTR gyoker = uj_elem ();
  gyoker->ertek = '/';
  BINFA_PTR fa = gyoker;

  while (read (0, (void *) &b, 1))
    {
      write (1, &b, 1);
      if (b == '0')
  {
    if (fa->bal_nulla == NULL)
      {
        fa->bal_nulla = uj_elem ();
        fa->bal_nulla->ertek = 0;
        fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
        fa = gyoker;
      }
    else
      {
        fa = fa->bal_nulla;
      }
  }
      else
  {
    if (fa->jobb_egy == NULL)
      {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
      }
    else
      {
        fa = fa->jobb_egy;
      }
  }
    }

  printf ("\n");
  kiir (gyoker);
  extern int max_melyseg;
  printf ("melyseg=%d", max_melyseg);
  szabadit (gyoker);
}
```

A main függvénybe lesznek megírva a főbb dolgok. Itt kerül megírásra az is, hogy mi alapján rendeli hozzá melyik értéket melyik elem gyermekéhez.

```
            static int melyseg = 0;
```

```
int max_melyseg = 0;

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

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

A fő függvények, metódusok pedig ide kerülnek, köztük a destruktor is.

Futtatása:

```
               gcc LZWBinfa.c -o LZWc
```

```
               ./LZWc <in.txt
az Egyszer volt hol nem volt, volt egyszer egy magányos txt file.
Léte abba merült ki hogy gazdája random programok parancssori argumentumába ↩
    írta bele.

Ez valóban egy igencsak magányos és szomorú lét, hogy boldog volt-e? nem

ha azonban lefordulne ez a cucc annak nagyon örülnék.

-----------------------------------------------------------------------1(24) ↩
```

```
------------------------------------------------------------------1(23)
-----------------------------------------------------------------1(22)
----------------------------------------------------------------1(21)
---------------------------------------------------------------1(20)
--------------------------------------------------------------1(19)
-------------------------------------------------------------1(18)
------------------------------------------------------------1(17)
-----------------------------------------------------------1(16)
----------------------------------------------------------1(15)
---------------------------------------------------------1(14)
--------------------------------------------------------1(13)
-------------------------------------------------------1(12)
------------------------------------------------------1(11)
-----------------------------------------------------1(10)
----------------------------------------------------1(9)
---------------------------------------------------1(8)
--------------------------------------------------1(7)
-------------------------------------------------1(6)
------------------------------------------------1(5)
-----------------------------------------------1(4)
----------------------------------------------1(3)
---------------------------------------------1(2)
--------------------------------------------/(1)
```

6.3. Fabejárás

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

A binfa egy olyan adatszerkezet, melynek minden elemének van két mutatója, mely két másik különböző elemére mutat, gyermekére.

Ez a feladat igencsak egyszerű ha tudjuk mi az az inorder és postorderi bejárás. Inorderi bejárás az, amikor először a bal oldali elemet írom ki, utána a gyökeret, és végül a jobb oldali elemet. Postorderi bejárás amikor először a bal, majd jobb és végül az gyökér kerül kiírásra. Tehát valahogy így fog kinézni a kiir() függvény az előző példából:

Inorder:

```
        void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
  max_melyseg = melyseg;

             kiir (elem->bal_nulla); // Bal olal
```

```
            printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem ↩
               ->ertek, // gyökér
        melyseg);
      kiir (elem->jobb_egy); //jobb oldal


      for (int i = 0; i < melyseg; ++i)
  printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
         ,
        melyseg);

      --melyseg;
    }
}
```

Postorder:

```
        void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
  max_melyseg = melyseg;

            kiir (elem->bal_nulla); // Bal olal
              kiir (elem->jobb_egy); //jobb oldal

            printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem ↩
               ->ertek, // gyökér
        melyseg);



      for (int i = 0; i < melyseg; ++i)
  printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
         ,
        melyseg);

      --melyseg;
    }
}
```

6.4. Tag a gyökér

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

A binfa alapja ugyanaz mint az előző feladatban, viszont most hogy adott a c++ lehetőség, kicsit egyszerűbben is meg lehet írni a kódot.

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



class LZWBinFa
{
public:

    LZWBinFa (): fa(&gyoker) {}
```

Tag a gyökér!

```
    void operator<<(char b)
    {

        if (b == '0')
        {

            if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
            {

                Csomopont *uj = new Csomopont ('0');

                fa->ujNullasGyermek (uj);

                fa = &gyoker;
            }
            else
            {

                fa = fa->nullasGyermek ();
            }
        }

        else
        {
            if (!fa->egyesGyermek ())
            {
                Csomopont *uj = new Csomopont ('1');
```

```
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }
```

Operátor túlterhelés, ez majd akkor fog kelleni mikor új elemet kívánunk betenni a fába. Illetve akkor lesz hasznosítva.

```
    void kiir (void)
    {

        melyseg = 0;

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

    }
```

Destruktor. Letörli az egész fát.

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


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

private:
    class Csomopont
    {
    public:
```

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

        Csomopont *nullasGyermek () const {
            return balNulla;
        }

        Csomopont *egyesGyermek () const {
            return jobbEgy;
        }

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

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

        char getBetu() const {
            return betu;
        }

    private:

        char betu;

        Csomopont *balNulla;
        Csomopont *jobbEgy;

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

Csomopont class, az függvényeive és metódusaival.

```
    Csomopont *fa;

    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;

    LZWBinFa (const LZWBinFa &);
    LZWBinFa & operator=(const LZWBinFa &);


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

```
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ↩
           leállítása
        if (elem != NULL)
        {
            ++melyseg;
            kiir (elem->egyesGyermek(), os);
            // ez a postorder bejáráshoz képest
            // 1-el nagyobb mélység, ezért -1
            for (int i = 0; i < melyseg; ++i)
                os << "---";
            os << elem->getBetu() << "(" << melyseg - 1 << ")" << std::endl ↩
               ;
            kiir (elem->nullasGyermek(), os);
            --melyseg;
        }
    }
    void szabadit (Csomopont * elem)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ↩
           leállítása
        if (elem != NULL)
        {
            szabadit (elem->egyesGyermek());
            szabadit (elem->nullasGyermek());
            // ha a csomópont mindkét gyermekét felszabadítottuk
            // azután szabadítjuk magát a csomópontot:
            delete elem;
        }
    }
```

Ez a metódus fog lezajlani, mikor ki kívánjuk írni a fa elemeit.

```
protected:



    Csomopont gyoker;
    int maxMelyseg;
    double atlag, szoras;

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

};

int LZWBinFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (&gyoker);
```

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

    rszoras (&gyoker);

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

    return szoras;
}
void LZWBinFa::rmelyseg (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek());
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        rmelyseg (elem->nullasGyermek());
        --melyseg;
    }
}
void
LZWBinFa::ratlag (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyesGyermek());
        ratlag (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)
        {
            ++atlagdb;
```

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




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

A program használata.

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



    if (argc != 4) {

        usage();

        return -1;
    }


    char *inFile = *++argv;


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

```
}
```

Emlékeztető ha rosszul használtuk

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

unsigned char b; // ide olvassik majd a bejövő fájl bájtjait
LZWBinFa binFa; // s nyomjuk majd be az LZW fa objektumunkba

while (beFile.read ((char *) &b, sizeof (unsigned char))) {

    for (int i = 0; i < 8; ++i)
    {
    // maszkolunk
        int egy_e = b & 0x80;
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if  ↩
       megmondja melyik:
        if ((egy_e >> 7) == 1)
    // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW fa  ↩
       objektumunkba
            binFa << '1';
        else
    // különben meg a '0' betűt:
            binFa << '0';
        b <<= 1;
    }

}

//std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi  ↩
   verziókban de, hogy izgalmasabb legyen
// a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

kiFile << binFa; // ehhez kell a globális operator<< túlterhelése, lásd ↩
    fentebb
// (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, ↩
    mégis megy, hurrá)

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

binFa.szabadit ();

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

return 0;
```

```
}
```

6.5. Mutató a gyökér

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

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


class LZWBinFa {
public:

     LZWBinFa (){
    std::cout << "LZWBinFa konstruktor" << std::endl;

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

          std::cout << "LZWBinFa destruktor" << std::endl;
          szabadit( gyoker);
     }

     LZWBinFa ( const LZWBinFa & regi ) {
          std::cout << "LZWBinFa másoló konstruktor" << std::endl;

          gyoker = masol(regi.gyoker,regi.fa);
     }

     LZWBinFa ( LZWBinFa && regi ) {
          std::cout << "LZWBinFa mozgató konstruktor" << std::endl;

          gyoker = nullptr;
          *this = std::move (regi);

     }
     LZWBinFa & operator= ( LZWBinFa && regi)
     {
     std::cout << "LZWBinFa mozgató értékadás" << std::endl;
```

```
    std::swap (gyoker, regi.gyoker);

    return *this;

}

LZWBinFa & operator= ( LZWBinFa & regi)
    {
  Csomopont * ujgyoker = masol(regi.gyoker, regi.fa);


  szabadit (gyoker);
  gyoker = ujgyoker;
  return* this;
}
```

Konstruktor, Destruktor, mozgató konstruktor, másoló konstruktor, mozgató és másoló értékadás. Ezek találhatóak a kód ezen szegmensében.

```
    LZWBinFa& operator<< ( char b ) {

        if ( b == '0' ) {

            if ( !fa->nullasGyermek () ) {

                Csomopont *uj = new Csomopont ( '0' );

                fa->ujNullasGyermek ( uj );

                fa = gyoker;
            } else {

                fa = fa->nullasGyermek ();
            }
        }

        else {
            if ( !fa->egyesGyermek () ) {
                Csomopont *uj = new Csomopont ( '1' );
                fa->ujEgyesGyermek ( uj );
                fa = gyoker;
            } else {
                fa = fa->egyesGyermek ();
            }
        }
    }
```

Operátor túlterhelés. Így egyszerűbben lehet megírni, ha be akarunk írni valamit a fába.

```
    void kiir ( void ) {

        melyseg = 0;

        kiir ( gyoker, std::cout );
    }

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

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

Kiírása a fának. Láthatjuk hogy ez is operátor túlterheléssel lett megoldva, így ami az operátor bal oldalára kerül kapja meg a kiírást.

```
private:
    class Csomopont {
    public:

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

        Csomopont *nullasGyermek () const {
            return balNulla;
        }

        Csomopont *egyesGyermek () const {
            return jobbEgy;
        }

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

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

        char getBetu () const {
            return betu;
        }

    private:

        char betu;

        Csomopont *balNulla;
        Csomopont *jobbEgy;

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


    };
```

Csomópont osztály, a fontos függvényeivel.

```
    Csomopont *fa;

    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;

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

        if ( elem != NULL ) {
            ++melyseg;
            kiir ( elem->egyesGyermek (), os );

            for ( int i = 0; i < melyseg; ++i )
                os << "---";
            os << elem->getBetu () << "(" << melyseg - 1 << ")" << std:: ↩
               endl;
            kiir ( elem->nullasGyermek (), os );
            --melyseg;
        }
    }
```

Itt történik a kiírás, vagyis hogy hogyan fog valójában kiíródni a fa, vagyis a fa elemei.

```
    void szabadit ( Csomopont * elem ) {

        if ( elem != NULL ) {
```

```
            szabadit ( elem->egyesGyermek () );
            szabadit ( elem->nullasGyermek () );

            delete elem;
        }
    }
```

Destruktor

```
    Csomopont *  masol ( Csomopont * elem, Csomopont * regifa ) {

        Csomopont * ujelem = NULL;

        if ( elem != NULL ) {
            ujelem = new Csomopont ( elem->getBetu() );

            ujelem->ujEgyesGyermek ( masol ( elem->egyesGyermek (),  ↩
               regifa ) );
            ujelem->ujNullasGyermek ( masol ( elem->nullasGyermek (),  ↩
               regifa ) );

            if ( regifa == elem )
              fa = ujelem;

        }

        return ujelem;
    }
```

Másoló függvény, mely a teljes fát átmásolja egy másikba.

```
protected:

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

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

};



int
LZWBinFa::getMelyseg ( void )
```

```
{
    melyseg = maxMelyseg = 0;
    rmelyseg ( gyoker );
     return maxMelyseg - 1;
}

double
LZWBinFa::getAtlag ( void )
{
     melyseg = atlagosszeg = atlagdb = 0;
     ratlag ( gyoker );
     atlag = ( ( double ) atlagosszeg ) / atlagdb;
     return atlag;
}

double
LZWBinFa::getSzoras ( void )
{
     atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras ( gyoker );

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

    return szoras;
}

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

void
LZWBinFa::ratlag ( Csomopont * elem )
{
     if ( elem != NULL ) {
    ++melyseg;
```

```
    ratlag ( elem->egyesGyermek () );
    ratlag ( elem->nullasGyermek () );
    --melyseg;
    if ( elem->egyesGyermek () == NULL && elem->nullasGyermek () == NULL  ↩
       ) {
        ++atlagdb;
        atlagosszeg += melyseg;
    }
     }
}

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

Néhány jól kinéző művelet, melyeknek neve beszédes.

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


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

     if ( argc != 4 ) {

    usage ();

    return -1;
     }


     char *inFile = *++argv;
```

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

Ha rosszul lett használva a fa akkor egy használati útmutatást kap a felhasználó.

```
    std::fstream beFile ( inFile, std::ios_base::in );

    if ( !beFile ) {
   std::cout << inFile << " nem letezik..." << std::endl;
   usage ();
   return -3;
    }

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

    unsigned char b;
    LZWBinFa binFa;


    while (beFile.read ((char *) &b, sizeof (unsigned char)))
   {

for (int i = 0; i < 8; ++i)
{
    if (b & 0x80)
        binFa << '1';
    else
        binFa << '0';
    b <<= 1;
}

   }

    kiFile << binFa;


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




LZWBinFa binFa2, binFa4;
```

```
    binFa2 << '1' << '1' << '1' << '1' << '1' << '1' << '1' << '1' << '1'  ↩
       << '1';

    binFa4 << '0' << '0' << '0' << '0' << '0' << '0';
    std::cout << "egy"<< binFa2 << "egy" << std::endl;
    std::cout << "ketto" << binFa4 << "ketto" <<std::endl;

        binFa4 = binFa2;

    std::cout << "egy"<< binFa2 << "egy" << std::endl;
    std::cout << "ketto" << binFa4 << "ketto" <<std::endl;




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


       return 0;
}
```

A végén pedig néhány művelet hogy csináljunk is valamit a megírt függvényekkel, classokkal és egyebekkel.

A program futtatása:

```
            g++ LZWBinFa.cpp -o LZ
```

```
            ./LZ in.txt -o out.txt
LZWBinFa konstruktor
LZWBinFa konstruktor
LZWBinFa konstruktor
egy---------------1(4)
------------1(3)
---------1(2)
------1(1)
---/(0)
egy
ketto---/(0)
------0(1)
---------0(2)
------------0(3)
ketto
egy---------------1(4)
------------1(3)
---------1(2)
------1(1)
---/(0)
egy
```

```
ketto--------------1(4)
------------1(3)
---------1(2)
------1(1)
---/(0)
ketto
LZWBinFa destruktor
LZWBinFa destruktor
LZWBinFa destruktor
```

```
             cat out.txt
------------------1(5)
---------------1(4)
----------------------1(6)
------------------0(5)
----------------------0(6)
------------------------0(7)
------------1(3)
                 .
                 .
                 .
                 .

------------------0(5)
--------------------------1(8)
---------------------------------1(10)
----------------------------0(9)
------------------------1(7)
----------------------0(6)
depth = 11
mean = 8.17213
var = 1.11112
```

6.6. Mozgató szemantika

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

Mozgató értékadás:

```
             LZWBinFa & operator= ( LZWBinFa && regi)
    {
    std::cout << "LZWBinFa mozgató értékadás" << std::endl;
    std::swap (gyoker, regi.gyoker);
```

```
    return *this;

}
```

Megcseréli a két pointer (vagyis ahová mutatnak). Mivel ez a mozgató konstruktorból fog meghívódni, így az egyik pointer nullpointer lesz, így technikailag az egyik objektum elveszíti az elemeit.

Mozgató konstruktor:

```
         LZWBinFa ( LZWBinFa && regi ) {
        std::cout << "LZWBinFa mozgató konstruktor" << std::endl;

        gyoker = nullptr;
        *this = std::move (regi);

    }
```

Létrehoz egy új objektumot, melynek a gyoker mutatója null-ra fog mutatni, majd megcseréli egy, már létező objektummal, azzal hogy meghívja a mozgató értékadást.

A mozgató szemantika átmásolja egy új objektumba az eredeti objektum elemeit, majd (miközben) kitörli az összeset az eredetiből.

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:

A feladat arról szól, hogy a hangyák mozgását, viselkedését szimuláljuk le a qt vizuális környezetében. Fontos kiemelni hogy a hangyák feromon alapján tájékozódnak, tehát ha egy hangya talál valamit, akkor az útvonalát feromonnal jelöli ezzel jelezve a társainak hogy milyen irányba kell tartaniuk ha meg akarják találni azt a bizonyos talált dolgot.

A kódnak három fő fájlja van: antthread.cpp antwin.cpp main.cpp Ezeket egyesével fogom kitárgyalni röviden, mivel maga a kód nagyon hosszú és aránylag bonyolult is.

main.cpp

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

#include "antwin.h"

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

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

```
{

    QApplication a ( argc, argv );

    QCommandLineOption szeles_opt ( {"w","szelesseg"}, "Oszlopok (cellakban ↩
       ) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( {"m","magassag"}, "Sorok (cellakban)  ↩
       szama.", "magassag", "150" );
    QCommandLineOption hangyaszam_opt ( {"n","hangyaszam"}, "Hangyak szama. ↩
       ", "hangyaszam", "100" );
    QCommandLineOption sebesseg_opt ( {"t","sebesseg"}, "2 lepes kozotti  ↩
       ido (millisec-ben).", "sebesseg", "100" );
    QCommandLineOption parolgas_opt ( {"p","parolgas"}, "A parolgas erteke. ↩
       ", "parolgas", "8" );
    QCommandLineOption feromon_opt ( {"f","feromon"}, "A hagyott nyom  ↩
       erteke.", "feromon", "11" );
    QCommandLineOption szomszed_opt ( {"s","szomszed"}, "A hagyott nyom  ↩
       erteke a szomszedokban.", "szomszed", "3" );
    QCommandLineOption alapertek_opt ( {"d","alapertek"}, "Indulo ertek a  ↩
       cellakban.", "alapertek", "1" );
    QCommandLineOption maxcella_opt ( {"a","maxcella"}, "Cella max erteke." ↩
       , "maxcella", "50" );
    QCommandLineOption mincella_opt ( {"i","mincella"}, "Cella min erteke." ↩
       , "mincella", "2" );
    QCommandLineOption cellamerete_opt ( {"c","cellameret"}, "Hany hangya  ↩
       fer egy cellaba.", "cellameret", "4" );
    QCommandLineParser parser;

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

    parser.process ( a );

    QString szeles = parser.value ( szeles_opt );
    QString magas = parser.value ( magas_opt );
    QString n = parser.value ( hangyaszam_opt );
    QString t = parser.value ( sebesseg_opt );
    QString parolgas = parser.value ( parolgas_opt );
    QString feromon = parser.value ( feromon_opt );
```

```
    QString szomszed = parser.value ( szomszed_opt );
    QString alapertek = parser.value ( alapertek_opt );
    QString maxcella = parser.value ( maxcella_opt );
    QString mincella = parser.value ( mincella_opt );
    QString cellameret = parser.value ( cellamerete_opt );

    qsrand ( QDateTime::currentMSecsSinceEpoch() );

    AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), feromon ↩
       .toInt(), szomszed.toInt(), parolgas.toInt(),
                  alapertek.toInt(), mincella.toInt(), maxcella.toInt(),
                  cellameret.toInt() );

    w.show();

    return a.exec();
}
```

Ez a main, itt történnek meg a hívások, illetve ez a program gerince. Innen hívódik meg az összes többi osztály, melyben egyes dolgok vannak leszimulálva. Továbbá itt lesznek megadva az alapértékek is.

antwin.cpp

```
        #include "antwin.h"
#include <QDebug>

AntWin::AntWin ( int width, int height, int delay, int numAnts,
                 int pheromone, int nbhPheromon, int evaporation, int  ↩
                    cellDef,
                 int min, int max, int cellAntMax, QWidget *parent ) :  ↩
                    QMainWindow ( parent )
{
    setWindowTitle ( "Ant Simulation" );

    this->width = width;
    this->height = height;
    this->max = max;
    this->min = min;

    cellWidth = 6;
    cellHeight = 6;

    setFixedSize ( QSize ( width*cellWidth, height*cellHeight ) );

    grids = new int**[2];
    grids[0] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[0][i] = new int [width];
    }
```

```
    grids[1] = new int*[height];
    for ( int i=0; i<height; ++i ) {
        grids[1][i] = new int [width];
    }

    gridIdx = 0;
    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j ) {
            grid[i][j] = cellDef;
        }

    ants = new Ants();

    antThread = new AntThread ( ants, grids, width, height, delay, numAnts, ↩
        pheromone,
                                nbhPheromon, evaporation, min, max,  ↩
                                   cellAntMax);

    connect ( antThread, SIGNAL ( step ( int) ),
              this, SLOT ( step ( int) ) );

    antThread->start();

}

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

    grid = grids[gridIdx];

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

            double rel = 255.0/max;

            qpainter.fillRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight,
                                QColor ( 255 - grid[i][j]*rel,
                                         255,
                                         255 - grid[i][j]*rel) );

            if ( grid[i][j] != min )
            {
                qpainter.setPen (
                    QPen (
                        QColor ( 255 - grid[i][j]*rel,
                                 255 - grid[i][j]*rel, 255),
```

```
                        1 )
                );

                qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                    cellWidth, cellHeight );
            }



            qpainter.setPen (
                QPen (
                    QColor (0,0,0 ),
                    1 )
            );

            qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight );

        }
    }

    for ( auto h: *ants) {
        qpainter.setPen ( QPen ( Qt::black, 1 ) );

        qpainter.drawRect ( h.x*cellWidth+1, h.y*cellHeight+1,
                            cellWidth-2, cellHeight-2 );

    }

    qpainter.end();
}

AntWin::~AntWin()
{
    delete antThread;

    for ( int i=0; i<height; ++i ) {
        delete[] grids[0][i];
        delete[] grids[1][i];
    }

    delete[] grids[0];
    delete[] grids[1];
    delete[] grids;

    delete ants;
}

void AntWin::step ( const int &gridIdx )
{
```

```
    this->gridIdx = gridIdx;
    update();
}
```

Az antwin.cpp fájl kezeli a pályát, a rácsot. Ahogy észrevehetjük, töröl készít és mozgat hangyákat, emellett pedig itt történik meg a rács megrajzolása minden iterációban.

antthread.cpp

```
        #include "antthread.h"
#include <QDebug>
#include <cmath>
#include <QDateTime>

AntThread::AntThread ( Ants* ants, int*** grids,
                       int width, int height,
                       int delay, int numAnts,
                       int pheromone, int nbrPheromone,
                       int evaporation,
                       int min, int max, int cellAntMax)
{
    this->ants = ants;
    this->grids = grids;
    this->width = width;
    this->height = height;
    this->delay = delay;
    this->pheromone = pheromone;
    this->evaporation = evaporation;
    this->min = min;
    this->max = max;
    this->cellAntMax = cellAntMax;
    this->nbrPheromone = nbrPheromone;

    numAntsinCells = new int*[height];
    for ( int i=0; i<height; ++i ) {
        numAntsinCells[i] = new int [width];
    }

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j ) {
            numAntsinCells[i][j] = 0;
        }

    qsrand ( QDateTime::currentMSecsSinceEpoch() );

    Ant h {0, 0};
    for ( int i {0}; i<numAnts; ++i ) {

        h.y = height/2 + qrand() % 40-20;
        h.x = width/2 + qrand() % 40-20;
```

```
        ++numAntsinCells[h.y][h.x];

        ants->push_back ( h );

    }

    gridIdx = 0;
}

double AntThread::sumNbhs ( int **grid, int row, int col, int dir )
{
    double sum = 0.0;

    int ifrom, ito;
    int jfrom, jto;

    detDirs ( dir, ifrom, ito, jfrom, jto );

    for ( int i=ifrom; i<ito; ++i )
        for ( int j=jfrom; j<jto; ++j )

            if ( ! ( ( i==0 ) && ( j==0 ) ) ) {
                int o = col + j;
                if ( o < 0 ) {
                    o = width-1;
                } else if ( o >= width ) {
                    o = 0;
                }

                int s = row + i;
                if ( s < 0 ) {
                    s = height-1;
                } else if ( s >= height ) {
                    s = 0;
                }

                sum += (grid[s][o]+1)*(grid[s][o]+1)*(grid[s][o]+1);

            }

    return sum;
}

int AntThread::newDir ( int sor, int oszlop, int vsor, int voszlop )
{

    if ( vsor == 0 && sor == height -1 ) {
        if ( voszlop < oszlop ) {
            return 5;
```

```
        } else if ( voszlop > oszlop ) {
            return 3;
        } else {
            return 4;
        }
    } else if ( vsor == height - 1 && sor == 0 ) {
        if ( voszlop < oszlop ) {
            return 7;
        } else if ( voszlop > oszlop ) {
            return 1;
        } else {
            return 0;
        }
    } else if ( voszlop == 0 && oszlop == width - 1 ) {
        if ( vsor < sor ) {
            return 1;
        } else if ( vsor > sor ) {
            return 3;
        } else {
            return 2;
        }
    } else if ( voszlop == width && oszlop == 0 ) {
        if ( vsor < sor ) {
            return 7;
        } else if ( vsor > sor ) {
            return 5;
        } else {
            return 6;
        }
    } else if ( vsor < sor && voszlop < oszlop ) {
        return 7;
    } else if ( vsor < sor && voszlop == oszlop ) {
        return 0;
    } else if ( vsor < sor && voszlop > oszlop ) {
        return 1;
    }

    else if ( vsor > sor && voszlop < oszlop ) {
        return 5;
    } else if ( vsor > sor && voszlop == oszlop ) {
        return 4;
    } else if ( vsor > sor && voszlop > oszlop ) {
        return 3;
    }

    else if ( vsor == sor && voszlop < oszlop ) {
        return 6;
    } else if ( vsor == sor && voszlop > oszlop ) {
        return 2;
    }
```

```
    else { //(vsor == sor && voszlop == oszlop)
        qDebug() << "ZAVAR AZ EROBEN az iranynal";

        return -1;
    }

}

void AntThread::detDirs ( int dir, int& ifrom, int& ito, int& jfrom, int&  ↩
   jto )
{

    switch ( dir ) {
    case 0:
        ifrom = -1;
        ito = 0;
        jfrom = -1;
        jto = 2;
        break;
    case 1:
        ifrom = -1;
        ito = 1;
        jfrom = 0;
        jto = 2;
        break;
    case 2:
        ifrom = -1;
        ito = 2;
        jfrom = 1;
        jto = 2;
        break;
    case 3:
        ifrom =
            0;
        ito = 2;
        jfrom = 0;
        jto = 2;
        break;
    case 4:
        ifrom = 1;
        ito = 2;
        jfrom = -1;
        jto = 2;
        break;
    case 5:
        ifrom = 0;
        ito = 2;
        jfrom = -1;
        jto = 1;
```

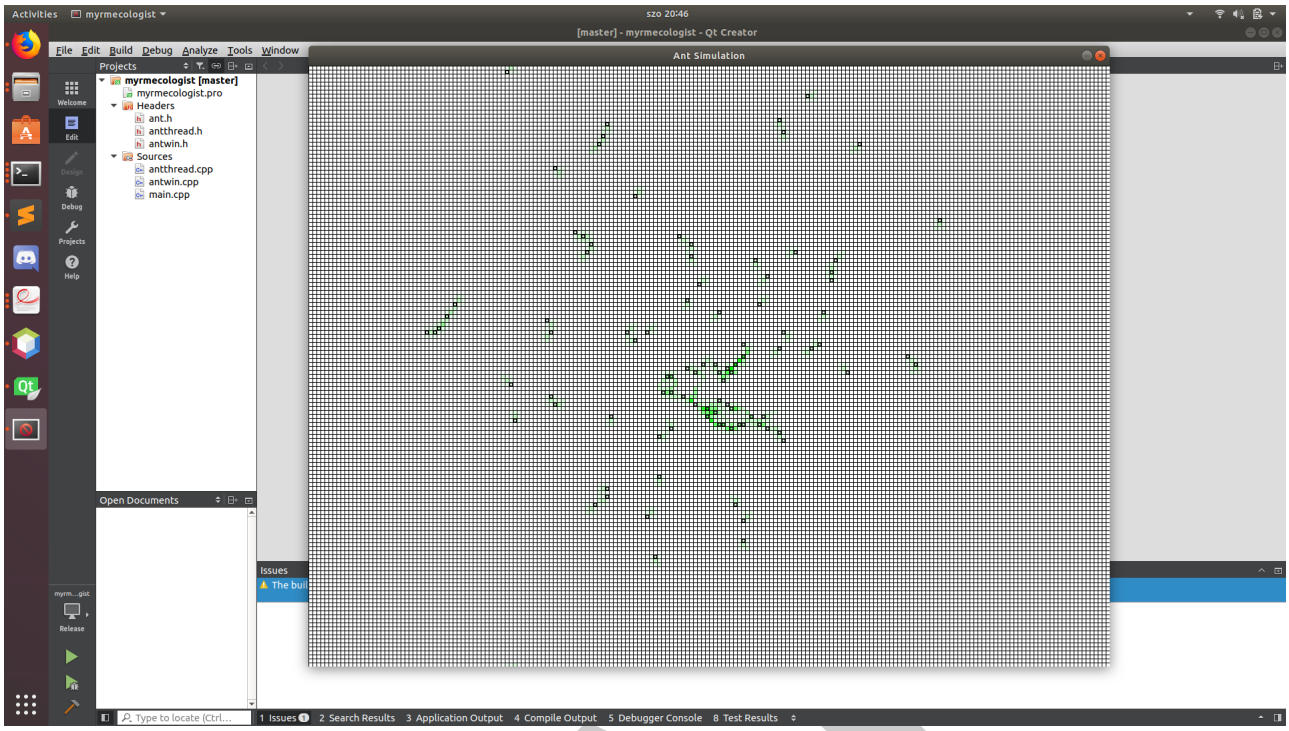
```
        break;
    case 6:
        ifrom = -1;
        ito = 2;
        jfrom = -1;
        jto = 0;
        break;
    case 7:
        ifrom = -1;
        ito = 1;
        jfrom = -1;
        jto = 1;
        break;

    }

}

int AntThread::moveAnts ( int **racs,
                          int sor, int oszlop,
                          int& vsor, int& voszlop, int dir )
{

    int y = sor;
    int x = oszlop;

    int ifrom, ito;
    int jfrom, jto;

    detDirs ( dir, ifrom, ito, jfrom, jto );

    double osszes = sumNbhs ( racs, sor, oszlop, dir );
    double random = ( double ) ( qrand() %1000000 ) / ( double ) 1000000.0;
    double gvalseg = 0.0;


    for ( int i=ifrom; i<ito; ++i )
        for ( int j=jfrom; j<jto; ++j )
            if ( ! ( ( i==0 ) && ( j==0 ) ) )
            {
                int o = oszlop + j;
                if ( o < 0 ) {
                    o = width-1;
                } else if ( o >= width ) {
                    o = 0;
                }

                int s = sor + i;
                if ( s < 0 ) {
                    s = height-1;
```

```
                } else if ( s >= height ) {
                    s = 0;
                }

                //double kedvezo = std::sqrt((double)(racs[s][o]+2));//( ↩
                   racs[s][o]+2)*(racs[s][o]+2);
                //double kedvezo = (racs[s][o]+b)*(racs[s][o]+b);
                //double kedvezo = ( racs[s][o]+1 );
                double kedvezo = (racs[s][o]+1)*(racs[s][o]+1)*(racs[s][o ↩
                   ]+1);

                double valseg = kedvezo/osszes;
                gvalseg += valseg;

                if ( gvalseg >= random ) {

                    vsor = s;
                    voszlop = o;

                    return newDir ( sor, oszlop, vsor, voszlop );

                }

            }

    qDebug() << "ZAVAR AZ EROBEN a lepesnel";
    vsor = y;
    voszlop = x;

    return dir;
}

void AntThread::timeDevel()
{

    int **racsElotte = grids[gridIdx];
    int **racsUtana = grids[ ( gridIdx+1 ) %2];

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j )
        {
            racsUtana[i][j] = racsElotte[i][j];

            if ( racsUtana[i][j] - evaporation >= 0 ) {
                racsUtana[i][j] -= evaporation;
            } else {
                racsUtana[i][j] = 0;
            }

        }
```

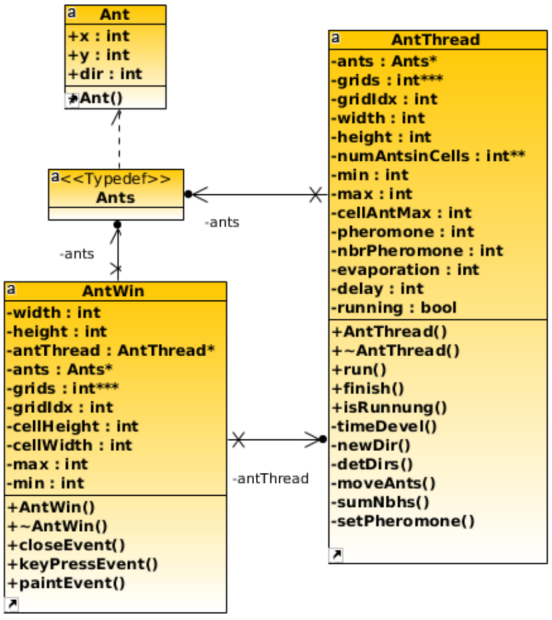
```
    for ( Ant &h: *ants )
    {

        int sor {-1}, oszlop {-1};
        int ujirany = moveAnts( racsElotte, h.y, h.x, sor, oszlop, h.dir );

        setPheromone ( racsUtana, h.y, h.x );

        if ( numAntsinCells[sor][oszlop] <cellAntMax ) {

            --numAntsinCells[h.y][h.x];
            ++numAntsinCells[sor][oszlop];

            h.x = oszlop;
            h.y = sor;
            h.dir = ujirany;

        }
    }

    gridIdx = ( gridIdx+1 ) %2;
}



void AntThread::setPheromone ( int **racs,
                        int sor, int oszlop )
{

    for ( int i=-1; i<2; ++i )
        for ( int j=-1; j<2; ++j )
            if ( ! ( ( i==0 ) && ( j==0 ) ) )
            {
                int o = oszlop + j;
                {
                    if ( o < 0 ) {
                        o = width-1;
                    } else if ( o >= width ) {
                        o = 0;
                    }
                }
                int s = sor + i;
                {
                    if ( s < 0 ) {
                        s = height-1;
                    } else if ( s >= height ) {
                        s = 0;
                    }
                }
```

```
                if ( racs[s][o] + nbrPheromone <= max ) {
                    racs[s][o] += nbrPheromone;
                } else {
                    racs[s][o] = max;
                }


            }

    if ( racs[sor][oszlop] + pheromone <= max ) {
        racs[sor][oszlop] += pheromone;
    } else {
        racs[sor][oszlop]  = max;
    }

}

void AntThread::run()
{
    running = true;
    while ( running ) {

        QThread::msleep ( delay );

        if ( !paused ) {
            timeDevel();
        }

        emit step ( gridIdx );

    }

}

AntThread::~AntThread()
{
    for ( int i=0; i<height; ++i ) {
        delete [] numAntsinCells[i];
    }

    delete [] numAntsinCells;
}
```

Ebben a fájlban van kezelve a hangyakolónia. A hangyák egyesével és összesítve. Mozgásuk, viselkedésük, cselekedeteik itt vannak megírva. Továbbá az is itt van megírva, hogy mi számít a hangyának kedvezőnek, vagy éppen kedvezőtlennek.



7.1. ábra. Hangyák



7.2. ábra. UML

7.2. Java életjáték

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

Az életjáték egy rácsban játszódik, melyben minden cellának két állapota lehet: élő vagy halott. A szabályok alapján a rácsok állapota minden iterációban változik. Érdekessége a játéknak, hogy bár a szabályai egyszerűek, mégis igencsak komplex dolgok tudnak előállni még a legvéletlenszerűbb leosztásból is. Véleményem szerint ez egy szép allegória, a 'minden komplex dolog valójában csak egyszerű szabályok ismétlődése'.

A Game of Life-nak három szabája van: - A sejt életben marad ha 2 vagy 3 szomszédja van. - A sejt elpusztul ha 3-nál több vagy 2-nél kevesebb szomszédja van. - Ha egy cellának pontosan 3 élő szomszédja van, életre kell. A következő Java kód ezek alapján a szabályok alapján lett megírva.

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

Deklaráljuk a fő változókat, illetve objektumokat, melyeket később fogunk használni.

```
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);
        }
    });
```

Megcsináljuk a rácsot.

```
        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();
            }
        });
```

Készítünk egy keylistenert mely a billentyűzetről jött inputot kezeli.

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

készítünk egy Mouselistenert, mely az egérről jött inputot kezeli.

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

```
            }
        });
```

A mouseMotionListener az egér mozgását, húzást kezeli.

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

        setTitle("Sejtautomata");
        setResizable(false);
        setSize(szélesség*cellaSzélesség,
                magasság*cellaMagasság);
        setVisible(true);
        new Thread(this).start();
    }
```

Megcsináljuk az ablakot.

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

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

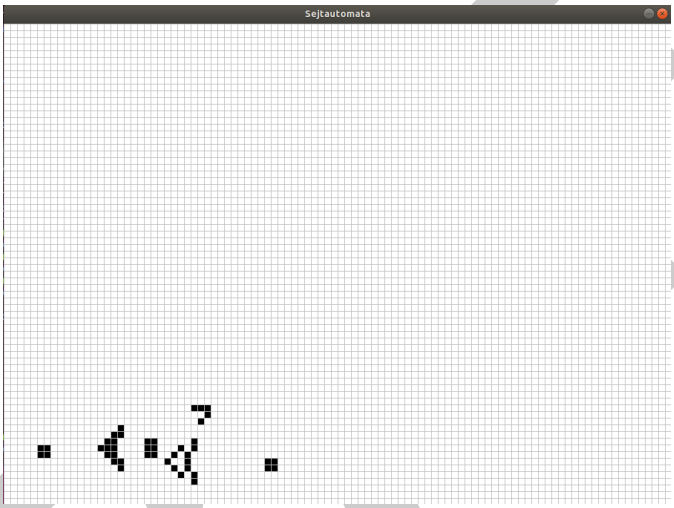
```
                magasság*cellaMagasság)));
        }
    }

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

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

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

        return állapotúSzomszéd;
    }

    public void időFejlődés() {

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

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

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

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

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

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

```
                        rácsUtána[i][j] = HALOTT;
                }
            }
        }
        rácsIndex = (rácsIndex+1)%2;
    }

    public void run() {

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

            időFejlődés();
            repaint();
        }
    }

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

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

    }

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

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

        rács[y+ 3][x+ 13] = ÉLŐ;

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

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

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

```
        rács[y+ 6][x+ 23] = ÉLŐ;
        rács[y+ 6][x+ 24] = ÉLŐ;
        rács[y+ 6][x+ 25] = ÉLŐ;

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

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

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

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

        rács[y+ 11][x+ 25] = ÉLŐ;

    }

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

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

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

A kód utolsó részében pedig megadunk néhány Élő cellát ami azért fog kezeskedni hogy a siklókilövő meg legyen alkotva.



7.3. ábra. Életjáték, siklókilövő

7.3. Qt C++ életjáték

Most Qt C++-ban!

Az életjáték szabályai nem változnak ebben a feladatban (nyilvánvalóan) azonban ezúttal qt keretrendszerben lesz megírva a kód c++ nyelven.

Három fő fájl tartozik ehhez a programhoz: main.cpp sejtablak.cpp sejtszal.cpp

main.cpp

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

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

```
  QApplication a(argc, argv);
  SejtAblak w(100, 75);
  w.show();

  return a.exec();
}
```

A main-ben történik meg a meghívás, innen indul a program. Ahogy láthatjuk itt fog meghívódni a sejtablak, illetve itt fogja végül 'megmutatni' az ablakot a program.

sejtablak.cpp

```
            #include "sejtablak.h"

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;

    //siklo(racs, 2, 2);

    sikloKilovo(racs, 5, 60);

  eletjatek = new SejtSzal(racsok, szelesseg, magassag, 120, this);
```

```
  eletjatek->start();

}

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

  s
  bool **racs = racsok[racsIndex];
  // racsot rajzoljuk ki:
  for(int i=0; i<magassag; ++i) {
    for(int j=0; j<szelesseg; ++j) {

      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();
}


SejtAblak::~SejtAblak()
{
  delete eletjatek;

  for(int i=0; i<magassag; ++i) {
    delete[] racsok[0][i];
    delete[] racsok[1][i];
  }

  delete[] racsok[0];
  delete[] racsok[1];
  delete[] racsok;


}

void SejtAblak::vissza(int racsIndex)
{
  this->racsIndex = racsIndex;
  update();
```

```
}


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

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

}

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;

}
```

A program ezen része kezeli a rácsot. Itt fog kirajzolódni a virtuális megjelenítése az életjátéknak. Láthatjuk hogy néhány előre beállított rács élőre van állítva az inicializáláskor. Mint az előző feladatban, itt is ugyanúgy minden egyes rácspontot külön megvizsgálunk, megnézve hogy a következő iterációban milyen státuszban lesz-e és ha ez megvan, krajzoljuk.

sejtszal.cpp

```
            #include "sejtszal.h"

SejtSzal::SejtSzal(bool ***racsok, int szelesseg, int magassag, int  ↩
   varakozas, SejtAblak *sejtAblak)
{
    this->racsok = racsok;
    this->szelesseg = szelesseg;
    this->magassag = magassag;
    this->varakozas = varakozas;
    this->sejtAblak = sejtAblak;

    racsIndex = 0;
}


int SejtSzal::szomszedokSzama(bool **racs,
                              int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;

    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
```

```
            if(!((i==0) && (j==0))) {

        int o = oszlop + j;
        if(o < 0)
            o = szelesseg-1;
        else if(o >= szelesseg)
            o = 0;

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

        if(racs[s][o] == allapot)
            ++allapotuSzomszed;
    }

    return allapotuSzomszed;
}


void SejtSzal::idoFejlodes() {

    bool **racsElotte = racsok[racsIndex];
    bool **racsUtana = racsok[(racsIndex+1)%2];

    for(int i=0; i<magassag; ++i) { // sorok
        for(int j=0; j<szelesseg; ++j) { // oszlopok

            int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);

            if(racsElotte[i][j] == SejtAblak::ELO) {

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

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

```
void SejtSzal::run()
{
    while(true) {
        QThread::msleep(varakozas);
        idoFejlodes();
        sejtAblak->vissza(racsIndex);
    }

}

SejtSzal::~SejtSzal()
{
}
```

Többek között megtalálható egy konstruktor illetve az ellenőrzés, melyet minden ráncspontra megtesz a program minden iterációban, megnézve, hogy a következő iterációban milyen állapota lesz az adott rácsnak.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

A BarinB benchmark lényege, hogy felmérje az emberek mentális képességeit azzal, hogy 10 percen keresztül egy bizonyos entitáson tartják az egerüket, miközben a program újabb és újabb zavaró tényezőt bocsát ki rájuk.

A program ismételten 3 fő részből áll. Ezek a következők: - main.cpp - BrainBThread.cpp - BrainBWin.cpp

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();
}
```

Mint minden más c++ programban, itt is a main függvény az ami lefut. Itt elsősorban számtalan fontos információ közöl velünk a program, amit jó észben tartani, majd elkezdi meghívni az metódusokat, classokat, objektumokat stb, ami végül magát a programot fogja eredményezni. Legalábbis minden azt ami a BrainB benchmarkot jelképezni.

BrainBThread.cpp

```
#include "BrainBThread.h"
```

```
BrainBThread::BrainBThread ( int w, int h )
{

        dispShift = heroRectSize+heroRectSize/2;

        this->w = w - 3 * heroRectSize;
        this->h = h - 3 * heroRectSize;

        std::srand ( std::time ( 0 ) );

        Hero me ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) -  ↩
           100,
                this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) -  ↩
                   100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 9 );

        Hero other1 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
           ) - 100,
                    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
                       ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ↩
                        5, "Norbi Entropy" );
        Hero other2 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
           ) - 100,
                    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
                       ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ↩
                        3, "Greta Entropy" );
        Hero other4 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
           ) - 100,
                    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
                       ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ↩
                        5, "Nandi Entropy" );
        Hero other5 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
           ) - 100,
                    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0  ↩
                       ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ↩
                        7, "Matyi Entropy" );

        heroes.push_back ( me );
        heroes.push_back ( other1 );
        heroes.push_back ( other2 );
        heroes.push_back ( other4 );
        heroes.push_back ( other5 );

}

BrainBThread::~BrainBThread()
{

}
```

```
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;

}
```

Ez a része a programnak többek között méri az időt, mely igencsak fontos, hisz 10 perc áll rendelkezésre a játékban.

BrainBWin.cpp

```
#include "BrainBWin.h"
```

```
const QString BrainBWin::appName = "NEMESPOR BrainB Test";
const QString BrainBWin::appVersion = "6.0.3";

BrainBWin::BrainBWin ( int w, int h, QWidget *parent ) : QMainWindow (  ↩
   parent )
{

//    setWindowTitle(appName + " " + appVersion);
//    setFixedSize(QSize(w, h));

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

        brainBThread = new BrainBThread ( w, h - yshift );
        brainBThread->start();

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

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

}

void BrainBWin::endAndStats ( const int &t )
{

        qDebug()  << "\n\n\n";
        qDebug()  << "Thank you for using " + appName;
        qDebug()  << "The result can be found in the directory " + statDir;
        qDebug()  << "\n\n\n";

        save ( t );
        close();
}

void BrainBWin::updateHeroes ( const QImage &image, const int &x, const int ↩
    &y )
{

        if ( start && !brainBThread->get_paused() ) {

                int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) +  ↩
                   ( this->mouse_y - y ) * ( this->mouse_y - y );

                if ( dist > 121 ) {
                        ++nofLost;
```

```
                        nofFound = 0;
                        if ( nofLost > 12 ) {

                                if ( state == found && firstLost ) {
                                        found2lost.push_back ( brainBThread ↩
                                           ->get_bps() );
                                }

                                firstLost = true;

                                state = lost;
                                nofLost = 0;
                                //qDebug() << "LOST";
                                //double mean = brainBThread->meanLost();
                                //qDebug() << mean;

                                brainBThread->decComp();
                        }
                } else {
                        ++nofFound;
                        nofLost = 0;
                        if ( nofFound > 12 ) {

                                if ( state == lost && firstLost ) {
                                        lost2found.push_back ( brainBThread ↩
                                           ->get_bps() );
                                }

                                state = found;
                                nofFound = 0;
                                //qDebug() << "FOUND";
                                //double mean = brainBThread->meanFound();
                                //qDebug() << mean;

                                brainBThread->incComp();
                        }

                }

        }
        pixmap = QPixmap::fromImage ( image );
        update();
}

void BrainBWin::paintEvent ( QPaintEvent * )
{
        if ( pixmap.isNull() ) {
                return;
        }
```

```
        QPainter qpainter ( this );

        xs = ( qpainter.device()->width() - pixmap.width() ) /2;
        ys = ( qpainter.device()->height() - pixmap.height() +yshift ) /2;

        qpainter.drawPixmap ( xs, ys, pixmap );

        qpainter.drawText ( 10, 20, "Press and hold the mouse button on the ↩
            center of Samu Entropy" );

        int time = brainBThread->getT();
        int min, sec;
        millis2minsec ( time, min, sec );
        QString timestr = QString::number ( min ) + ":" + QString::number ( ↩
            sec ) + "/10:0";
        qpainter.drawText ( 10, 40, timestr );

        int bps = brainBThread->get_bps();
        QString bpsstr = QString::number ( bps ) + " bps";
        qpainter.drawText ( 110, 40, bpsstr );

        if ( brainBThread->get_paused() ) {
                QString pausedstr = "PAUSED (" + QString::number (  ↩
                   brainBThread->get_nofPaused() ) + ")";

                qpainter.drawText ( 210, 40, pausedstr );
        }

        qpainter.end();
}

void BrainBWin::mousePressEvent ( QMouseEvent *event )
{

        brainBThread->set_paused ( false );

}

void BrainBWin::mouseReleaseEvent ( QMouseEvent *event )
{

        //brainBThread->set_paused(true);

}

void BrainBWin::mouseMoveEvent ( QMouseEvent *event )
{

        start = true;
```

```
        mouse_x = event->pos().x() -xs - 60;
        //mouse_y = event->pos().y() - yshift - 60;
        mouse_y = event->pos().y() - ys - 60;
}

void BrainBWin::keyPressEvent ( QKeyEvent *event )
{

        if ( event->key() == Qt::Key_S ) {
                save ( brainBThread->getT() );
        } else if ( event->key() == Qt::Key_P ) {
                brainBThread->pause();
        } else if ( event->key() == Qt::Key_Q ||  event->key() == Qt:: ↩
           Key_Escape ) {
                close();
        }

}

BrainBWin::~BrainBWin()
{

}
```

Ez a fáj többek között kezeli az egér bemenetét, a pozícióját, illetve hogy leköveti-e 'Samut'. Ezen felül különféle eseményeket kezel le, illetve kiírja hogy hol találhatóak meg a mérés eredményei.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

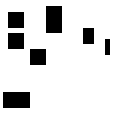
Megoldás forrása: https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0 (/tensorflow-0.9.0/tensorflow/exa
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Sajnos azt kell hogy bevalljam hogy életemben nem használtam/írtam/olvastam Python kódot, gépi tanulásról pedig még csak történeteket hallottam, illetve vázlatosan ismerem az elméleti hátterét. Ettől függetlenül valamit fogok ide produkálni mert nincs szándékomban megbukni csak azért mert unreasonable feladatokat kell megoldani hétről hétre, olyan feladatokat amiket még nem vettünk, olyan nyelven amit még nem használtunk.

Tehát akkor...

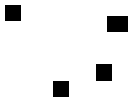
Az MNIST egy adatbázis, melynek tartalma kézzel írt számok amerikai emberek álltal.Minden szám egy 28x28 pixeles fekete fehér kép. Hogy mi mit fogunk ezzel az egésszel csinálni? Regenerálunk egy csomó 28x28 pixeles képet, melyek fehér alapon fekete pontok, majd megnézetjük hogy a számítógéppel hogy mit vél kivenni ezekből a pontokból. Röviden ennyi. Hogy ezt hogy fogjuk megcsinálni? Tanárúr gitlabjára segítségképpen feltöltött programját fogjuk használni SMNIST

Ezzel a programmal generáltam 100 képfájlt Íme ezekből három darab:



8.1. ábra. 1

8.2. ábra. 2



8.3. ábra. 3

Itt pedig a Python kód szerepel:

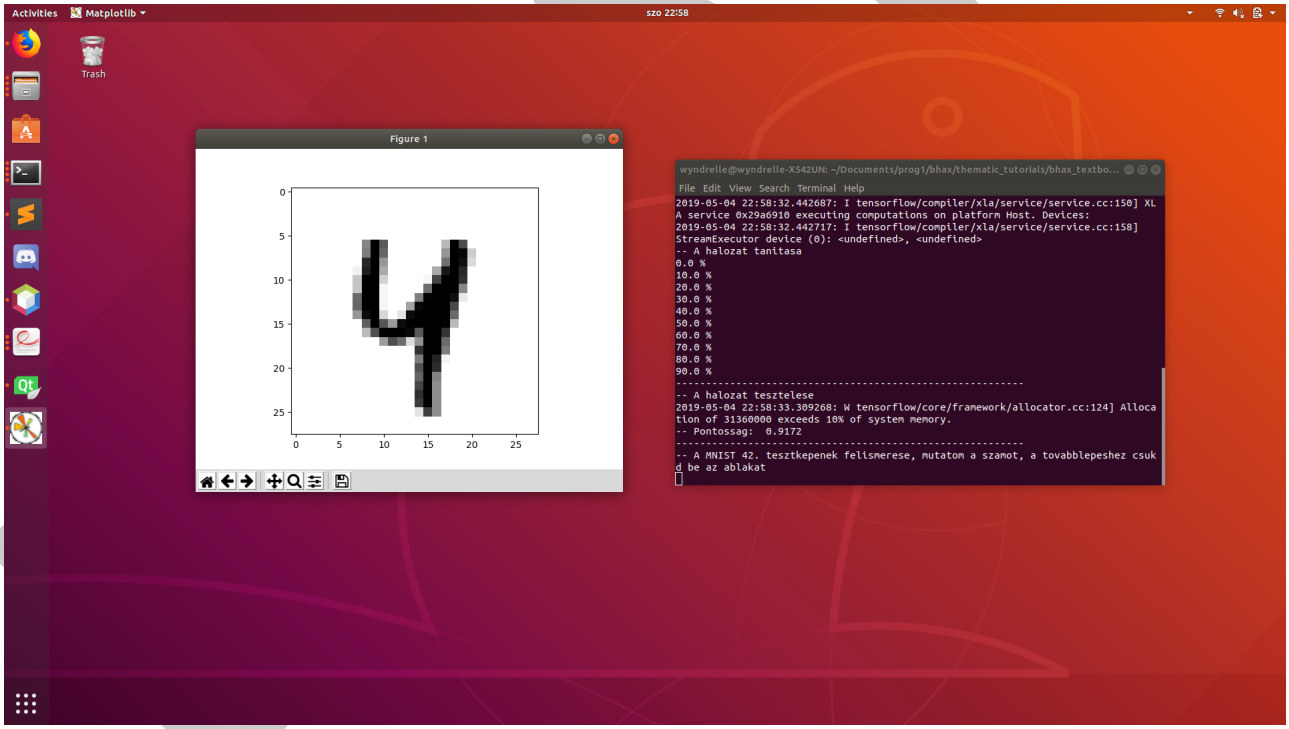
```
            # Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#  ↩
   ==============================================================================


# Norbert Batfai, 27 Nov 2016
# Some modifications and additions to the original code:
# https://github.com/tensorflow/tensorflow/blob/r0.11/tensorflow/examples/ ↩
   tutorials/mnist/mnist_softmax.py
# See also http://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol
#  ↩
   ==============================================================================


"""A very simple MNIST classifier.

See extensive documentation at
http://tensorflow.org/tutorials/mnist/beginners/index.md
```

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

import argparse

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

import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

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)

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

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

  # The raw formulation of cross-entropy,
  #
  #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
  #                                 reduction_indices=[1]))
  #
  # can be numerically unstable.
  #
  # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
  # outputs of 'y', and then average across the batch.
  cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits( ↩
     labels = y_, logits = y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize( ↩
     cross_entropy)
```

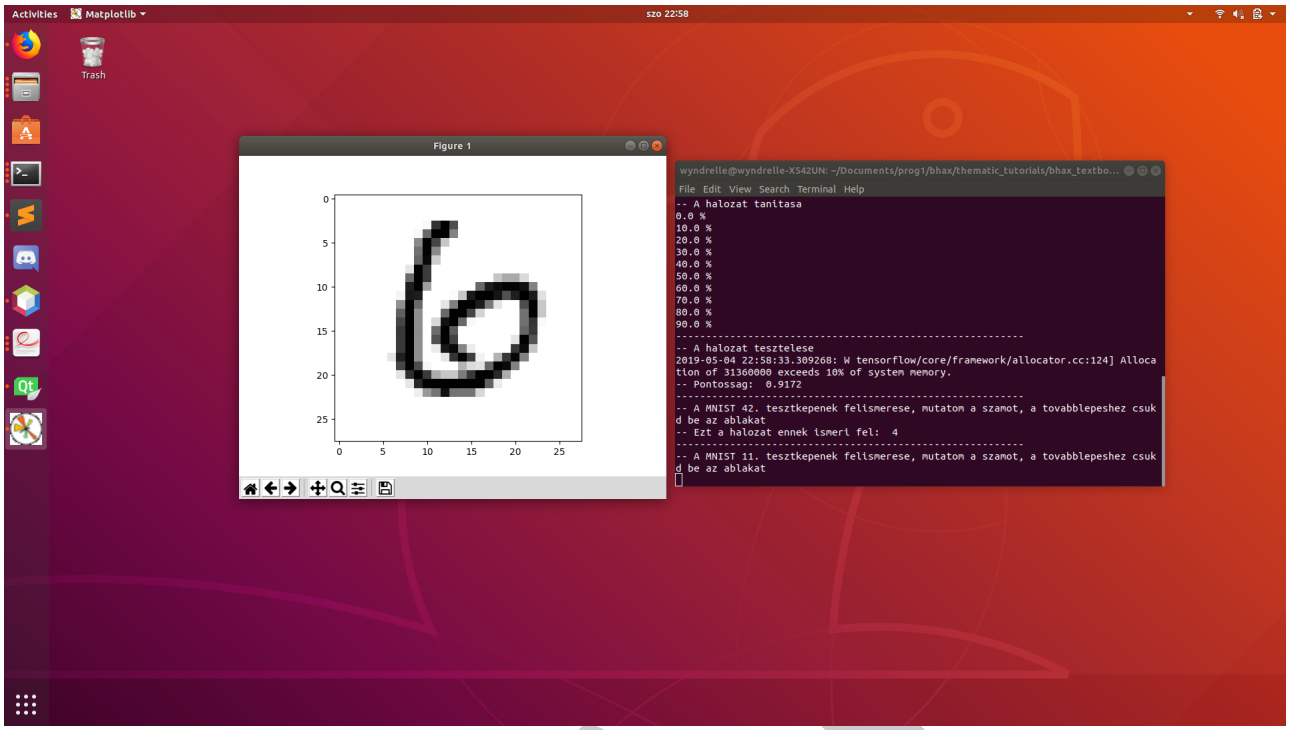
```
  sess = tf.InteractiveSession()
  # Train
  tf.initialize_all_variables().run(session=sess)
  print("-- A halozat tanitasa")
  for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
      print(i/10,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

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

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

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

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

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

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

  #print("-- A sajat kezi 8-asom felismerese, mutatom a szamot, a  ↩
     tovabblepeshez csukd be az ablakat")
  print("-- A MNIST 11. tesztkepenek felismerese, mutatom a szamot, a  ↩
     tovabblepeshez csukd be az ablakat")
#  img = readimg()
#  image = img.eval()
#  image = image.reshape(28*28)
  img = mnist.test.images[11]
  image = img
  matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm. ↩
     binary)
  matplotlib.pyplot.savefig("8.png")
  matplotlib.pyplot.show()
```

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

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

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

Sajnos már említettem hogy nem értek Python kódhoz egyáltalán, úgyhogy elég valószínű hogy téves információt közölnék ha megpróbálnám darabjaira szedni a forráskódot amitől szeretnék távol maradni. Ettől függetlenül lefuttattam a kódot (miközben a hajam nagyrészét kitéptem).



8.4. ábra. Láthatjuk hogy felismerte a négyes számot



8.5. ábra. Felismerte a hatos számot is

8.2. Mély MNIST

[SKIP] - 3

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

[SKIP] - 4

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

List Processing - Listafeldolgozás Röviden LISP. Ez egy igen régi nyelv, kialakulása egészen 1950-ig vezethező vissza, azonban mindettől függetlenül igen széleskörben használt egészen a mai napig. Érdekessége hogy dialektusok nem infix hanem prefix alakban dolgoznak. Bár ez talán elsőre kényelmetlennek hangozhat, valójába mindössze arról van szó, hogy (x+y) helyett +(x,y)-nal kell dolgoznunk. Ezen feladatban a Scipt-fu dialektust fogjuk használni.

Fontos kiemelni hogy ebben a dialektusban nem állnak rendelkezésünkre olyan kifejezések mint például 'for' 'while' vagy 'switch'. Azonban rendelkezésünkre áll az 'if', amit ki is fogunk használni a következő kis programban:

```
            (define (faktorialis n)
        (if (= n 1)
                1
                (* n (faktorialis (- n 1) ))
        ))

(display (faktorialis 3))
```

Ez a rekurzív megoldása a feladatnak. Úgy gondolom a legegyszerűbb ha sorról sorra haladunk végig

```
            (define (faktorialis n)
```

A 'define' az eljárást létrehozó kulcsszó. 'faktorialis n' egy egy elemű allista, melynek eleme lesz a paraméter.

```
            (if (= n 1)
                1
```

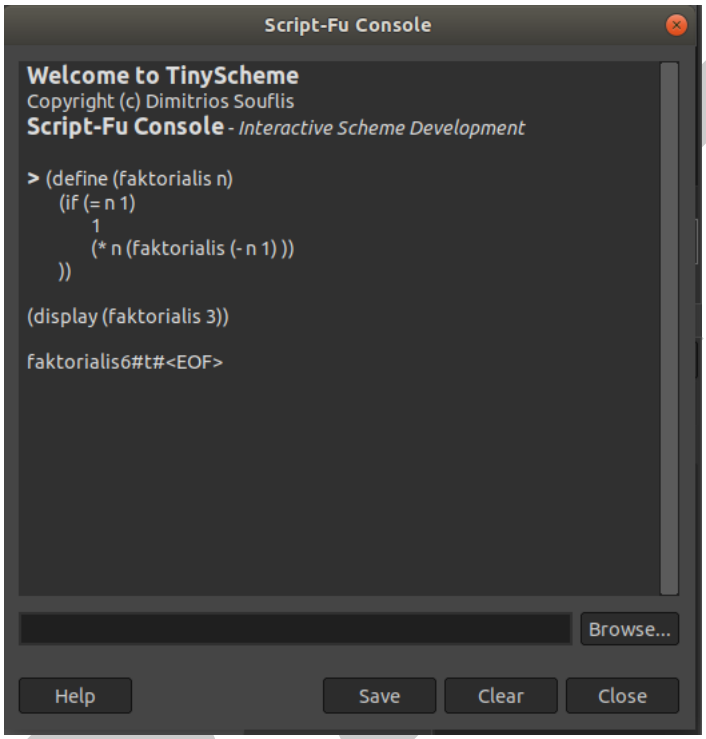
Egy egyszerű if statement. Lefordítva: 'Ha n egyenlő 1-gyel' akkor a következő történik: Visszaad egyet (vagy az eljárás értéke 1)

```
            (* n (faktorialis (- n 1) ))
```

Ha pedig nem, akkor megszorozzuk az ’n’ változót a rekurzívan újrahívott fakgorialis eljárásunkkal, azonban most n-1 paraméterekkel.

```
            (display (faktorialis 3))
```

Végül pedig megnézzük működik-e a program azzal hogy kiíratjuk vele a 3 faktoriálisát.

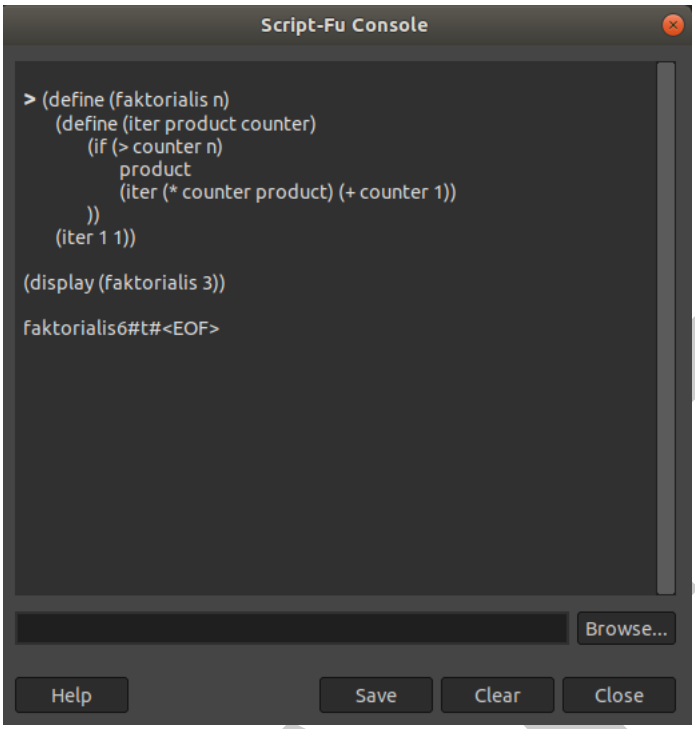


9.1. ábra. Rekurzív

Most pedig ugyanezt Iteratívan

```
            (define (faktorialis n)
        (define (iter product counter)
                (if (> counter n)
                        product
                        (iter (* counter product) (+ counter 1))
                ))
        (iter 1 1))

(display (faktorialis 3))
```

Míg az előző ’3-tól számolt lefele’, ez ’1-től fog 3-ig’. Ami történik, hogy paraméterként adunk meg egyet, ami addig lesz megszorozva a nála egyel nagyobbik számmal amíg el nem éri a hármat, vagyis azt a számot aminek meg akartuk kapni a faktoriálisát.



9.2. ábra. Iteratív

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

Megoldás forrása:

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

Ebben a feladatban egy Script-fu szkriptet fogunk írni ami 'krómosítja' a szöveget. Erre a Tanárúr által felkínált, már megírt programot fogjuk használni. Szerencsére minden GIMP függvény és asset a rendelkezésünkre áll ebben a feladatban.

```
          (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)
)
```

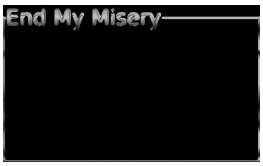
Tömbökbe vesszük a fő színeket, amikre szükségünk lesz. Ezek kellenek majd a krómhatás eléréséhez.

```
     (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. ábra.



9.4. ábra.

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 mandala az indiai hitvallás szerint az isten ábrázolása. A következő szkriptel fogjuk előállítani:

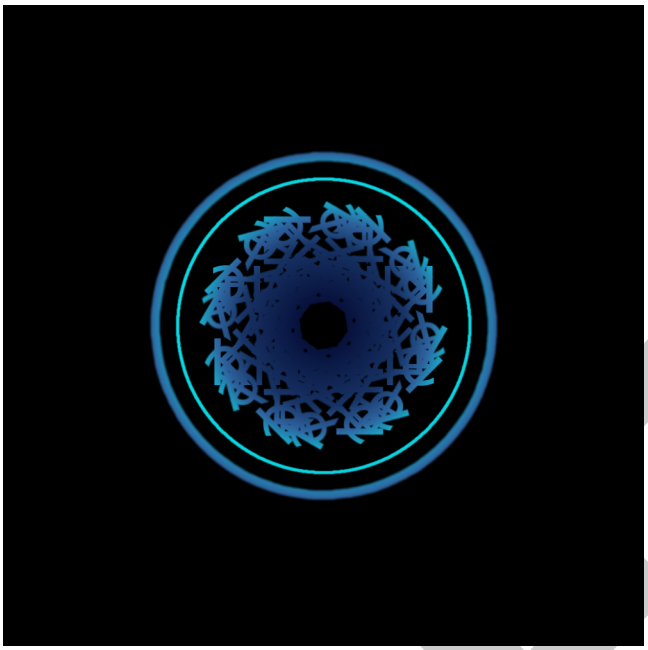
```
                (define (elem x lista)

    (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)))
```

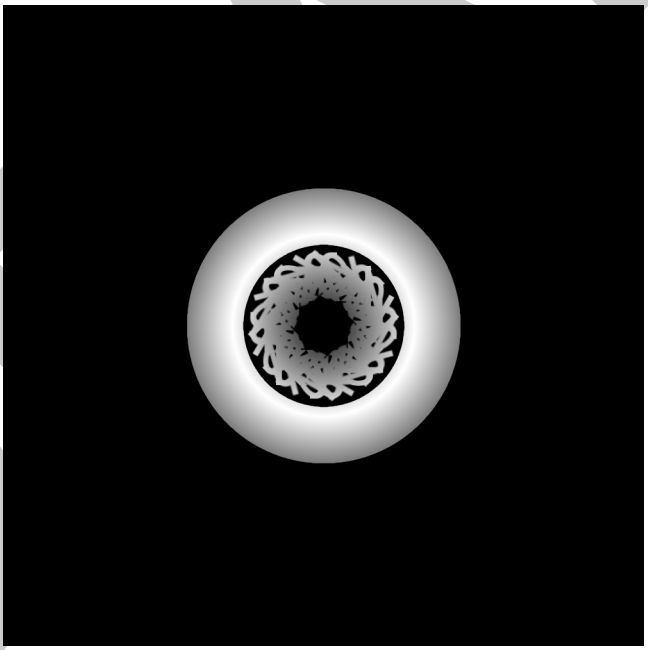
```
  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
  (set! textfs (car(gimp-image-merge-down image text-layer  ↩
     CLIP-TO-BOTTOM-LAYER)))

  (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"
)
```

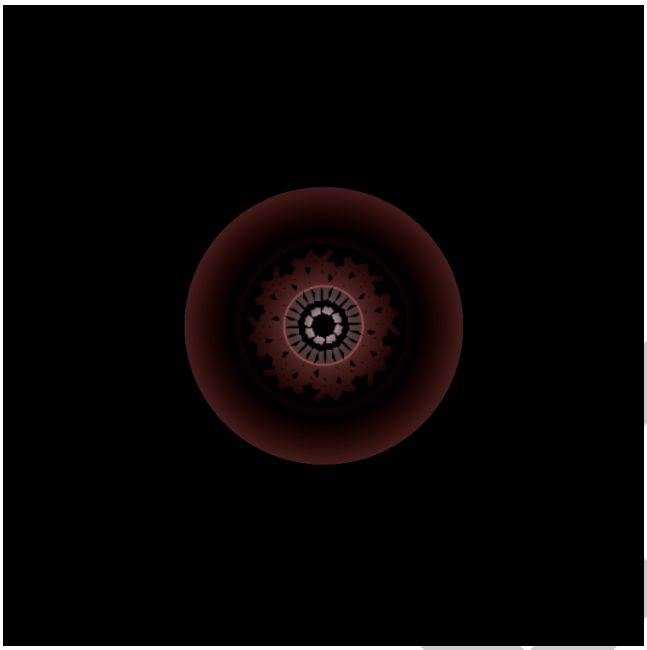
A szkriptel készített mandalák:



9.5. ábra.



9.6. ábra.



9.7. ábra.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

Ezen fejezetben tárgyalásra került a gépeken kialakult nyelvek szintjei, a gépi nyelv, assembly szintű nyelv, és a magas szintű nyelv. Ezen felül szó esett, hogy ezen szintek A Magas Szintű Programozási Nyelvek 1 és Magas Szintű Programozási Nyelvek 2 filozófiájával foglalkoznak.

Imsertetve lett a forrásprogram vagy forrásszöveg miléte. Leírásra került a forrásszöveg "nyelvtana", illetve hogy a magas szintű programozási nyelvek valójában csak átfordítják gépi nyelvre a forrásszöveget hiszen minden proesszor gépi nyelven dolgozik.

A fordítóprogram egy speciális program mely magas szintá programozási nyelvet konvertál gépi nyelvvé. Szó esett hogy az imént említett fordítóprogram a forrásszöveget egy egységként kezeli, és hogy milyen lépéseken megyek keresztül mikorontájt a fordítás megtörténik.

Lexikális egységekre darabolja, szintaktikai elemzést hajt végre, szemantikai elemzést hajt végre, illetve ha minden a helyén van, akkor legenerálja a gépinyelvű kódot.

10.2. Programozás bevezetés

Ezen fejezetben szó esett a C nyelv történetéről, illetve hogy közel áll történetileg az UNIX operációs rendszerhez. Ennek ellenére a nyelv nem kötődik operációs rendszerhez vagy géphez.

Szó esett hogy milyen sokoldalú a C nyelv, és hogy elsősorban milyen jól alkalmazható operációs rendszerek írására. Említésre kerülthogy a C nyelv viszonylag alacsony szintá nyelv, mivel főleg karakterekkel számokkal és címekkel dolgozik.

Nem tartozik hozzá objektum illetve karakterlánc, viszont nagyobb szabadságot nyújt a programozónak, hisz saját maga rendelkezhet a számítógép adta lehetőségekkel.

Nincsenek a C nyelvben read és write utasítások. A C nyelv hatékony arra is, hogy hardverfüggetlenül lehessen használni, ezzel szabadságot és sokszínűséget adva a nyelvhez.

A C nyelv tartalmaz if, for, while és switch utasításokat is. A C nyelv elérhetővé teszi a mutatók használatát és címaritmetikáját.

10.3. Programozás

A könyv első részében arról volt szó, hogy a C++ mennyiben másabb a C-nyelvhez képest, mik az újdonságok, illetve kitér az két nyelv objektumorientált különbségeire is.

Szó esett arról hogy az üres paraméterlista azt jelenti hogy nincs a függvénynek paramétere, nem pedig azt hogy tetszőleges számú paraméter hívható.

Említésre került hogy a main függvénybe lehet parancssori argumentumokat adni, illetve hogy a return 0 használata nem kötelező.

Megemlítették hogy az új bool típus használata átláthatóbb kódot eredményez. C++-ban mindenhol állhat deklaráció ahol utasítás is.

Több ugyanolyan függvényt is létrehozhatunk amiknek ugyanaz a neve, ameddig különbözik az argumentumlistája, így a függvénylétrehozás egyszerűbbé válik. Ez a függvénytúlterhelés. Megadhatunk alapértelmezett függvény argumentumokat is.

III. rész

Második felvonás

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

11. fejezet

Helló, Berners-Lee!

11.1. C++ és Java

Ezen feladatban a két, igencsak széleskörben használt és jól ismert nyelvet, a C++-t és a Java-t fogom összemérni, elemezni, Nyékné Dr. Gaizler Judit Java 2 útikalauz programozóknak, illetve Benedek Zoltán és Levendovszky Tihamér Szoftverfejlesztés C++ nyelven könyvei alapján.

A Java teljesen objektumorientált. Ezek a kezdőgondolatai Nyékné Dr. Gaizler Juditnak. De mit jelent ez egy átlag olvasónak akinek semmi köze a programozáshoz? Nos, úgy gondolom hogy annyiról már az átlag ember is tud, hogy a számítástechnikában egyesekkel és nullásokkal dolgoznak, amik információt reprezentálnak. Nos ez a programozásban úgy valósul meg, hogy ezekkez az információkkal dolgozunk a memóriában főleg. Mit jelent ez? Tekintsük azt, hogy egy ember életkorát akarjuk leprezentálni egy X változóval. Ha ennek a változónak értéket adunk, azt a számot, a változó nevével és címével együtt eltesszük a memóriába, és arra a memóriacímre hívatkozunk később, ha elő akarjuk hívni, vagy szeretnénk vele dolgozni. Egyszerű, nem igaz? Tegyük fel, hogy szeretnénk elraktározni továbbá a programunkkal a személy nevét is. Csinálunk egy Y változót, aminek értéket adunk. Ugyanúgy elraktározzuk a memóriában, és ugyanúgy tudunk dolgozni vele.

És ekkor jön egy kérés, hogy ezer ember adataival kellene dolgozni, az adatokba pedig nem csak a koruk és a nevük tartozik bele, hanem a születési dátumuk, a szemük színe, a TAJ számuk, az autójuk rendszáma ha van, a lakcímük, stb. Eljárás orientált programnyelveken itt szokott általában kisebb nehézségek adódni, persze korántsem áthidalhatatlan nehézségek. Azonban az objektum orientáció ezeket hidalja át. Mi lenne, ha nem egy 'int' típusú változót mentenénk el, hanem egy 'személy' típusú változót amihez tartoznak különböző attribútumok, például kor, név, TAJ szám, stb, és amikor szeretnénk az egyén értékeivel dolgozni, mi csak az adott személyre hivatkozunk és úgy nyerjük ki a hozzá tartozó adatokat. Egyszerűbb nem?

Erről szól az objektumorientáció. Megpróbáljuk objektumokba elképzeni a világot, és azt leírni a programunkba. Az objektumoknak pedig van viselkedése, és tulajdonsága. Például egy kutya objektumnak tulajdonsága hogy például barna a szőre, és viselkedése az hogy ugat (például kiírjuk hogy 'Vau' az outputra). Ilyen rendszerrel sokkal könnyebben, és ami fontosabb, sokkal intuitívebben tudunk adatokkal és információval foglalkozni, mi programozók. Ez a gyakorlatban úgy néz ki, hogy a forráskódban különböző osztályokat és alosztályokat írunk. Osztály lehet például kutya, alosztály pedig mondjuk vizsla. Ezekhez az osztályokhoz pedig tulajdonságokat és viselkedéseket írunk.

A modern programozásnak az egyik legnagyobb fegyvere az objektumorientáció, hisz ezzel emberi fejjel tudunk gondolkozni a gépek világában.

A C++ és a Java is objektumorientált. Mi akkor a legnagyobb különbség?

Igaz, hogy mindkét nyelv objektumorientált, azonban ha valaki ránéz egy C++ kódra rengeteg olyan jelet fog látni, ami nem bukkan elő egy Java kódban. Pointereket jelölő csillagokkal, és referenciákat jelölő 'és' jelekket van teletűzdelve a C++ kód. Javaban viszont minden egyszerűbben megy. Javaban minden referencia, így egy Java programozó előfordulhaz hogy soha nem fog pointerekkel találkozni. Ez bizonyos esetekben áldás, bizonyos esetekben pedig átok. Ugyan sokkal egyszerűbbé válik így a Java, mind tanulás mint pedig eszköz tekintetében, azonban fontos megjegyezni, hogy a C++ sokkal széleskörűbb felhasználást tud így nyújtani a használónak, illetve sokkal gyorsabb is. Persze a gyorsaság napjainkban már nem annyira releváns, hisz hatalmas erőforrások állnak a rendelkezésünkre, hogy ne kelljen aggódni hogy melyik nyelven íródott program fut le, nanoszekundumokkal később.

11.2. Python

A könyv olvasása, illetve a nyelv tesztelgetése és vizsgálódása alatt találtam rá, erre a szerintem igencsak szellemes poénra:

Hogyan írjuk át a pszeudokódot Python kódra?

mv code.txt code.py

Ezzel a kezdőgondolattal szeretném indítani rövid olvasónaplómat a Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprog- ramozásba című könyvről. Szerintem a poén igencsak nyilvánvalóvá teszi, hogy a Python nyelven íródott programok igencsak könnyen esnek a szemnek. Ez valóban így van. A Python híres arról hogy könnyen írható és olvasható. Ez egy magas szintű programozási nyelv amely platformfüggetlen és felhasználóbarát szintaxissal rendelkezik. De hogyan érte el mindezt?

A Java nelvben van egy extra réteg, amin végig megy a kód, amikor lefut, mielőtt még a compiler megkaparinthatná, s többek között ez az oka, hogy lassabban fut le. A Python-ban is van egy hasonló feature, azonban ez egy kicsit máshogy működik. Az interpreter közvetlenül a kódot olvassa és ezzel egyidejűleg futtatja, fordítás nélkül.

Ez az interpreter teszi platformfüggetlenné is a nyelvet. Amelyik gépen megtalálható az interpreter, azon futni fog a python kód. Ennélfogva a Java-hoz és a C++-hoz hasonlóan egy igencsak elterjedt és közkedvelt nyelvről van szó. Sokan lenézik a Java-t és a Pythont, egyszerűsége miatt, azonban saját véleményem épp az ellenkezője: Az egyszerűségre kell törekedni. Miért jó az, ha egy kód annyira bonyolult hogy szinte olvashatatlan egy átlag embernek? Miért jó az, ha tele van csillagokkal meg 'és' jelekkel az egész? Elegánsabb lesz tőle? Szebb? Nem. Csupán bonyolultabb. Személyes véleményem, hogy a programozást, mint eszközt, elérhetővé kellene tenni minél több embernek, hisz olyan mint az ecset a festő kezében, vagy a toll az íróében.

Maga a Python, egy számtalan csomagot és eljárást tartalmazó szkriptnyelv, és mivel igen széleskörű eszközök tárháza, elterjedt nyelv nagyobb alkalmazások készítésére is, ezen felül rengeteg 'nem programozó' is használja, mint például matematikusok vagy fizikusok. Talán pont emiatt is ilyen sikeres, hiszen manapság rengeteg szakterülthez kell programozási tudás, gondolok itt a matematikára, a fizikára, a biológiára, kémiára, viszont ezen szakterületeken kevesebb idő marad a programozási kézségek fejlesztésére, így a Python egy remek megoldást nyújt azoknak az embereknek akik nem feltétlenül a programozásra szentelték rá az életüket, de mégis szükségük van rá.

A Python, a temrészetéből adódóan rövid kódokat eredményez, hiszen sok minden ami máshol extra kifejtést igényelne, itt a programnyelvvel jár együtt. Magas szintű adattípusokat is tartalmaz, melyek ugyancsak hozzájárulnak a könnyed kezeléshez, ráadásul így nem szükséges a változótípis definiálása sem.

Természetesen a többi nyelvhez hasonlóan itt is jelen vannak a már megszokott ciklusok, a for, while, illetve az if-ek. Ezen felül definiálhatóak osztályok, amelyből adódnak az objektumok. Ezen felül rengeteg kényelmi funkcióval rendelkezik, különböző modulok formájában.

IV. rész

Irodalomjegyzék

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.