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, Bátfai, Mátyás, Bátfai, Nándor, Ã©s Bátfai, Margaréta	2020. május 28.	

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?

Kapd le a https://gitlab.com/nbatfai/bhax git repót, vagy ha már saját könyvön dolgozol, érdemesebb forkolnod belőle egyet magadnak!

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:

```
hp@b3r3s:~$ cd Dokumentumok/bhax/thematic_tutorials/ ↩
   bhax_textbook_IgyNeveldaProgramozod/
hp@b3r3s:~/Dokumentumok/bhax/thematic_tutorials/ ↩
   bhax_textbook_IgyNeveldaProgramozod$ 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 jól ment, akkor most éppen ezt az elkészített `bhax-textbook-fdl.pdf` fájlt olvasod.

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: http://esr.fsf.hu/hacker-howto.html!
- Olvasgasd aztán a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- C kapcsán a [KERNIGHANRITCHIE] könyv adott részei.
- C++ kapcsán a [BMECPP] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a The GNU C Reference Manual, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf`
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [BMECPP] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, https://www.imdb.com/title/tt0478087/, benne a Monty Hall probléma bemutatása.
- Kódjátszma, https://www.imdb.com/title/tt2084970, benne a kódtörő feladat élménye.

- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: Így neveld a programozód!VII. - VirtBox-os 3.

Megoldás forrása: Turing mappában fellelhető fájlok.

Ha csak azt akarjuk, hogy egy mag fusson 100 százalékban:

```
Egymag100

  int main(){

  for(;;);

  return 0;
  }
```

Illetve ha minden magot ~100%-on akarunk látni:

```
Mindenmag100
  #include <omp.h>

  int main(){

    #pragma omp parallel
    {
    for(;;);
    }

  return 0;
  }
```

A végeredményt a top parancsal kiválóan tudjuk szemléltetni:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
  5327 hp        20   0   68356   1028    932 R 792,0  0,0   1:33.35  ↩
     szazmag

  top - 13:10:16 up  1:30,  1 user,  load average: 1,91, 0,71, 0,70
  Tasks: 266 total,   2 running, 197 sleeping,   0 stopped,   0 zombie
  %Cpu0  :100,0 us,  0,0 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,   ↩
     0,0 st
  %Cpu1  :100,0 us,  0,0 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,   ↩
     0,0 st
  %Cpu2  : 99,7 us,  0,0 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,3 si,   ↩
     0,0 st
  %Cpu3  : 97,0 us,  2,3 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,7 si,   ↩
     0,0 st
  %Cpu4  : 98,7 us,  1,3 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,   ↩
     0,0 st
  %Cpu5  :100,0 us,  0,0 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,   ↩
     0,0 st
  %Cpu6  : 99,0 us,  1,0 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,   ↩
     0,0 st
  %Cpu7  : 99,3 us,  0,7 sy, 0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,   ↩
     0,0 st
  KiB Mem :  8036012 total,  3091312 free,  2655316 used,  2289384 buff/ ↩
     cache
  KiB Swap:  2097148 total,  2097148 free,        0 used.  4664684 avail  ↩
     Mem
```

Láthatjuk, hogy ha nincs is mindegyik 100%-on de nagyon közel áll hozzá. Ettől még az így kapott eredményünk elfogadható

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 fog 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? Sajnos legfőképp sehogy, mert ilyen `Lefagy` függvényt, azaz a T100 program nem is létezik.

2.3. Változók értékének felcserélése

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

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

Változókat többféleképpen is fel tudunk cserélni. Talán az egyik legegyszerűbb módja (segéd változó nélkül) az alábbi példa:

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

int a= 11;
int b= 7;

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

b=b-a;
a=a+b;
b=a-b;

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

  return 0;
  }
```

Ezen kívül természetesen még létezik jópár megoldás az egyik általunk tanult XOR (másnéven a kizáró vagy) módszer is alkalmas erre a feladatra.

```
#include <stdio.h>

int main(){

int a= 9;
int b= 13;

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

a= a^b;
b= b^a;
a= a^b;
```

```
  printf("Új a: %d \n",a);
  printf("Új b: %d \n",b);

  return 0;
  }
```

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül.

Hogy mindenkiben tisztázódjon mit is jelent az a pattogtatás, érdemes megnézni az alábbi bemutató videót.

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

Megoldás forrása:

Az első és talán a könnyebbik (if-es) verzió:

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

int main (void){

  WINDOW *ablak;
  ablak = initscr ();

  int x=0;
  int y=0;

  int deltax=1;
  int deltay=1;

  int mx;
  int my;

for (;;){

  getmaxyx(ablak , mx , my);

  myprintw(y , x , 'O');

  refresh();
  usleep(100000);

  clear();

  x=x+deltax;
  y=y+deltay;
```

```
  if( x <=0 ){
    deltax=deltax * -1;
  }
  if( x >=mx-1 ){
    deltax= deltax * -1;
  }
  if( y<=3){
    deltay=deltay * -1;
  }
  if( y>=my-1){
    deltay=deltay * -1;
  }
}
return 0;
}
```

Ahogy azt eddig is megszoktuk, a program itt is 'C' nyelvben íródott.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: Így neveld a programozód! VIII. - VirtBox-os 4. és Így neveld a programozód! IX. - VirtBox-os 5.

Megoldás forrása:

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

int main()
{
    printf("Adj meg egy számot: ");
    int a;
    int count=0;
    scanf("%d",&a);
    while (a!=0)
    {
        a<<=1;
        count++;
    }
    printf("%d\n",count);
}
```

A programunk lényege nem más mint, hogy az általunk bekért (kettes számrendszerben) értelmezett számot folyamatosan eltolja eggyel balra, egyészen a túlcsordulásig. Ezzel megkapjuk, hogy a a szám utolső 1-es bitje hány eltolással fog eltűnni, vagyis mekkora lehet a maximális mérete.

```
hp@b3r3s:~/Dokumentumok/bhax/thematic_tutorials/bhax_textbook_ ↩
   IgyNeveldaProgramozod/Turing$ ./a.out
Adj meg egy számot: 1
32
```

2.6. Helló, Google!

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

Megoldás forrása: Programozó Páternoszter

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

void printResult(double array[], int n);

double distance(double PR[], double PRv[], int n);

int main(){
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};

for(;;)
{
  for(int i = 0; i < 4; i++)
    PR[i] = PRv[i];
  for(int i = 0; i < 4; i++)
  {
    PRv[i] = 0.0;
  for(int j = 0; j < 4; j++){
    PRv[i] += L[i][j]*PR[j];
    }
  }
  if(distance(PR, PRv, 4) < 0.000001)
  break;
}
printResult(PR, 4);

return 0;
}
```

```
void printResult(double array[], int n){
  for(int i = 0; i < n; ++i){
    printf("%f\n", array[i]);
  }
}
double distance(double PR[], double PRv[], int n){
  double sum = 0.0;
  for(int i = 0; i < n; ++i){
    sum += (PR[i] - PRv[i])*(PR[i]-PRv[i]);
  }
return sqrt(sum);
}
```

A forrásban ugyan ott van a futtatása is, azonban jobb kétszer mint soha:

```
hp@b3r3s:~/Dokumentumok$ g++ pagerank.c -o pagerank
hp@b3r3s:~/Dokumentumok$ ./pagerank
0.090909
0.545454
0.272728
0.090909
```

Egy gyorstalpaló a link-assistant weboldal segítségével:

A PageRank (vagy röviden PR) a Google alapítói, Larry Page és Sergey Brin által a Stanfordi Egyetemen kifejlesztett webhelyek rangsorolására szolgál. Arra használják, hogy az egyes oldalak relatív pontszámot és tekintélyt kapjanak a linkek minőségének és mennyiségének értékelésével. Ezzel optimalizálva később a keresést, illetve a linkek sorrendjét. Azonban fontos tudni, hogy nem csupán a weblapokat tudunk beárazni.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

Hogy érthetőbb legyen minden, vegyük az egészet lépésről, lépésre. Elsőnek is lépjünk be az 'R' programunkon belül a matlab library-be, hogy a következő matematikai lépések elérhetőek legyenek számunkra.

```
  hp@b3r3s:~$ R
  > library(matlab)
```

Ebben a két sorban elmentjük majd kiiratjuk a primes paranccsal 1-től 100-ig a prímszámokat.

```
  > primes=primes(100)
  > primes
  [1]  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83  ↩
     89 97
```

Ezután kiiratjuk az egymást követő prímek különbségét, vagyis 3-2,5-3,stb..

```
> diff = primes[2:length(primes)]-primes[1:length(primes)-1]
> diff
[1] 1 2 2 4 2 4 2 4 6 2 6 4 2 4 6 6 2 6 4 2 6 4 6 8
```

Majd megnézzük, hogy ez a különbség melyik esetben lesz 2, majd azoknak az indexét kiiratjuk.

```
> idx = which(diff==2)
> idx
[1]  2  3  5  7 10 13 17 20
```

Ezután kiiratjuk a párok első:

```
> t1primes = primes [idx]
> t1primes
[1]  3  5 11 17 29 41 59 71
```

Majd a második tagját is:

```
> t2primes = primes[idx]+2
> t2primes
[1]  5  7 13 19 31 43 61 73
```

Ezután pedig lefuttatjuk a már előre megírt 'stp' függvényt így ezzel el is mentjük R-ben.

```
> 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))
+ }
```

Végül pedig ábrázoljuk az alábbi pontokat.

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Ezt a Monty hall féle problémát vagy paradoxont, a forrásban lévő programsorokkal jól tudjuk szemléltetni de többeknek már biztosan ismerős akár sorozatokból, akár filmekből is. Egy gyors talpaló előtte: A Monthy Hall probléma a Let's make a deal nevű amerikai TV-s vetélkedo híres műsorvezetőjéről kapta a nevét. A probléma a következoképpen néz ki: a műsorvezető mutat 3 ajtót a játékosnak, amelyek közül ˝ valamelyik mögött egy autó lapul. A játékos választ a 3 ajtó közül 1-et, majd ha a játékosnak nem sikerült beletrafálni, a műsorvezető a maradék 2 ajtó közül kinyit egyet, amelyik mögött garantáltan nincsen a nyeremény. Ezután a játékos választhat, hogy a döntését megtartja-e vagy megváltoztatja a másik zárt ajtóra.

```
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)
>
```

```
> 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)]
+
+ }
```

Jelen esetben 10millióra van álltíva a kísérletek száma de ezt állíthatjuk kedvünkre.

2.9. Vörös Pipacs Pokol

A feladat megtalálható a Turing mappában csigafel.py néven

A feladatunk az volt, hogy módosítsuk úgy a kódunkat, hogy Steve egy bizonyos csiga alakban haladjon felfelé az arénánkban. A megoldás megértéséhez elegendő lesz ez a rövidke kódrészlet. Mindenek előtt ledeklaráltam egy halad nevű változót aminek adtunk egy 2-es kezdőértéket. Amint látjuk utána következik egy while ciklus, amin belül látjuk a Steve-nek kiadott parancsokat. Ezek közül is a legfontosabb a "self.agent_host.sendCommand("move 1") time.sleep(halad)" parancs ami a későbbi "halad = halad + 1" változó növelésének segítségével teszi lehetővé azt, hogy Steve minden egyes alkalommal amikor lefut a while ciklus(vagyis amikor Steve elfordul) növeljük a megtett út időtartamát.

```
        halad = 2

        # Loop until mission ends:
        while world_state.is_mission_running:
            self.agent_host.sendCommand("move 1")
            time.sleep(halad)
            self.agent_host.sendCommand("jump 1")
            time.sleep(0.2)
            self.agent_host.sendCommand("jump 0")
            time.sleep(0.1)
            self.agent_host.sendCommand("turn 1")
            time.sleep(0.5)
            self.agent_host.sendCommand("turn 0")
            time.sleep(0.1)

            halad = halad + 1

            world_state = self.agent_host.getWorldState()
```

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Írjunnk egy olyan programot amely decimálisból unárisba váltja át majd iratja ki a számokat.

Megoldás videó:

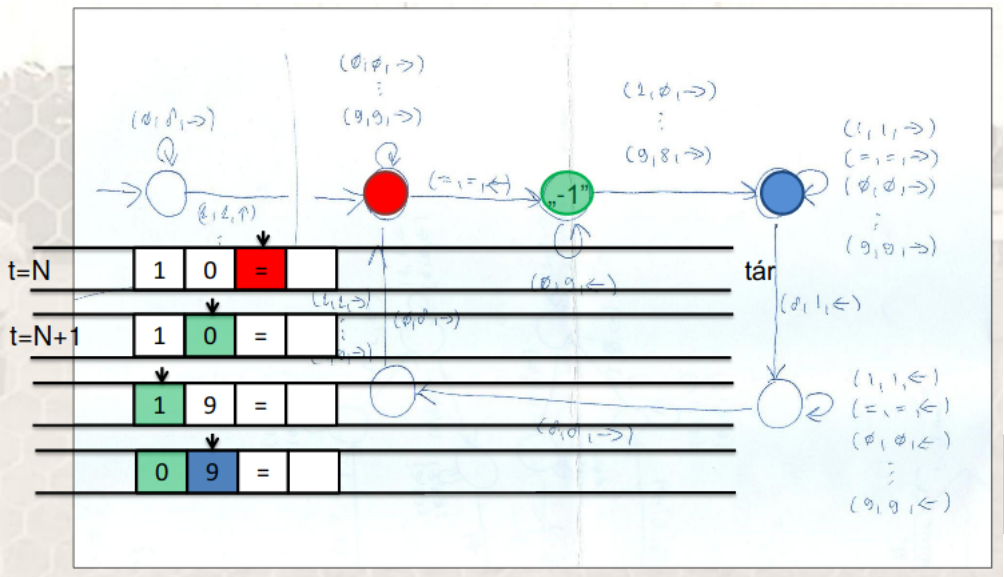
Megoldás forrása: Az előadás 27. fóliája.

Az első elképzelésem szerint haladtam miszerint az 1-es legyen a 'I', a 2 legyen 'II' és így tovább...

```
#include <stdio.h>
int main(){
int a = 0;
  printf("Irj be egy számot: ");
  scanf("%d",&a);
  printf("Unáris megjelenése: ");
  for(int i = 0; i<a;++i){
    printf("|\n");
}
printf("\n");
return 0;
}
```

Habár a programunk egy kicsit primitív és biztosan van ettől jobban, szebben működő is, egyelőre megteszi. Az egyetlen hátránya, hogy a nagyobb számoknál a kiiratás nem túlzottan kedvező a szemnek.

```
hp@b3r3s:~/Dokumentumok$ g++ decimalis.c -o decimalis
hp@b3r3s:~/Dokumentumok$ ./decimalis
Irj be egy számot: 20
Unáris megjelenése: ||||||||||||||||||||
hp@b3r3s:~/Dokumentumok$
```



3.1. ábra. Az előadás során is tárgyalt Turing gép működése

Forrás a működés megértéséhez: http://bodaistvan.hu/infalap/infotort/Turing_gep.html

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

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

Megoldás forrása: Az előadás 30-32. fóliája.

Nézzük meg milyen amikor A,B,C változók és a,b,c konstansok:

```
1. A -> aAB
2. A -> aC
3. CB -> bCc
4. cB -> Bc
5. C -> bc

A (A -> aC)
aC (C -> bc)
abc
A (A -> aAB)
aAB (A -> aAB)
aaABB (A -> aC)
aaaCBB (CB -> bCc)
aaabCcB (cB -> Bc)
aaabCBc (CB -> bCc)
aaabbCcc (C -> bc)
aaabbbccc
```

Ez kicsit zavaró lehet, ezért adjunk a változóknak valami eltérő nevet, mondjuk S,X,Y a konstansaink pedig változatlanul is a,b,c.

```
1. S -> abc
2. S -> aXbc
3. Xb -> bX
4. Xc -> Ybcc
5. bY -> Yb
6. aY -> aaX
7. aY -> aa


S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
aYbbcc (aY -> aaX)
aaXbbcc (Xb -> bX)
aabXbcc (Xb -> bX)
aabbXcc (Xc -> Ybcc)
aabbYbccc (bY -> Yb)
aabYbbccc (bY -> Yb)
aaYbbbccc (aY -> aaX)
aaaXbbbccc (Xb -> bX)
aaabXbbccc (Xb -> bX)
aaabbXbccc (Xb -> bX)
aaabbbXccc (Xc -> Ybcc)
aaabbbYbcccc (bY -> Yb)
aaabbYbbcccc (bY -> Yb)
aaabYbbbcccc (bY -> Yb)
aaaYbbbbcccc (aY -> aa)
aaaabbbbcccc
```

3.3. Hivatkozási nyelv

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

Megoldás forrása: Az előadás 63-65. fóliája.

Nézzünk egy programot erre a C89 és C99 közti különbségre a geeksforgeeks segítségét felhasználva.

```c
#include <stdio.h>
static inline int foo()
{
    return 2;
}


int main()
```

```
{

    int ret;


    ret = foo();

    printf("Output is: %d\n", ret);
    return 0;
}
```

Láthatjuk, hogy a C89 nem ismeri fel az ’inline’-t mivel az majd csak a C99-be került be.

```
hp@b3r3s:~/Dokumentumok/bhax/thematic_tutorials/ ↩
   bhax_textbook_IgyNeveldaProgramozod/Chomsky$ gcc 33.c -o proba1 -std=c89
33.c:2:15: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘__attribute__’ before  ↩
   ‘int’ static inline int foo()
```

Mint ahogy arra számíthattunk is, a programunk ezúttal zavartalanul le is fordul.

```
hp@b3r3s:~/Dokumentumok/bhax/thematic_tutorials/ ↩
   bhax_textbook_IgyNeveldaProgramozod/Chomsky$ gcc 33.c -o proba2 -std=c99
```

3.4. Saját lexikális elemző

A megírt programunknak a lényege, hogy számolja és megértse a bemeneten megjelenő valós számokat. A forrásunk egy lex forrás ami saját maga csinál belőle egy C programot.

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;
}
```

Hogy mindenkineki értherő legyen, vegyük az egészet lépsről lépésre. Elsőnek is a realnum.é fájlból csinálnunk kell egy c forrást.

```
hp@b3r3s: lex -o realnumber.c -o realnumber.l
```

Ha ezzel megvagyunk, akkor a jól megszokott módon fordítjuk, és a -lfl paranccsal a végén linkeljük a flexet.

```
hp@b3r3s: gcc realnumber.c -o realnumber -lfl
```

A végén pedig csak le kell futtatnunk és ne féljünk megfejelni azt a billentyűzetet, a programunk majd kiválogatja a számára értékes és Valós számokat.

```
./realnumber
  wdawgt4Z75S6ISRha5j6skhzxt
  wdawgt[realnum=4 4.000000]Z[realnum=75 75.000000]S[realnum=6 6.000000] ↩
     ISRha[realnum=5 5.000000]j[realnum=6 6.000000]skhzxt
```

Ahogy az végeredményként láthatjuk, a sok betű és szám közül megtalálta a 4-et, a 75-öt, a 6-ot, az 5-öt és ismételten a 6-ot. A programunk természetesen nem csak egész számokkal tud megbirkózni, ez csak szimpla véletlen hogy a homlokom ezeket a billentyűket nyomta le.

3.5. Leetspeak

Egész biztos vagyok benne, hogy mindenki találkozott már a leetspek-el még akkor is ha nem tudott róla, de a biztonság kedvéért ITT egy gyorstalpaló róla. Az interneten gyorsan elterjedté és kedveltté vált. Én személy szerinte elsőnek egy számítógépes játékon belül találkoztam ezzel, ahol az illetőnek H3nt3s volt a felhasználóneve. Igaz akkor még csak azt tudtam, hogy mit helyettesítenek azok a hármasok, de a történetéről és a valódi hátteréről még nem sejtettem semmit.

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

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

Az alábbi példa és a Chomsky mappában lévő doksi is C nyelvre specializálódott, így ajánlott elsőnek azzal próbálkozni. Természetesen aki szeretné később arra nyelvre alakítja át amelyikre szeretné.

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

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

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

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

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

  };

%}
%%
. {

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

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

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

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

        found = 1;
        break;
      }

    }

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

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

A chomsky mappában lévő fájlt elsőnek forítanunk kell C nyelvre, akiknek alapból nincs letöltve annak előtte a -sudo apt install flex- commanddal le kell töltenie a lex parancsot.

```
hp@b3r3s: lex -o l337d1c7.c -o l337d1c7.l
```

Majd ha ez megtörtént a gcc paranccsal lefordíttatjuk.

```
hp@b3r3s: gcc l337d1c7.c -o l337d1c7 -lfl
```

Ha mindezzel sikeresen megvagyunk, futtathajuk is.

```
hp@b3r3s: ./l337d1c7 -lfl
```

Ezután kedvünkre írhatunk be a parancssorba bármit, a programunk majd teszi a dolgát ha mindent jól csináltunk.

```
hp@b3r3s: My name is Jeff
          my |\|4m3 1$ j3ff
```

Mivel ez a nyelv alapjába véve titkosítás céljaként jött létre, ennél fogva egyes betűknél érdemes benthagyni az eredeti verziót is, ezzel is nehezítjük a megfejthetőségét. Ezzel ugyanis adunk egy extra nehezítést mivel

nem tudhatjuk biztosra, hogy a kiválasztott betű az valóban azt a betűt takarja vagy egy másiknak a kódolt verziója.

Az általunk megírt program természetesen nagyban hasonlít az összes többi, akár már az interneten is megtalálható átváltókhoz amik valószínűleg hasonló algoritmus alapján működnek.

3.6. A források olvasása

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

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

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

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

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

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

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

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

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

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

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

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

Kezdjük tehát a legegyszerűbbtől, a bonyolultak felé, tehát fentről lefelé.

i: Ha a SIGINT jel nem volt figyelmen kívül hagyva, akkor ezen túl se legyen. Ez a példánk nagyban hasonlít a mintapéldához, itt azonban a '!=' párossal úgymond negáljuk az egyenlőségjelet.

ii: Jelen esetben van egy for ciklusunk ami előreláthatólag 5x fog lefutni ugyanis az i változónk rendelkezik egy kezdeti 0 értékkel (i=0) ami folyamatosan növekszik 1-el (++i) egészen addig amíg kisebb mint 5, vagyis 4-ig.

iii: A példánk lényegében ugyan az mint az előző, annyi eltéréssel, hogy itt elsőnek megkapjuk az i értékét és aztán növeljük.

iv: A ciklusunk változatlanul 5x fog lefutni, viszont bevezettünk egy tömböt is, melynek i-edik eleméhez folyamatosan (iterációnként) hozzárendeljük az i++ értékét.

v: A forrásunk legfontosabb az az első ';' után található dupla feltétel, melynek mindkettőnek teljesülnie kell, hogy a ciklusunk tovább fusson.

vi: Az már a legelején látszik, hogy két számot fogunk kiiratni, mégpedig a két f függvényünk egy-egy értékét.

vii: A már jól megszokott kiiratást láthatjuk a printf felhasználásával, mégpedig az f függvény visszatérési értékét, majd pedig magát az 'a'-t (nem elfeledve, hogy 'a' jelen esetben egy szám lesz).

viii: És végül de nem utolsó sorban, a már jól begyakorolt kiiratás fog lezajlani, elsőnek tehát f visszatérési értéke (Figyelem! Most a-nak a memóriacíme a paraméter.) majd pedig maga az 'a'.

3.7. Logikus

Az alábbi Ar nyelvi példákat lefordítjuk a természetes nyelvre. Most összesen 4 típust nézünk meg de ezeken kívűl rengeteg féle kifejezést tudunk készíteni hasonló módon. Mindezt a tudást valószínűleg az első félévben az Informatika logikai alapjai tantárgy során remélhetőleg már elsajátítottátok így nem kell, hogy gondot jelentsen egyik tanulónak sem.

Mielőtt nekivágunk azonban tisztázzuk az előforduló a kifejezéseket, hogy ne merüljenek fel kételyek. "\forall" = ∀, "\exists" = ∃, "\wedge" = ∧, "\supset" = ⊃, "\neg" = ¬. Ezek tudatában már gyerekjátéknak kell hogy legyen.

Nézzük az elsőt, ígérem nem lesz nehéz kitalálni. Minden x-nél létezik egy tőle nagyobb y prímszám, összegezve: végetelen sok prím van.

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

Ejtsd: Végtelen sok ikreprímszám van. Azaz minden x prímnél van egy 2-vel nagyobb y prím is.

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

A következő példánk jelentése nem más mint, hogy véges sok prímszám van. Egy gyors magyarázat rá, hogy minden x prímszámhoz tartozik egy tőle nagyobb y szám.

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

Ne essünk kétségbe ha ugyanazt az eredményt kaptuk mint az előzőnél, ugyanis a jelentésük teljesen megegyezik, egyedül a szerkezetük más.

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

Ha valami azonban mégse lenne világos, lejjebb görgetve találtok egy megoldás videót illetve annak forrását.

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

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

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
  int a;
  ```
- ```
  int *b = &a;
  ```
- ```
  int &r = a;
  ```
- ```
  int c[5];
  ```

- ```
  int (&tr)[5] = c;
  ```
- ```
  int *d[5];
  ```
- ```
  int *h ();
  ```
- ```
  int *(*l) ();
  ```
- ```
  int (*v (int c)) (int a, int b)
  ```
- ```
  int (*(*z) (int)) (int, int);
  ```

Megoldás videó:

Megoldás forrása:

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összahasonlítása azt mutatja meg, hogy miért érdemes a **typedef** használata: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr.c, bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr2.c.

```
#include <stdio.h>

int
sum (int a, int b)
{
    return a + b;
}

int
mul (int a, int b)
{
    return a * b;
}

int (*sumormul (int c)) (int a, int b)
{
    if (c)
        return mul;
    else
        return sum;

}

int
main ()
{
```

```
    int (*f) (int, int);

    f = sum;

    printf ("%d\n", f (2, 3));

    int (*(*g) (int)) (int, int);

    g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

```
#include <stdio.h>

typedef int (*F) (int, int);
typedef int (*(*G) (int)) (int, int);

int
sum (int a, int b)
{
    return a + b;
}

int
mul (int a, int b)
{
    return a * b;
}

F sumormul (int c)
{
    if (c)
        return mul;
    else
        return sum;
}

int
main ()
{

    F f = sum;

    printf ("%d\n", f (2, 3));
```

```
    G g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

3.9. Vörös Pipacs Pokol/csiga diszkrét mozgási parancsokkal

A feladatunk megoldás megtalálható a Chomsky mappában secondcircle.py néven.

A mappában található kód már egy kicsit előrehaladott, ugyanis tartalmazza a pipacsok érzékelését, illetve azoknak kiszedését is és ez alapján halad felfele az arénában.

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!

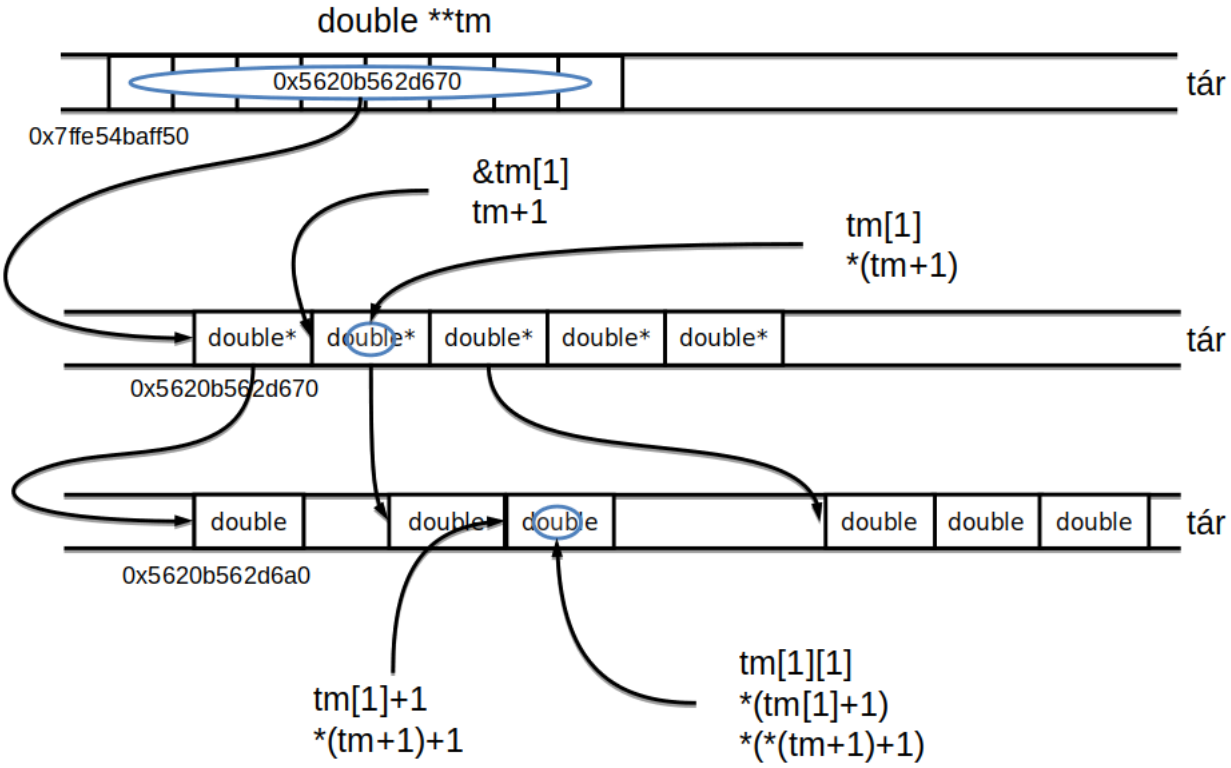
Megoldás videó: https://youtu.be/1MRTuKwRsB0, https://youtu.be/RKbX5-EWpzA.

Illetve a saját (védési) videó https://www.youtube.com/watch?v=8HSc1-dwNic

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

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

int main(){

int nr = 5;
double**tm;

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

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

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

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

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


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

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

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

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

  free (tm);
return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Hogy miképpen történik az alsó háromszög mátrix tárolása? A magyarázat az, hogy a mátrix felsőháromszög mátrixa 0-val lesz tele, emiatt ezt nem szükséges eltárolnunk (ezért is hívjuk helytakarékosnak), csak a fontos adatot, vagyis az alsó részt. Miután a malloc sikeresen lefutott, az jóesetben visszaadja az adatra mutató pointer-t. Amennyiben a futtatása nem sikerül, egy null pointert kapunk vissza. A lefoglalt terület nagyságát malloc paramétereként kell megadni. (legelső esetben nr* sizeof(double *), nr = 5 esetén 40 bájt). ???még nem fix???

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: Felvételt hirdet a CIA

Itt látható a C nyelvben íródott titkosítóprogram.

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

#define MAX_KULCS 100
#define BUFFER_MERET 245

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

```
char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];

int kulcs_index = 0;
int olvasott_bajtok = 0;

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

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

  write (1, buffer, olvasott_bajtok);
  }

}
}
```

A titkosító programunk mellé szükségünk lesz, egy sima szöveges állományra is amibe egy általunk kiválasztott szöveget másolhatunk. Ezt fogjuk később titkosítani, majd feloldani. Nevezzük el ezt a fájlt tiszta.szoveg -nek. Én a BIblia I. fejezetéből választottam. Ha megvan a titkosítattni kívánt szöveg, kezdjük is meg a szemléltetését. Elsőnek is fordítsuk le a programunkat a következőképpen:

```
hp@b3r3s:~/Dokumentumok/prog1$ gcc e.c -o e -std=c99
```

Ezután a már megírt titkosítóba bevisszük a tiszta szövegünket majd az így keletkezett "bináris szemetet" a titkos.szoveg fájlba tároljuk el. Így ni:

```
hp@b3r3s:~/Dokumentumok/prog1$ ./e kercerece <tiszta.szoveg >titkos.szoveg
```

Ezt ha nem baj most nem reprezentálnám, hogy mi lesz belőle ugyanis csupa krikszkraszokból fog állni a terminál ha végigpörgetjük. Akit azonban mégis érdekelne a sajátja, vagy csak letesztelné hogy működik-e, az az alábbi paranccsal.

```
hp@b3r3s:~/Dokumentumok/prog1$ more titkos.szoveg
ZERCE9
...
```

Mindez azonban fordítva is működik, azaz a mások által ugyan ezzel a programmal lefordított szövegeket tudjuk dekódolni is. Mondanám, hogy aki nem hiszi járjon utána de ez esetben mi is így teszünk. Nem nehéz kitalálni az eddigiek alapján, hogy hogyan is kell visszafejteni de a biztonság kedvéért bemutatom:

```
hp@b3r3s:~/Dokumentumok/prog1$ ./e kercerece <titkos.szoveg
1    Kezdetben vala az Íge, és az Íge vala az Istennél, és Isten vala az ↩
   Íge.
```

```
2    Ez kezdetben az Istennél vala.
...
```

Ez eddigiek során számomra talán ez volt az egyik legizgalmasabb és leglátványosabb feladat. Alapból mindig is érdekelt a számítástechnikának ez az ágazata. Nagyon elcsépelt szöveg de valóban általános iskolás korom óta etikus hacker szeretnék lenni. Még akkor is ha mindez nagyon távol áll ettől az álomtól, akkor is talán mondhatni, ez egy amolyan kis "bevezető" lehet ebbe a világba. Így hát nem mondok el nagy titkot ha azt állítom, hogy (személy szerint) ez volt az eddigi legjobb feladat. Természetesen a kódsorozat gyorsan és könnyen elkészíthetősége miatt is tetszett de ez már csak hab a tortán.

4.3. Java EXOR titkosító

Az előző programunkból kiindulva , megírjuk a java féle titkosítónkat is, ezúttal azonban az eddigiektől eltérően, Java-t fogunk használni.

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#exor_titkosito

```
public class ExorTitkosító {

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

        byte [] kulcs = kulcsSzöveg.getBytes();   //Adjunk értéket a ↩
           kulcsunknak
        byte [] buffer = new byte[256];     //Megkreáljuk a buffert
        int kulcsIndex = 0;
        int olvasottBájtok = 0;

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

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

                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]); //XOR m ↩
                   űvelet végrehajtása a buffer és a kulcs között, vagyis ↩
                   titkosítunk.
                kulcsIndex = (kulcsIndex+1) % kulcs.length;

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

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

```
        try {
            new ExorTitkosító(args[0], System.in, System.out);
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

Remélem ezúttal senki sem fog megharagudni illetve pánikba esik ha azt mondom, hogy nem fogok minden parancsot külön-külön elmagyarázni, hogy mit is csinál, ugyanis minden megegyezik a C programéval. Ettől függetlenül természetesen bemutatom, hogy mit is kell a terminálba ügyeskednünk.

```
hp@b3r3s:~/Dokumentumok/prog1$ javac ExorTitkosító.java
hp@b3r3s:~/Dokumentumok/prog1$ java ExorTitkosító erdeti > kodolt.szöveg
Malmö Svédország harmadik legnépesebb városa.
hp@b3r3s:~/Dokumentumok/prog1$ more kodolt.szöveg
...valami olvashatatlan...
hp@b3r3s:~/Dokumentumok/prog1$ java ExorTitkosító eredeti < kodolt.szöveg
Malmö Svédország harmadik legnépesebb városa.
```

4.4. C EXOR törő

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

Megoldás forrása: Programozó Páternoszter illetve a t.c fájl.

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

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

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

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

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

  // A tiszta szöveg feltehetően tartalmazza a gyakori magyar szavakat ↩
     illetve az átlagos szóhossz vizsgálatával csökkentjük a potenciális ↩
     töréseket.
  double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

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

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

  // összes kulcs előállítása
  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);

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

  return 0;
}
```

A program fordítása és futtatása előtt győződjünk meg róla, hogy elkészült a titkos szövegünk.

```
hp@b3r3s:~/Dokumentumok/bhax/prog1$  ./e kercerece <tiszta.szoveg > titkos. ↩
```

```
   szoveg
```

Ezután jöhet a jól megszokott parancs páros amivel ha minden igaz vissza tudjuk fejteni a lekódolt szöveget.

```
hp@b3r3s:~/Dokumentumok/bhax/prog1$  gcc t.c -o t
hp@b3r3s:~/Dokumentumok/bhax/prog1$  ./t < titkos.szoveg
```

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

Azoknak esetleg akiknek újdonság lenne a neurális hálózat mint fogalom, itt egy gyorstalapló a tankonyvtar oldaláról: Neurális hálózatnak nevezzük azt a hardver vagy szoftver megvalósítású párhuzamos, elosztott működésre képes információfeldolgozó eszközt, amely:

-azonos, vagy hasonló típusú – általában nagyszámú – lokális feldolgozást végző műveleti elem, neuron (processing element, neuron) többnyire rendezett topológiájú, nagymértékben összekapcsolt rendszeréből áll,

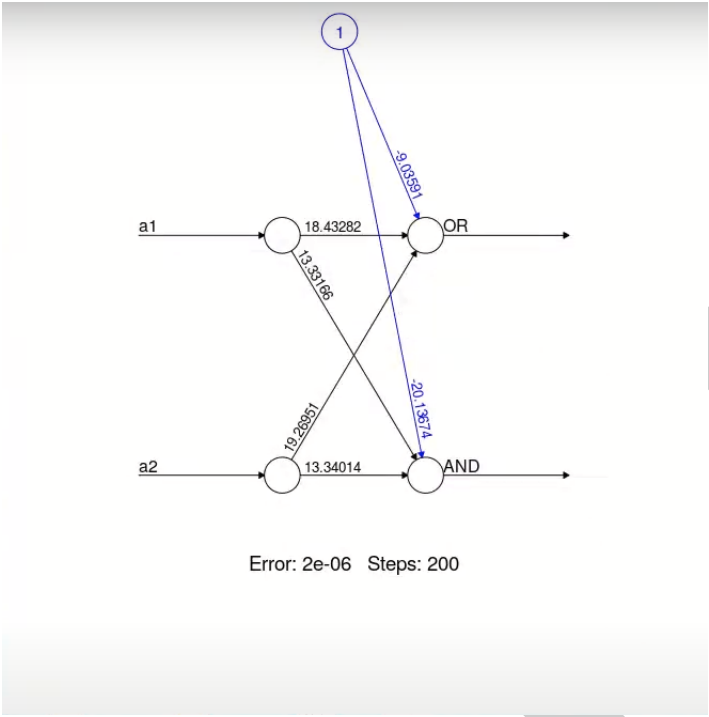
-rendelkezik tanulási algoritmussal (learning algorithm), mely általában minta alapján való tanulást jelent, és amely az információfeldolgozás módját határozza meg,

-rendelkezik a megtanult információ felhasználását lehetővé tevő információ előhívási, vagy röviden előhívási algoritmussal (recall algorithm).

```
  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])
```

A fenti programban a neuralnet libraryt használjuk. A program első részében az OR logikai műveletet mutatjuk be és taníttatjuk meg a programunkkal. Az OR művelet eredménye 1, ha a két tag közül legalább az egyik 1-es. Ezt a szabályt adjuk meg neki a program elején az 'a1', 'a2' és 'OR' segítségével. A program többi részében is ezt láthatjuk az EXOR-ral és az AND-del. A program a szabályok megadásával megtanulja, hogy az EXOR (különböző bitek -> 1, azonos bitek -> 0) és az AND (különböző vagy 0 bitekUniverzális programozás -> 0, két 1-es bit -> 1) hogyan működik.



4.2. ábra. Gráf alapú modell

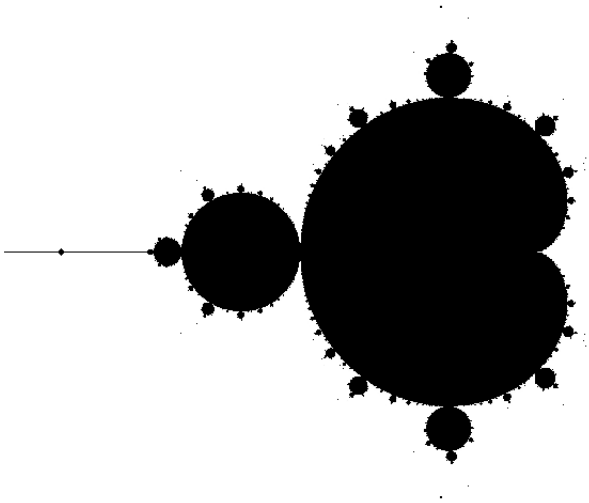
4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

```
#include <iostream>
#include "mlp.hpp"
#include <png++/png.hpp>
int main ( int argc, char **argv){
png::image <png::rgb_pixel> png_image (argv[1]);
int size = png_image.get_width()*png_image.get_height();
Perceptron* p = new Perceptron (3, size, 256, 1);
double* image = new double[size];
for ( int i {0}; i < png_image.get_width(); ++i)
for ( int j {0}; j < png_image.get_height(); ++j)
image[i*png_image.get_width() +j] = png_image[i][j].red;
double value = (*p)(image);
std::cout<< value <<std::endl;
delete p;
delete [] image;
FT
}
```



4.3. ábra. Gráf alapú modell

A teszteléshez a ql.hpp-ben található Perceptron osztályt használjuk. Ezzel a programmal a mandelbrot halmaznak a képét olvassuk be a neurális hálónk bemenetébe a következőképpen:

```
hp@b3r3s:~/Dokumentumok/bhax/prog1$ gcc main.cpp mlp.cpp -o percep -lpng
hp@b3r3s:~/Dokumentumok/bhax/prog1$ ./percep mandel.png
```

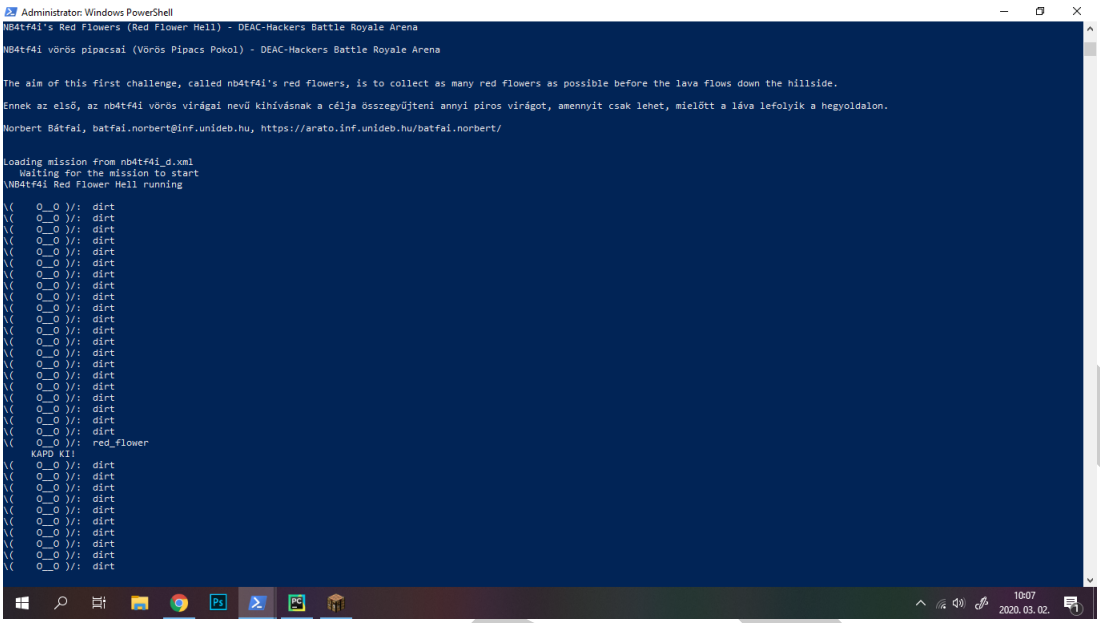
Egy neurális háló 3 rétegből áll. Az első réteg a bemeneti réteg, ennek a rétegnek a mérete a kép szélessége * kép magassága. A második réteg a rejtett réteg, itt alakítjuk át a bemeneten kapott információt, itt számolások történnek, a fenti kódban ez a két for ciklussal történik. A harmadik réteg a kimeneti réteg, itt a kapott eredményt iratjuk ki.

4.7. Vörös Pipacs Pokol/ írd ki mit lát Steve

Mivel a Chomsky-féle programunk már egyébként tartalmazta ezt a tulajdonságot is, így elég ha csak azt az aprócska kódcsipetet ide bemásolom.

```
while world_state.is_mission_running:
            if world_state.number_of_observations_since_last_state != 0:
                sensations = world_state.observations[-1].text
                observations = json.loads(sensations)
                nbr3x3x3 = observations.get("nbr3x3", 0)

                if "LineOfSight" in observations :
                    lineOfSight = observations["LineOfSight"]
                    self.lookingat = lineOfSight["type"]
                    print("\(    O__O )/: ", self.lookingat)
```

Ez lényegében annyit jelent, hogy ameddig a programunk fut, a terminálba folyamatosan látni fogjuk amire éppen Steve néz.



4.4. ábra. Lássuk, mit lát Steve

5. fejezet

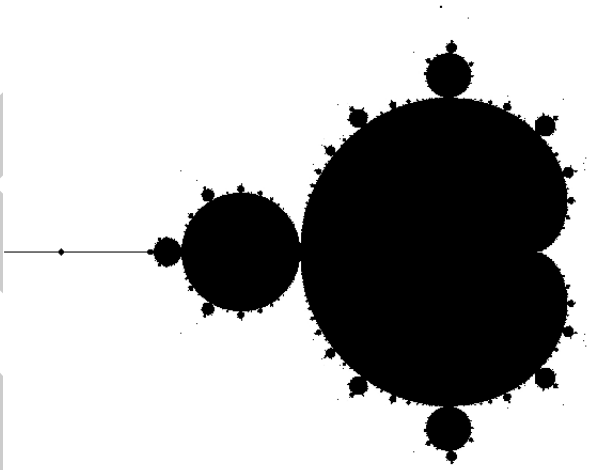
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása: `bhax/attention_raising/CUDA/mandelpngt.c++` nevű állománya.



5.1. ábra. A Mandelbrot halmaz a komplex síkon

A Mandelbrot halmazt 1980-ban találta meg Benoit Mandelbrot a komplex számsíkon. Komplex számok azok a számok, amelyek körében válaszolni lehet az olyan egyébként értelmezhetetlen kérdésekre, hogy melyik az a két szám, amelyet összeszorozva -9-et kapunk, mert ez a szám például a 3i komplex szám.

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak

felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1}=z_n{}^2+c$, (0<=n) képlet alapján úgy, hogy a c az éppen vizsgált rácspont. A z0 az origó. Alkalmazva a képletet a

- $z_0 = 0$
- $z_1 = 0^2+c = c$
- $z_2 = c^2+c$
- $z_3 = (c^2+c)^2+c$
- $z_4 = ((c^2+c)^2+c)^2+c$
- ... s így tovább.

Azaz kiindulunk az origóból (z_0) és elugrunk a rács első pontjába a z_1 = c-be, aztán a c-től függően a további z-kbe. Ha ez az utazás kivezet a 2 sugarú körből, akkor azt mondjuk, hogy az a vizsgált rácspont nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végtelen sok z-t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácsponthoz. Ha eközben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácspont a halmaz része. (Színes meg úgy lesz a kép, hogy változatosan színezzük, például minél későbbi z-nél lép ki a körből, annál sötétebbre).

```
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>
#define MERET 600
#define ITER_HAT 32000
void
mandel (int kepadat[MERET][MERET]) {
  // Mérünk időt (PP 64)
  clock_t delta = clock ();
  // Mérünk időt (PP 66)
  struct tms tmsbuf1, tmsbuf2;
  times (&tmsbuf1);
  // számítás adatai
  float a = -2.0, b = .7, c = -1.35, d = 1.35;
  int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;
  // a számítás
  float dx = (b - a) / szelesseg;
  float dy = (d - c) / magassag;
  float reC, imC, reZ, imZ, ujreZ, ujimZ;
  // Hány iterációt csináltunk?
  int iteracio = 0;
  // Végigzongorázzuk a szélesség x magasság rácsot:
  for (int j = 0; j < magassag; ++j)
  {
    //sor = j;
    for (int k = 0; k < szelesseg; ++k)
    {
    // c = (reC, imC) a rács csomópontjainak
    // megfelelő komplex szám
```

```
    reC = a + k * dx;

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

      ++iteracio;
    }

    kepadat[j][k] = iteracio;
  }
}
times (&tmsbuf2);
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;
delta = clock () - delta;
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}
int main (int argc, char *argv[])
{
  if (argc != 2)
  {
  std::cout << "Hasznalat: ./mandelpng fajlnev";
  return -1;
  }
  int kepadat[MERET][MERET];
  mandel(kepadat);
  png::image < png::rgb_pixel > kep (MERET, MERET);

  for (int j = 0; j < MERET; ++j)
  {
    //sor = j;
    for (int k = 0; k < MERET; ++k)
    {
```

```
      kep.set_pixel (k, j,
        png::rgb_pixel (255 -(255 * kepadat[j][k]) / ITER_HAT, 255 -(255 *  ↩
           kepadat[j][k]) / ITER_HAT, 255-(255 * kepadat[j][k]) / ITER_HAT) ↩
           );
    }
  }

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

Ezután már csak annyi dolgunk van, hogy a programunkat lefordítsuk, majd futtassuk a szokásos parancsokkal.

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:

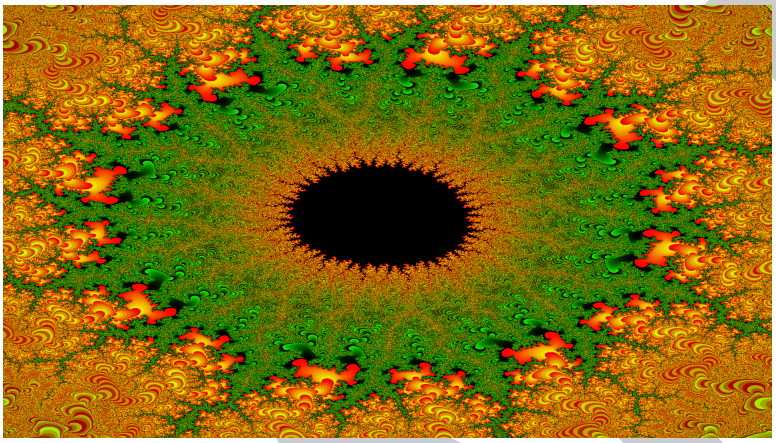
A Mandelbrot halmaz pontban vázolt ismert algoritmust valósítja meg a repó bhax/attention_raising/Mandelbrot/3.1.2.cpp nevű állománya.

```
#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;

}
```

Arról, hogy miképpen kell futtatnunk a megírt programunkat, arról a-../../../bhax/attention_raising/Mandelbrot/3.1 fájlon belül pontos leírást kapunk.



5.2. ábra. A Mandelbrot halmaz

5.3. Biomorfok

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

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

A biomorfokra (a Julia halmazokat rajzoló bug-os programjával) rátaláló Clifford Pickover azt hitte természeti törvényre bukkant: https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf (lásd a 2307. oldal aljától).

A különbség a Mandelbrot halmaz és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

```
  // j megy a sorokon
  for ( int j = 0; j < magassag; ++j )
    {
      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;
            }
```

Ezzel szemben a Julia halmazos csipetben a cc nem változik, hanem minden vizsgált z rácspontra ugyanaz.

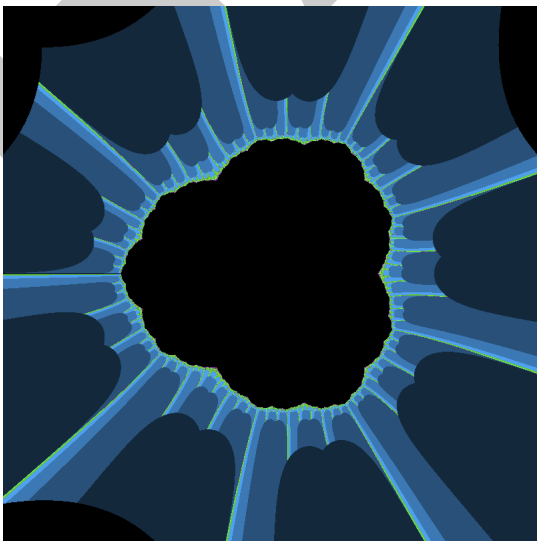
```
    // j megy a sorokon
    for ( int j = 0; j < magassag; ++j )
    {
        // k megy az oszlopokon
        for ( int k = 0; k < szelesseg; ++k )
        {
            double reZ = a + k * dx;
            double imZ = d - j * dy;
            std::complex<double> z_n ( reZ, imZ );

            int iteracio = 0;
            for (int i=0; i < iteraciosHatar; ++i)
            {
                z_n = std::pow(z_n, 3) + cc;
                if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                {
                    iteracio = i;
                    break;
                }
            }
```

A bimorfos algoritmus pontos megismeréséhez ezt a cikket javasoljuk: https://www.emis.de/journals/-TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf. Az is jó gyakorlat, ha magából ebből a cikkből from scratch kódoljuk be a sajátunkat, de mi a királyi úton járva a korábbi Mandelbrot halmazt kiszámoló forrásunkat módosítjuk. Viszont a program változóinak elnevezését összhangba hozzuk a közlemény jelöléseivel:

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

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

```
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag =  atoi ( argv[3] );
        iteraciosHatar =  atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );

    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↩
            d reC imC R" << std::endl;
        return -1;
    }

    png::image < png::rgb_pixel > kep ( szelesseg, magassag );

    double dx = ( xmax - xmin ) / szelesseg;
    double dy = ( ymax - ymin ) / magassag;

    std::complex<double> cc ( reC, imC );

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

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

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

            double reZ = xmin + x * dx;
            double imZ = ymax - y * dy;
```

```
            std::complex<double> z_n ( reZ, imZ );

            int iteracio = 0;
            for (int i=0; i < iteraciosHatar; ++i)
            {

                z_n = std::pow(z_n, 3) + cc;
                //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
                if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                {
                    iteracio = i;
                    break;
                }
            }

            kep.set_pixel ( x, y,
                            png::rgb_pixel ( (iteracio*20)%255, (iteracio ↩
                               *40)%255, (iteracio*60)%255 ));
        }

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

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

}
```



5.3. ábra. A biomorf

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.

A CUDA (Compute Unified Device Architecture) – az NVIDIA grafikus processzorainak általános célú programozására használható környezet. Tudhatjuk meg az Elte tudástárának segítségével.

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

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


#define MERET 600
#define ITER_HAT 32000

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

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

  // a számítás
  float dx = (b - a) / szelesseg;
  float dy = (d - c) / magassag;
  float reC, imC, reZ, imZ, ujreZ, ujimZ;
  // Hány iterációt csináltunk?
  int iteracio = 0;

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

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

      ++iteracio;

    }
  return iteracio;
}


/*
__global__ void
mandelkernel (int *kepadat)
{

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

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

}
*/

__global__ void
mandelkernel (int *kepadat)
{

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

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

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

}

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

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

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

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

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

}

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

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

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

  int kepadat[MERET][MERET];

  cudamandel (kepadat);

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

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

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

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

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

}
```

A CUDA SDK az **nvcc** saját fordítóval érkezik, mind Windows mind Linux operációs rendszerekre. Ez az egy fordító végzi a gazda kódok fordítását is. Működése a következő: lefordítja a GPU kódot majd a CPU kódba behelyettesíti a megfelelő CUDA hívásokat. A CPU kódot a továbbiakban pedig átadja az adott platformon alapértelmezett fordítónak. Ez Windows rendszereken a Visual C++ fordító, míg Linux rendszereken a GCC.

A program fordítása előtt előfordulat, hogy szükségünk lesz az nvcc parancsra, ugyanis nem mindenkinek áll egyből rendelkezésre, úgyhogy ajánlott a **sudo apt install nvidia-cuda-toolkit** command-al letölteni.

Ezután már fordíthatjuk és futtathatjuk is a következőképpen:

```
hp@b3r3s:~/Dokumentumok$ nvcc mandelpngc.cu -lpng16 -o mandelpngc
hp@b3r3s:~/Dokumentumok$ ./mandelpngc cuda.png
```

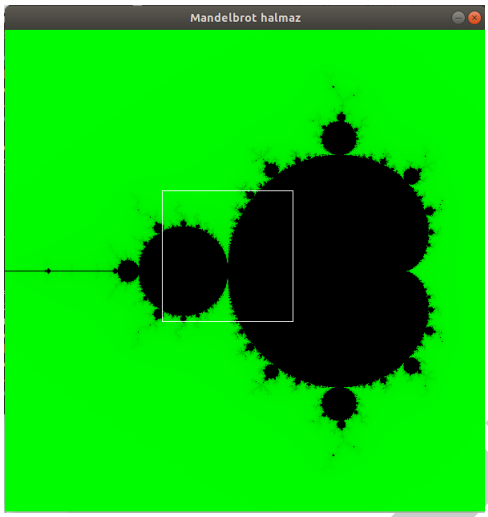
5.5. Mandelbrot nagyító és utazó C++ nyelven

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!

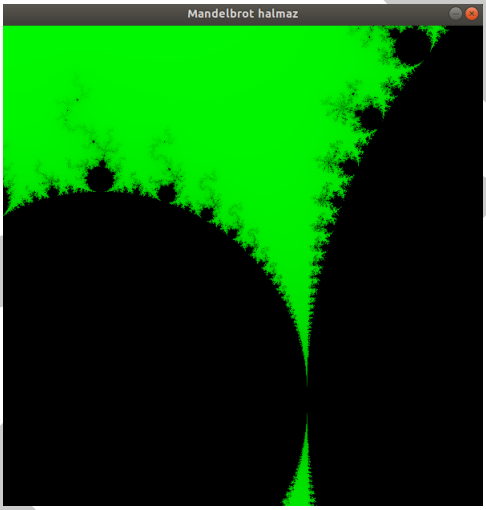
Megoldás videó: Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal.

Megoldás forrása: 26-33 fólián át láthatjuk, illetve az ITT látható repóban is.

A program fordítása és futtatás után a lenti képet fogjuk kapni eredményül, amin belül az egér segítségével nagyíthatunk kedvünkre.



5.4. ábra. Nagyító használata



5.5. ábra. A felnagyított Mandelbrot halmaz

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

A forrásban fellelhető kódunk ezúttal, ahogy azt a feladat is megköveteli, Java nyelven íródott de ez ne ijesszen meg senkit, ugyan úgy működik akár a c++-ban megírt program.

```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /** A nagyítandó kijelölt területet bal felső sarka. */
    private int x, y;
    /** A nagyítandó kijelölt terület szélessége és magassága. */
```

```
    private int mx, my;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nygítani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     *
     * @param      a              a [a,b]x[c,d] tartomány a koordinátája.
     * @param      b              a [a,b]x[c,d] tartomány b koordinátája.
     * @param      c              a [a,b]x[c,d] tartomány c koordinátája.
     * @param      d              a [a,b]x[c,d] tartomány d koordinátája.
     * @param      szélesség      a halmazt tartalmazó tömb szélessége.
     * @param      iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
            int szélesség, int iterációsHatár) {
        // Az ős osztály konstruktorának hívása
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        // Egér kattintó események feldolgozása:
        addMouseListener(new java.awt.event.MouseAdapter() {
            // Egér kattintással jelöljük ki a nagyítandó területet
            // bal felső sarkát vagy ugyancsak egér kattintással
            // vizsgáljuk egy adott pont iterációit:
            public void mousePressed(java.awt.event.MouseEvent m) {
                // Az egérmutató pozíciója
                x = m.getX();
                y = m.getY();
                // Az 1. egér gombbal a nagyítandó terület kijelölését
                // végezzük:
                if(m.getButton() == java.awt.event.MouseEvent.BUTTON1 ) {
                    // A nagyítandó kijelölt területet bal felső sarka: (x, ↩
                       y)
                    // és szélessége (majd a vonszolás növeli)
                    mx = 0;
                    my = 0;
                    repaint();
                } else {
                    // Nem az 1. egér gombbal az egérmutató mutatta c
                    // komplex számból indított iterációkat vizsgálhatjuk
                    MandelbrotIterációk iterációk =
                            new MandelbrotIterációk(
                            MandelbrotHalmazNagyító.this, 50);
                    new Thread(iterációk).start();
                }
            }
            // Vonszolva kijelölünk egy területet...
            // Ha felengedjük, akkor a kijelölt terület
            // újraszámítása indul:
            public void mouseReleased(java.awt.event.MouseEvent m) {
                if(m.getButton() == java.awt.event.MouseEvent.BUTTON1 ) {
```

```
                    double dx = (MandelbrotHalmazNagyító.this.b
                            - MandelbrotHalmazNagyító.this.a)
                            /MandelbrotHalmazNagyító.this.szélesség;
                    double dy = (MandelbrotHalmazNagyító.this.d
                            - MandelbrotHalmazNagyító.this.c)
                            /MandelbrotHalmazNagyító.this.magasság;
                    // Az új Mandelbrot nagyító objektum elkészítése:
                    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);
                }
            }
        });
        // Egér mozgás események feldolgozása:
        addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
            // Vonszolással jelöljük ki a négyzetet:
            public void mouseDragged(java.awt.event.MouseEvent m) {
                // A nagyítandó kijelölt terület szélessége és magassága:
                mx = m.getX() - x;
                my = m.getY() - y;
                repaint();
            }
        });
    }
    /**
     * Pillanatfelvételek készítése.
     */
    public void pillanatfelvétel() {
        // Az elmentendő kép elkészítése:
        java.awt.image.BufferedImage mentKép =
                new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        java.awt.Graphics g = mentKép.getGraphics();
        g.drawImage(kép, 0, 0, this);
        g.setColor(java.awt.Color.BLACK);
        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();
        // A pillanatfelvétel képfájl nevének képzése:
        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("MandelbrotHalmazNagyitas_");
        sb.append(++pillanatfelvételSzámláló);
        sb.append("_");
        // A fájl nevébe belevesszük, hogy melyik tartományban
        // találtuk a halmazt:
        sb.append(a);
        sb.append("_");
        sb.append(b);
        sb.append("_");
        sb.append(c);
        sb.append("_");
        sb.append(d);
        sb.append(".png");
        // png formátumú képet mentünk
        try {
            javax.imageio.ImageIO.write(mentKép, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * A nagyítandó kijelölt területet jelző négyzet kirajzolása.
     */
    public void paint(java.awt.Graphics g) {
        // A Mandelbrot halmaz kirajzolása
        g.drawImage(kép, 0, 0, this);
        // Ha éppen fut a számítás, akkor egy vörös
        // vonallal jelöljük, hogy melyik sorban tart:
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
        // A jelző négyzet kirajzolása:
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
    }
    /**
     * Hol áll az egérmutató?
     * @return int a kijelölt pont oszlop pozíciója.
     */
    public int getX() {
        return x;
    }
    /**
     * Hol áll az egérmutató?
```

```
     * @return int a kijelölt pont sor pozíciója.
     */
    public int getY() {
        return y;
    }
    /**
     * Példányosít egy Mandelbrot halmazt nagyító obektumot.
     */
    public static void main(String[] args) {
        // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
        // tartományában keressük egy 600x600-as hálóval és az
        // aktuális nagyítási pontossággal:
        new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```

5.7. Vörös Pipacs Pokol/fel a láváig és vissza

Az ehhez használatos kód megtalálható a Mandelbrot nevű mappában, lentrolfel.py néven.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás forrása: A második előadás 17-22. fóliában fellelhető.

```
#ifndef POLARGEN_H
#define POLARGEN_H
#include <cstdlib>
#include <cmath>
#include <ctime>
class PolarGen
{
  public:
  PolarGen ()
  {
  nincsTarolt = true;
  std::srand (std::time (NULL));
  }
  ~PolarGen(){
  }
  double kovetkezo();
  private:
  bool nincsTarolt;
  double Tarolt;
};
#endif
```

Ezután definiáljuk a Polargen osztályt:

```
#include "polargen.h"

double
```

```
PolarGen::kovetkezo ()
{
  if(nincsTarolt){
  double u1, u2, v1, v2, w;
  do{
  u1 = std::rand() / (RAND_MAX + 1.0);
  u2 = std::rand() / (RAND_MAX + 1.0);
  v1 = 2*u1 -1;
  v2 = 2*u2 -1;
  w = v1*v1+v2*v2;
  }
  while( w > 1);
  double r = std::sqrt((-2*std::log(w)) / w);
  #include "polargen.h"
  Tarolt = r*v2;
  nincsTarolt = !nincsTarolt;
  return r*v1;
  }
  }
  else
  {
  nincsTarolt = !nincsTarolt;

  return Tarolt;
  }
}
```

Nem szabad megfeledkeznünk a main függvényünk megírásáról sem:

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

int main(int argc, char **argv){
  PolarGen pg;
  for (int i = 0; i < 10; ++i)
  std::cout <<pg.kovetkezo() << std::endl;

  return 0;
}
```

Nem marad el ezúttal sem a programunk fordítása majd futtatása ami a szokásos módon fog zajlani:

```
hp@b3r3s:~/Dokumentumok$ g++ polar.cpp polargen.h polargen.cpp -o polargen
hp@b3r3s:~/Dokumentumok$ ./polargen
0.581635
-0.330038
-1.53916
0.67204
-1.56939
1.89336
-0.845487
```

```
-1.09789
-0.0786508
-1.26333
```

6.2. LZW

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

Megoldás forrása: Programozó Páternoszter by: Bátfai Norbert

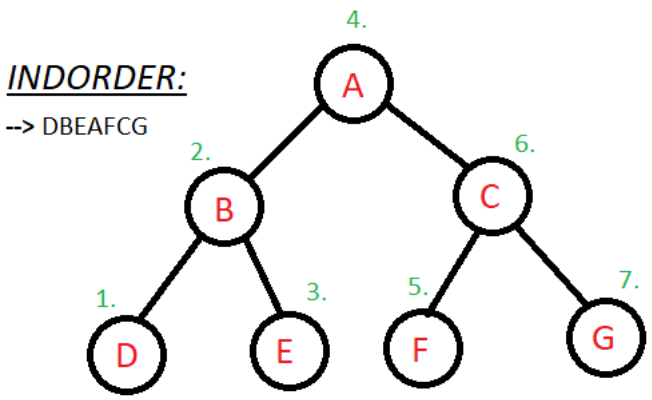
A feladatunk megoldását tartalmazza a 6.6 Mozgató szemantika feladata, illetve a Welch mappában a z.c fájl.

6.3. Fabejárás

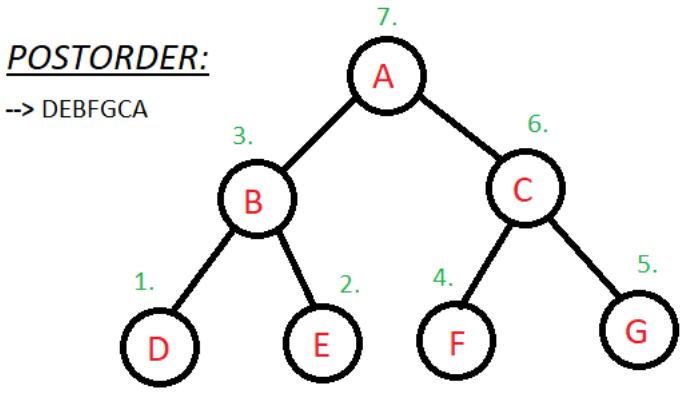
Járd be az előző (inorder bejárású) fát pre- és posztorder is!

Egy fát mint azt már az Adatszerkezetek és algoritmusok órán is elsajátíthattuk, 3 féle képpen tudjuk bejárni. Ezek a bejárások pedig a következők: Inorder, postorder és preorder.

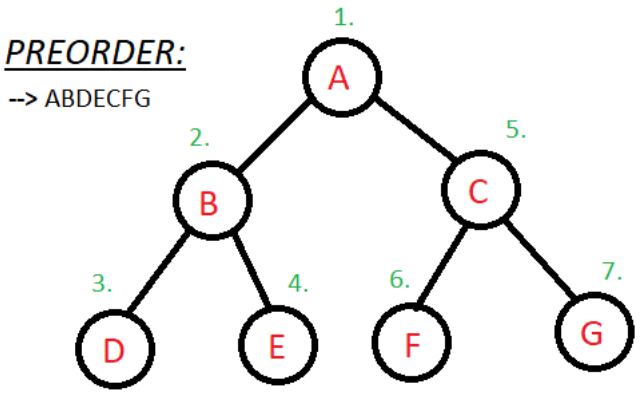
Ezek szemléltetésére igénybe kellet hogy vegyem a hatalmas paint tudásomat. Ha esetleg nem lenne elég egyértelmű, a zöld színnel jelölt számok, az egyes csomópontok beolvasási sorszámát jelöli. (1-től 7-ig)



6.1. ábra. INORDER



6.2. ábra. POSTORDER



6.3. ábra. PREODER

6.4. Tag a gyökér

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

Megoldás videó: https://www.youtube.com/watch?v=_mu54BDkqiQ

A feladatunk megoldását tartalmazza a 6.6 Mozgató szemantika feladata.

6.5. Mutató a gyökér

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

Megoldás videó: https://www.youtube.com/watch?v=_mu54BDkqiQ

A feladatunk megoldását tartalmazza a 6.6 Mozgató szemantika feladata.

6.6. Mozgató szemantika

Írj az előző programhoz másoló/mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva, a másoló értékadás pedig a másoló konstruktorra.

Megoldás videó: https://www.youtube.com/watch?v=QBD3zh5OJ0Y

A feladat során csak pár kód csipetet jelenítünk meg de akinek szüksége lenne az egészre, az megtalálja a Welch mappában "LZWBinFa.cpp" néven.

Természetesen ezekben a csipetekben fellelhetőek lesznek az előző feladatokban beígérteket is.

A program, vagyis a fa építése a BinRandTree class létrehozásával kezdődik, azon belül is a Node osztály elhelyezésével egy protected részben. Gondolom senkinek sem árulok el nagy titkot ha felfedem, hogy a node, a csomópontokat jelenti, vagyis ezeknek lehet (max) 1 jobb, illetve 1 bal oldali gyermeke. Természetesen minden ilyen gyermek egy-egy újabb csomópont lesz.

6.4

```
template <typename ValueType>
class BinRandTree {

protected:
    class Node {

    private:
        ValueType value;
        Node *left;
        Node *right;
        int count{0};

        Node(const Node &);
        Node & operator=(const Node &);
        Node(Node &&);
        Node & operator=(Node &&);


    public:
        Node(ValueType value, int count=0): value(value), count(count),  ↩
           left(nullptr), right(nullptr) {}
        ValueType getValue() const { return value; }
        Node * leftChild() const { return left; }
        Node * rightChild() const { return right; }
        void leftChild(Node * node){ left = node; }
        void rightChild(Node * node){ right = node; }
        int getCount() const { return count; }
        void incCount() { ++count; }
    };
```

6.5

Mint azt már említettük, kelleni fognak a mutatók is, ezt a szerepet fogja betölteni a " * ". A *root értelemszerűen a gyökérmutatónka lesz, míg a *treep jelzi, hogy hol is járunk éppen.

```
    Node *root;
    Node *treep;
    int depth{0};
```

Majd amiről a 6.6. fejezetünk is szól, jöhetnek a mozgató és másoló konstruktorok.

```
public:
    BinRandTree(Node *root = nullptr, Node *treep = nullptr): root(root),  ↩
       treep(treep) {
        std::cout << "BT ctor" << std::endl;
    }

    BinRandTree(const BinRandTree & old) {
        std::cout << "BT copy ctor" << std::endl;

        root = cp(old.root, old.treep);

    }

    Node * cp(Node *node, Node *treep)
    {
        Node * newNode = nullptr;

        if(node)
        {
            newNode = new Node(node->getValue(), 42 /*node->getCount()*/);

            newNode->leftChild(cp(node->leftChild(), treep));
            newNode->rightChild(cp(node->rightChild(), treep));

            if(node == treep)
                this->treep = newNode;
        }

        return newNode;
    }


    BinRandTree & operator=(const BinRandTree & old) {
        std::cout << "BT copy assign" << std::endl;

        BinRandTree tmp{old};
        std::swap(*this, tmp);
        return *this;
    }

    BinRandTree(BinRandTree && old) {
```

```
        std::cout << "BT move ctor" << std::endl;

        root = nullptr;
        *this = std::move(old);
    }

    BinRandTree & operator=(BinRandTree && old) {
        std::cout << "BT move assign" << std::endl;

        std::swap(old.root, root);
        std::swap(old.treep, treep);

        return *this;
    }

    ~BinRandTree(){
        std::cout << "BT dtor" << std::endl;
        deltree(root);
    }
    BinRandTree & operator<<(ValueType value);
    void print(){print(root, std::cout);}
    void printr(){print(*root, std::cout);}
    void print(Node *node, std::ostream & os);
    void print(const Node &cnode, std::ostream & os);
    void deltree(Node *node);

    Unirand ur{std::chrono::system_clock::now().time_since_epoch().count(), ↩
        0, 2};

    int whereToPut() {

            return ur();
    }

};
```

6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid

A feladatunk tulajdonképpen nem túl bonyolult. Eddig ugyebár csak 3x3x3 érzékelőink voltak vagyis Steve csak ezekről a blokkokról kapott információt. Ezt most azonban kiterjesztjük 5x5x5-re, az nb4tf4i_d.xml belepiszkálásával. Mindenképp ennek az átírásával kell kezdenünk, hisz a python fájlunk is ezt fogja majd használni.

```
<ObservationFromGrid>
    <Grid name="nbr5x5">
        <min x="-2" y="-2" z="-2"/>
        <max x="2" y="2" z="2"/>
    </Grid>
```

```
</ObservationFromGrid>
```

Amint ez kész, kezdődhet a pipacs szedés egy jóval nagyob griddel. Azonban ne felejtsük el a programunkon belül is végrehajtani a módosításokat.

```
            if world_state.number_of_observations_since_last_state != 0:

                sensations = world_state.observations[-1].text
                observations = json.loads(sensations)
                nbr5x5x5 = observations.get("nbr5x5", 0)
    print(" 5x5x5 neighborhood of Steve: ", nbr5x5x5)
```

Így ni, hajrá előre Haxorok!

7. fejezet

Helló, Conway!

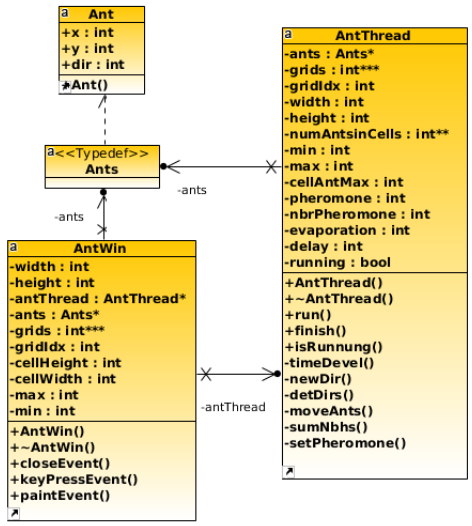
7.1. Hangyaszimulációk

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

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

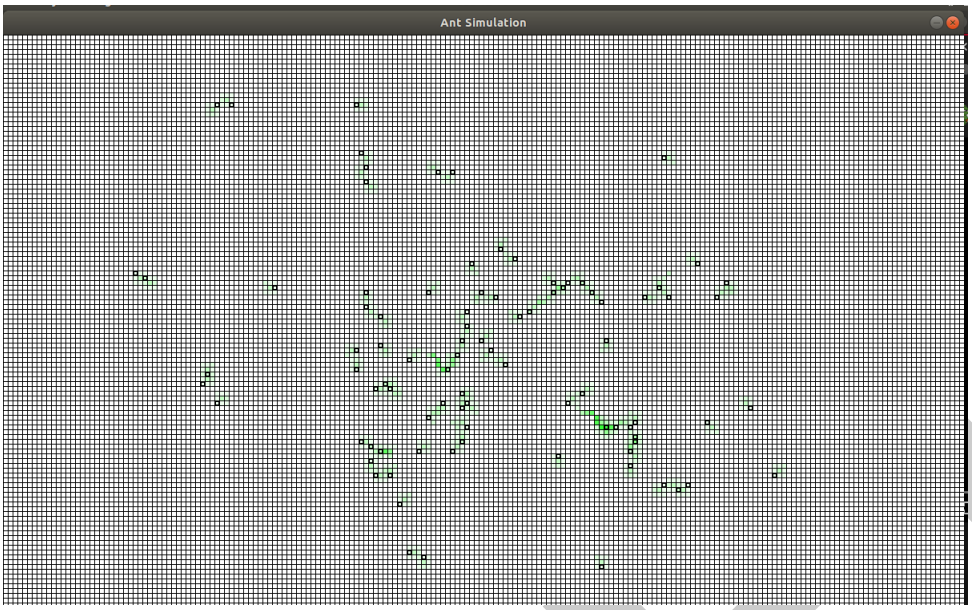
Megoldás forrása: https://gitlab.com/nbatfai/bhax/-/tree/master/attention_raising_2FMyrmecologist

A programunnk nem véletlenül kapta a hangyaszimuláció nevet, ugyanis a kész kódunk ha minden jól sül el, pontosan a hangyák véletlenszerű mozgását próbálja szemléltetni. Mindez egy 2 dimenziós felületen fog zajlani de erről később képet is láthatunk.



7.1. ábra. A hangyaszimulációs program UML diagramja

Ahogy azt a diagrammon is látjuk, a programunk 3 főbb osztályból fog kialakulni ezek pedig névszerint: AntWin, AntThread és az Ant.



7.2. ábra. A szimuláció

7.2. Java életjáték

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

A feladathoz szükséges forráskódunk megtalálható az alább beágyazott linken, illetve a Conway mappában is, "Sejtautomata.java" néven.

Megoldás forrása: https://regi.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apb.html#conway

" Itt egy sejt egy sejttér eleme, a sejt állapota lehet élő vagy halott. A diszkrét időben működő sejttér adott sejtjének állapotát a következő időpillanatban a következő átmeneti szabályok alapján számolhatjuk ki:

Élő sejt élő marad, ha kettő vagy három élő szomszédja van, különben halott lesz.

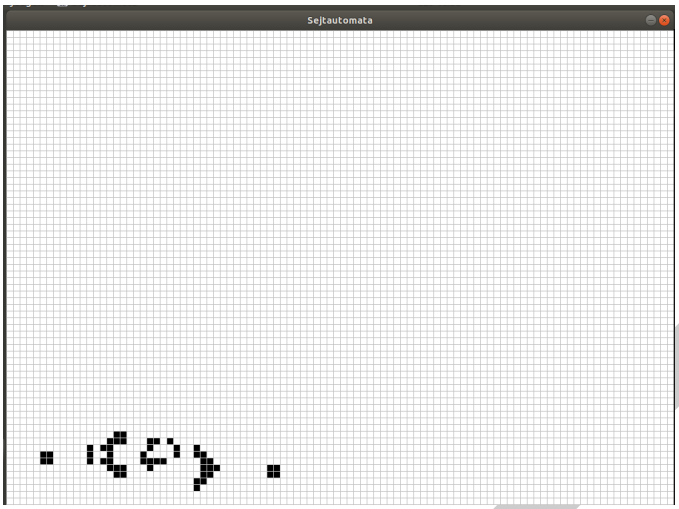
Halott sejt halott marad, ha három élő szomszédja van, különben élő lesz.

Ezeket az átmeneti szabályokat az időFejlődés() függvényben fogalmaztuk meg. "

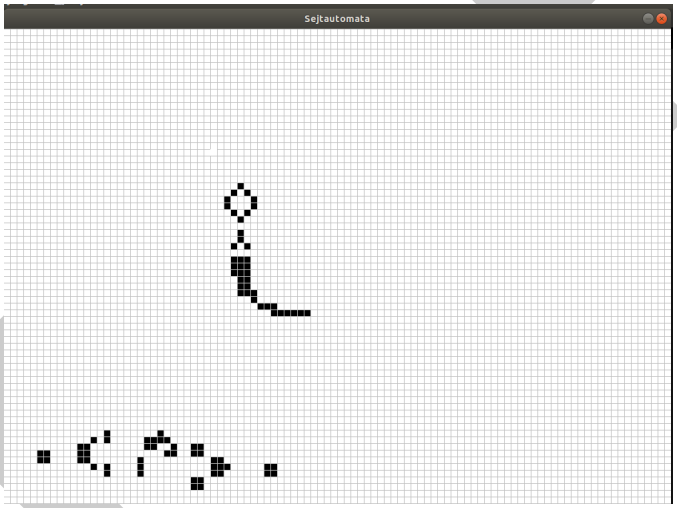
A programunk fordítás illetve futtatása bizonyára senkinek sem jelent már gondot de azért parancsorosn belül gyorsan bemutatom.

```
hp@b3r3s:~/Dokumentumok$ javac Sejtautomata.java
hp@b3r3s:~/Dokumentumok$ java Sejtautomata
```

Ha minden rendeltetésszerűen működik, az alsó képet kell kapnunk eredményül, természetesen nem PNG formátumba hanem egy folyamatosan futó java program verziót, amibe ha szeretnénk mi is belenyúlhatunk a kurzor segítségével.



7.3. ábra. Életjáték



7.4. ábra. Életjáték, miután közbeavatkoztunk

7.3. Qt C++ életjáték

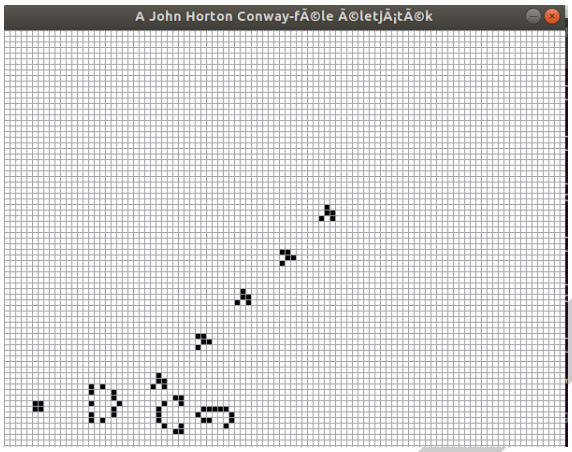
Most Qt C++-ban!

Megoldás videó és forrása: https://bhaxor.blog.hu/2018/09/09/ismerkedes_az_eletjatekkal

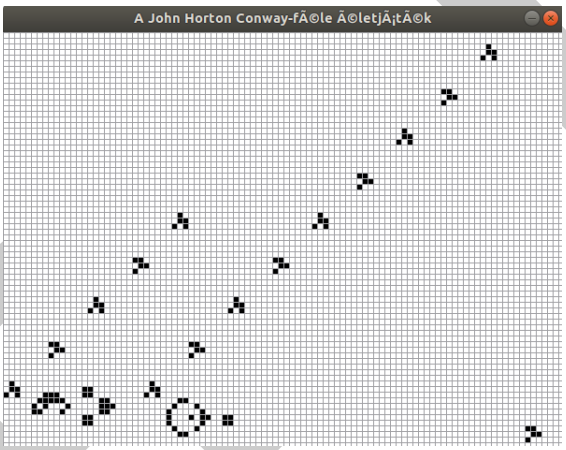
A programunk futtatására először is be kell lépnünk (parancssoron belül) abba a mappánkba ahol korábban létrehoztuk a .cpp illetve a header fájljainkat. Amint ez megtörtént, kiadunk egy qmake-qt4 -project parancsot ami remélhetőleg létrehoz egy .pro fájlt. Ezután erre ráküldhetjük a qmake-t ami egy újabb fájlt fog létrehozni, mégpedig a MakeFile-t. Előtte azonban érdemes egy **apt-get update** illetve egy **sudo apt-get install libqt4-dev** parancsot is lefuttatni (személyes tapasztalat).

Végül de nem utolsósorban egy dolgunk maradt, mégpedig lefuttatni, (az én esetemben) a Sejtauto fájlt.

Az eredmény pedig magáért beszél:



7.5. ábra. Qt C++ életjáték



7.6. ábra. Qt C++ életjáték, kis idő eltelte után

Gyors magyarázat by: Wikipédia

A négyzetrács mezőit celláknak, a korongokat sejteknek nevezzük. Egy cella környezete a hozzá legközelebb eső 8 mező (tehát a cellához képest „átlósan” elhelyezkedő cellákat is figyelembe vesszük, feltesszük hogy a négyzetrácsnak nincs széle). Egy sejt/cella szomszédjai a környezetében lévő sejtek. A játék körökre osztott, a kezdő állapotban tetszőleges számú (egy vagy több) cellába sejteket helyezünk. Ezt követően a játékosnak nincs beleszólása a játékmenetbe. Egy sejttel (cellával) egy körben a következő három dolog történhet:

1. A sejt túléli a kört, ha két vagy három szomszédja van.
2. A sejt elpusztul, ha kettőnél kevesebb (elszigetelődés), vagy háromnál több (túlnépesedés) szomszédja van.
3. Új sejt születik minden olyan cellában, melynek környezetében pontosan három sejt található.

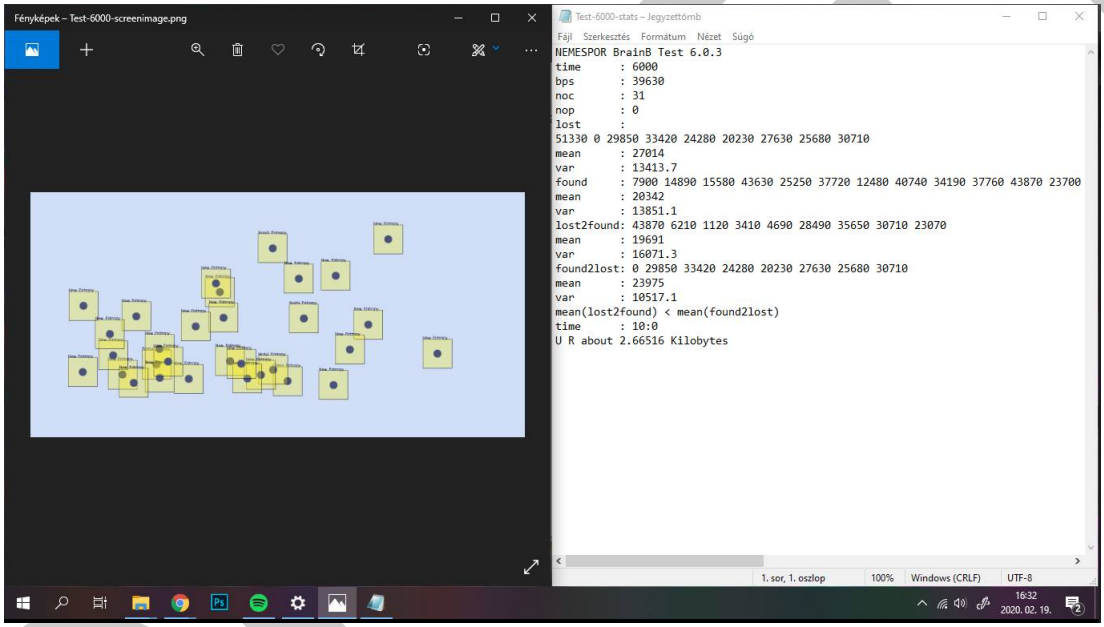
7.4. BrainB Benchmark

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

Megoldás videó: https://www.twitch.tv/videos/139186614

Ez a feladatunk tulajdonképpen egy teszt lenne, ami azt méri, hogy mennyire tudjuk követni a karakterünket (jelen esetben egy négyzetet) az egyre növekedő káoszban. Akik játszottak már a League of Legends játékkal, azoknak biztosan ismerős az az 5 az 5 elleni teamfight helyzet, hogy nyomja a spelleket bízván abba, hogy azok be is találnak hiába sejtelme sincs a hőse helyzetéről a nagy kavalkádban. Ez a program pontosan az ilyen helyzetek leküzdésének fejlesztésére szolgál.

A mi feladatunk egész pontosan, hogy a saját négyzetünket kövessük az egérmutatónkkal miközben nyomva tartjuk a balegérgombunkat. Erre időlimit van ami maximum 10 perc. Végül kapunk egy eredményt abból számítva, hogy hányszor vesztettük el a négyzetünket vagy éppen mennyire tudtuk követni.



7.7. ábra. BrainB teszt saját eredmény

7.5. Vörös Pipacs Pokol/19RF

Megoldás videó: Saját kóddal 22 pipacs.

Megoldás forrása: Fentrolle22.py kód.

A kód amit használtam a RFH 3 során, illetve ezzel volt 22 pipacs a saját rekordom.

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

# Copyright (c) 2016 Microsoft Corporation
```

```
#
# Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
# associated documentation files (the "Software"), to deal in the Software without restriction,
# including without limitation the rights to use, copy, modify, merge, publish, distribute,
# sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all copies or
# substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
# NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# ------------------------------------------------------------------------------------------------


# Tutorial sample #2: Run simple mission using raw XML

# Added modifications by Norbert Bátfai (nb4tf4i) batfai.norbert@inf.unideb.hu, mine.ly/nb4tf4i.1
# 2018.10.18, https://bhaxor.blog.hu/2018/10/18/malmo_minecraft
# 2020.02.02, NB4tf4i's Red Flowers, http://smartcity.inf.unideb.hu/~norbi/NB4tf4iRedFlowerHell


from builtins import range
import MalmoPython
import os
import sys
import time
import random
import json
import math

if sys.version_info[0] == 2:
    sys.stdout = os.fdopen(sys.stdout.fileno(), 'w', 0)  # flush print output immediately
```

```
else:
    import functools
    print = functools.partial(print, flush=True)

# Create default Malmo objects:

agent_host = MalmoPython.AgentHost()
try:
    agent_host.parse( sys.argv )
except RuntimeError as e:
    print('ERROR:',e)
    print(agent_host.getUsage())
    exit(1)
if agent_host.receivedArgument("help"):
    print(agent_host.getUsage())
    exit(0)

# -- set up the mission -- #
missionXML_file='nb4tf4i_d.xml'
with open(missionXML_file, 'r') as f:
    print("NB4tf4i's Red Flowers (Red Flower Hell) - DEAC-Hackers Battle ↩
       Royale Arena\n")
    print("NB4tf4i vörös pipacsai (Vörös Pipacs Pokol) - DEAC-Hackers ↩
       Battle Royale Arena\n\n")
    print("The aim of this first challenge, called nb4tf4i's red flowers, ↩
       is to collect as many red flowers as possible before the lava flows ↩
       down the hillside.\n")
    print("Ennek az első, az nb4tf4i vörös virágai nevű kihívásnak a célja ↩
       összegyűjteni annyi piros virágot, amennyit csak lehet, mielőtt a ↩
       láva lefolyik a hegyoldalon.\n")
    print("Norbert Bátfai, batfai.norbert@inf.unideb.hu, https://arato.inf. ↩
       unideb.hu/batfai.norbert/\n\n")
    print("Loading mission from %s" % missionXML_file)
    mission_xml = f.read()
    my_mission = MalmoPython.MissionSpec(mission_xml, True)
    my_mission.drawBlock( 0, 0, 0, "lava")


class Hourglass:
    def __init__(self, charSet):
        self.charSet = charSet
        self.index = 0
    def cursor(self):
        self.index=(self.index+1)%len(self.charSet)
        return self.charSet[self.index]

hg = Hourglass('|/-\|')

class Steve:
    def __init__(self, agent_host):
```

```
        self.agent_host = agent_host
        self.x = 0
        self.y = 0
        self.z = 0
        self.yaw = 0
        self.pitch = 0

    def run(self):
        world_state = self.agent_host.getWorldState()

        for x in range (6):
            self.agent_host.sendCommand("move 1")
            time.sleep(.1)
        for y in range (21):
            self.agent_host.sendCommand("jumpmove 1")
            time.sleep(.1)
            self.agent_host.sendCommand("move 1")
            time.sleep(.1)
        self.agent_host.sendCommand("look 1")
        time.sleep(.1)
        self.agent_host.sendCommand("look 1")
        time.sleep(.1)
        self.agent_host.sendCommand("turn -1")
        time.sleep(.1)

        # Loop until mission ends:
        while world_state.is_mission_running:
            if world_state.number_of_observations_since_last_state != 0:
                sensations = world_state.observations[-1].text
                observations = json.loads(sensations)
                nbr3x3x3 = observations.get("nbr3x3", 0)

                if "Yaw" in observations:
                    self.yaw = int(observations["Yaw"])
                if "Pitch" in observations:
                    self.pitch = int(observations["Pitch"])
                if "XPos" in observations:
                    self.x = int(observations["XPos"])
                if "ZPos" in observations:
                    self.z = int(observations["ZPos"])
                if "YPos" in observations:
                    self.y = int(observations["YPos"])

                if "LineOfSight" in observations :
                    lineOfSight = observations["LineOfSight"]
                    self.lookingat = lineOfSight["type"]
                print("\(    O__O )/: ", self.lookingat)


                if nbr3x3x3[13] == "red_flower":
```

```
                    print("     KAPD KI!")
                    self.agent_host.sendCommand("attack 1")
                    time.sleep(1.5)
                    self.agent_host.sendCommand("jumpmove -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("move -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("strafe -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("strafe -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("move 1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("move 1")
                    time.sleep(.1)

                self.agent_host.sendCommand("move 1")
                time.sleep(.1)

                if nbr3x3x3 [16] =='dirt' and nbr3x3x3[12] =='dirt':
                    self.agent_host.sendCommand("turn -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("move 1")
                    time.sleep(.1)
                if nbr3x3x3[16] =='dirt' and nbr3x3x3[14] =='dirt':
                    self.agent_host.sendCommand("turn -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("move 1")
                    time.sleep(.1)
                if nbr3x3x3[14]=='dirt'  and nbr3x3x3[10]=='dirt':
                    self.agent_host.sendCommand("turn -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("move 1")
                    time.sleep(.1)
                if nbr3x3x3[10]=='dirt'  and nbr3x3x3[12]== 'dirt':
                    self.agent_host.sendCommand("turn -1")
                    time.sleep(.1)
                    self.agent_host.sendCommand("move 1")
                    time.sleep(.1)



            world_state = self.agent_host.getWorldState()

num_repeats = 1
for ii in range(num_repeats):

    my_mission_record = MalmoPython.MissionRecordSpec()

    # Attempt to start a mission:
```

```
    max_retries = 6
    for retry in range(max_retries):
        try:
            agent_host.startMission( my_mission, my_mission_record )
            break
        except RuntimeError as e:
            if retry == max_retries - 1:
                print("Error starting mission:", e)
                exit(1)
            else:
                print("Attempting to start the mission:")
                time.sleep(2)

    # Loop until mission starts:
    print("   Waiting for the mission to start ")
    world_state = agent_host.getWorldState()

    while not world_state.has_mission_begun:
        print("\r"+hg.cursor(), end="")
        time.sleep(0.15)
        world_state = agent_host.getWorldState()
        for error in world_state.errors:
            print("Error:",error.text)

    print("NB4tf4i Red Flower Hell running\n")
    steve = Steve(agent_host)
    steve.run()

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

8. fejezet

Helló, Schwarzenegger! (4passzolás)

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

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin! (4Passzolás)

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Tanulságok, tapasztalatok, magyarázat...

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

Tanulságok, tapasztalatok, magyarázat...

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[PICI KONYV]

Kicsi a bors de erős!

Legelső sorban be kell, hogy valljam, az iskolás éveim alatt akárhányszor kötelező olvasmányt kaptunk, azt leginkább a hátam közepére kívántam volna mivel tipikusan olyan könyveket kellet olvasnunk ami nem az én érdeklődési körömbe tartozott. Sosem értettem, hogy miért nem adnak meg a gyereknek mondjuk 10 témát amiben keresgélhet és választani kell egy vagy több könyvet, (természeteset az általa kiválasztott témában) amit a saját szórakoztatására olvashat.

Ezúttal már láttam a fényt az alagút végén mivel habár az olvasás iránti szeretetem nem nőtt azóta de most-már legalább olyan témáról kellet olvasnom ami érdekelt és tudtam, hogy a könyv tartalma a tanulmányi évim során még akár a hasznomra is válhat. Ez a meglátásom többynire be is igazolódott és miközben olvastam a könyvet, nem jött elő az régi és jólismert szituáció miszerint: csak olvasok és olvasok majd egy 10 perc után realizálom, hogy az eddigi elolvasott 'x' oldal teljesen kiesett és nem emlkészek belőle semmire.

Összességében, habár számunkra nem az egész könyv volt feladva de biztos vagyok benne hogy a könyv töbi része is tartalmaz hasznos információkat és azoknak akik kicsit szeretnének belelátni a dolgokba kicsit mélyebben annak nyugodt szívvel tudom ajánlani. Mindamellet, hogy érdekesnek találtam a könyvet, ezzel egyidőben egyértelműen tudom ajánlani azoknak az embereknek is akik insomnia-val szenvednek ugyanis egy fárasztóbb nap után saját tabasztalatból mondhatom, hogy meglehetősen könnyen és gyorsan el tudtam mellette aludni.

10.2. Bevezetés a mobilprogramozásba

[MOBIL]

A könyv ahogy azt a címéből is ésszerűen kikövetkeztethetjük, a mobilprogramozásról és amellett még a mobilok tulajdonságairól, funkcióiról és típusairól szól. A könyv olvasás közben azért néha elmosolyodunk az olyan kijelentéseken min, hogy a "Ez a legbővebb kategória, a ma kapható készülékek jelentős része idetartozik". Nem, a könyv írója itt nem a mai okostelefonokra gondolt, hanem az akkoriban mindenki álatal használt egyszerűbb, általános mobilkészülékek. Ezek mára már elfeledetté váltak, csak nagyok

kevesek által használt a technológiai visszamaradottsága miatt. Természetesen 5,10,20 év múlva a mi fiaink vagy később az unokáink valószínűleg ugyan így fognak érezni mint mi és röhögve fogják olvasni ezt a kis olvasónaplót.

Mindemmellett a könyv rengeteg hasznos dolgot leír és segít megértettni azokkal akik a közeljövőben a mobilprogramozásban képzelik el a jövőjüket. Számomra is nagyon vonzó ez a téma ugyanis mint mindenki énis napi rendszerességgel használom a telefonomat, legyen az zenehallgatás, olvasás, média és természetesen itt se felejthetjük el a játékokat, amik egyre inkább modernizálódnak és lassan már nem fogunk tudni különbséget tenni egy film és egy telefonos játék grafikája között. Habár engem nagyon kevés telefonos játék tud hosszabb időre lekötni, ennek ellenére ha rendelkeznék a megfelelő tudással programozás terén és felkérne egy cég, hogy segítsek nekik kifejleszteni a legújabb játékukat, valószínűleg szó nélkül és örömmel belemennék.

Természetesen azonban nem csak a játékokra kell gondolni amikor a programozásról beszélünk, ugyanis ez egy sokkal tágabb kör. Mindaz ami számunkra ma már természetesnek tűnhet a telefonunk használata közben, gondolok itt például az érintőképernyő, Face ID, wifi folytonos működésére, elérésére.

10.3. A C programozási nyelv

[KERNIGHANRITCHIE]

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

10.4. Szoftverfejlesztés C++ nyelven

[BMECPP]

III. rész

Második felvonás

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

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Programozási nyelvek

[PICI KONYV] Juhász, István, *Magas szintű programozási nyelvek 1*, mobiDIÁK könyvtár , 2008.

11.4. Általános

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

11.5. C

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

11.6. C++

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

11.7. Lisp

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

11.8. Mobilprogramozás (Java, Python)

[MOBIL] Ekler, Péter, Forstner, Bertalan, Ã©s Kelényi, Imre, *Bevezetés a mobilprogramozásba*, Szak Kiadó Kft. , 2008.

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.