Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

KÖZREMÜKÖDTEK			
	CÍM : Univerzális programozás		
HOZZÁJÁRULÁS	*NÉV*	*DÁTUM*	*ALÁÍRÁS*
ÍRTA	Oszuszki Richard és Bátfai Norbert	2019. szeptember 25.	

VERZIÓTÖRTÉNET			
VERZIÓ	DÁTUM	LEÍRÁS	NÉV
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-05-09	A könyv elkészült.	O.Richard

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés 1

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

II. Tematikus feladatok 3

2. Helló, Turing! 5
- 2.1. Végtelen ciklus . . . 5
- 2.2. Lefagyott, nem fagyott, akkor most mi van? . . . 6
- 2.3. Változók értékének felcserélése . . . 8
- 2.4. Labdapattogás . . . 8
- 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 11
- 2.6. Helló, Google! . . . 12
- 2.7. 100 éves a Brun tétel . . . 13
- 2.8. A Monty Hall probléma . . . 14

3. Helló, Chomsky! 15
- 3.1. Decimálisból unárisba átváltó Turing gép . . . 15
- 3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 16
- 3.3. Hivatkozási nyelv . . . 17
- 3.4. Saját lexikális elemző . . . 18
- 3.5. l33t.l . . . 19
- 3.6. A források olvasása . . . 21
- 3.7. Logikus . . . 23
- 3.8. Deklaráció . . . 23

4. Helló, Caesar! 27
4.1. int ** háromszögmátrix . . . 27
4.2. C EXOR titkosító . . . 29
4.3. Java EXOR titkosító . . . 30
4.4. C EXOR törő . . . 31
4.5. Neurális OR, AND és EXOR kapu . . . 31
4.6. Hiba-visszaterjesztéses perceptron . . . 39

5. Helló, Mandelbrot! 41
5.1. A Mandelbrot halmaz . . . 41
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 43
5.3. Biomorfok . . . 45
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 45
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 46
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 50

6. Helló, Welch! 51
6.1. Első osztályom . . . 51
6.2. LZW . . . 52
6.3. Fabejárás . . . 53
6.4. Tag a gyökér . . . 53
6.5. Mutató a gyökér . . . 53

7. Helló, Conway! 55
7.1. Hangyaszimulációk . . . 55
7.2. Qt C++ életjáték . . . 61
7.3. BrainB Benchmark . . . 62

8. Helló, Schwarzenegger! 64
8.1. Szoftmax Py MNIST . . . 64
8.2. Szoftmax R MNIST . . . 66
8.3. Mély MNIST . . . 67
8.4. Deep dream . . . 67
8.5. Minecraft-MALMO . . . 74

9. Helló, Chaitin! 75

9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 75

9.2. Weizenbaum Eliza programja . . . 76

10. Helló, Gutenberg! 77

10.1. Programozási alapfogalmak . . . 77

10.2. Programozás bevezetés . . . 78

10.3. Programozás . . . 79

III. Második felvonás 80

11. Helló, Berner-Lee! 81

11.1. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven VS. Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II . . . 81

11.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. . . . 83

12. Helló, Arroway! 85

12.1. OO szemlélet . . . 85

12.2. "Gagyi" . . . 86

12.3. Yoda . . . 87

12.4. Kódolás from scratch . . . 88

IV. Irodalomjegyzék 91

12.5. Általános . . . 92

12.6. C . . . 92

12.7. C++ . . . 92

12.8. Lisp . . . 92

Ábrák jegyzéke

3.1. Példa decimálisból unárisba átváltó Turing gépre. 15

5.1. A Mandelbrot halmaz a komplex síkon 43

8.1. Minta 65
8.2. A képen a kutyusom láththató, Viliem 72
8.3. Itt szintén ő, a deep dream változata 73

Előszó

Amikor programozónak terveztem állni, ellenezték a környezetemben, mondván, hogy kell szövegszerkesztő meg táblázatkezelő, de az már van... nem lesz programozói munka.

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz allokálni igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Mindenesetre a programozás a teoretikus kultúra csúcsa. A GNU mozgalomban látom annak garanciáját, hogy ebben a szellemi kalandban a gyerekeim is részt vehessenek majd. Ezért programozunk.

Hogyan forgasd

A könyv célja egy stabil programozási szemlélet kialakítása az olvasóban. Módszere, hogy hetekre bontva ad egy tematikus feladatcsokrot. Minden feladathoz megadja a megoldás forráskódját és forrásokat feldolgozó videókat. Az olvasó feladata, hogy ezek tanulmányozása után maga adja meg a feladat megoldásának lényegi magyarázatát, avagy írja meg a könyvet.

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyerekeknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogy lássuk mást is) példával.

Hogyan nyomjuk?

Rántsd le a https://gitlab.com/nbatfai/bhax git repót, vagy méginkább forkolj belőle magadnak egy sajátot a GitLabon, ha már saját könyvön dolgozol!

Ha megvannak a könyv DocBook XML forrásai, akkor az alább látható **make** parancs ellenőrzi, hogy „jól formázottak” és „érvényesek-e” ezek az XML források, majd elkészíti a dblatex programmal a könyved pdf változatát, íme:

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml  ↩
   --noout
output.xml validates
rm -f output.xml
dblatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
================================================
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dblatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

Ha minden igaz, akkor most éppen ezt a legenerált `bhax-textbook-fdl.pdf` fájlt olvasod.

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

A vegtelen ciklusban jelezzuk az operacios rendszernek a `sleep(100)` rendszerhivassal, hogy a kevetkezo 100 milliszekundumra nem szeretnenk processzoridot kapni, ezaltal a processzorhasznalat megkozelitoleg 0 szazalek lesz:

```
#include <unistd.h>   // sleep()

int main()
{
  for (;;) {
    sleep(100);
  }

  return 0;
}
```

Processzorhasználat 100 százalék, ezt egy egyszerű végtelen ciklussal elérhetjük:

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

  return 0;
}
```

Az összes processzor 100 százalékon fut, az OpenMp függvénykőnyvtár használatának köszönhetően, OpenMP egy igen magas szintű könyvtár, mely szinte mindent meg csinál helyettűnk:

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

int main(void)
{
int x=0;
#pragma omp parallel
while (x<1) {
}
}
///gcc -fopenmp 3.c -o 3 <--így futtasd...
```

Beszámoló: Abban az esetben ha egy üres ciklusmaggal ellátott végtelen ciklust futtatunk, a processzor az összes rendelkezésre álló processzor időt ki osztja, ennek köszönhető a 100 százalékos leterheltség, de ha bele helyezzük a `sleep(100)` rendszerhívást akkor jelzi, hogy a következő 100 milliszekundumra nem tart igényt a processzoridőre, ezáltal nem is kap.

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

```
true
```

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Alan Turing, a probléma vizsgálása idején bizonyította, egy matematika bizonyítás segítségével, hogy ilyen programot nem lehet létre hozni, míg esetleg kisebb programnál szemre meg lehet állapítani, de egy bonyolultabbnál kiszámíthatatlan...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

```
#include <stdio.h>

int main()
{

    int a = 7;
    int b = 5;


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

    int c = a;
    a = b;
    b = c;

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

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

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

}
```

Az *a* és *b* változók értékét megcserélljük az *XOR* operátor segítségével, segédváltozók nélkül! Az *XOR* operátor bitenként végzi el a műveleteket az értékeken (*XOR* szabálya, ha kétszer megszorozzuk ugyan azt akkor át vált az eredetibe).

2.4. Labdapattogás

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

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

Megoldás forrása:

Labda pattogtatás, if-ekkel:

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

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

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, "O" );

        refresh ();
        usleep ( 100000 );

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

        if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
            xnov = xnov * -1;
        }
        if ( x<=0 ) { // elerte-e a bal oldalt?
            xnov = xnov * -1;
        }
        if ( y<=0 ) { // elerte-e a tetejet?
            ynov = ynov * -1;
        }
        if ( y>=my-1 ) { // elerte-e a aljat?
            ynov = ynov * -1;
        }

    }

    return 0;
}
```

Ugyan ez a feladat, csak if, vagy bármiféle logikai utasítás vagy kifejezés nasználata nélkül:

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

int main()
{
  int xj = 0, xk = 0, yj = 0, yk = 0, mx = 0, my = 0, h = 0, w =
      0;

  initscr();
  cbreak();
  noecho();
  nodelay(stdscr, TRUE);
  getmaxyx(stdscr, h, w);
  mx = w * 2;
  my = h * 2;

  while (true) {
    xj = (xj - 1) % mx;
    xk = (xk + 1) % mx;

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

    clear();

    mvprintw(abs((yj + (my - yk)) / 2),
       abs((xj + (mx - xk)) / 2), "o");

    refresh();
    usleep(50000);
  }

  return 0;
}
```

A fő feladatunk ennél a problémánál, hogy olyan függvényt találjunk, ami le írja a labda pattogását, de mégsem vesz fel olyan értéke, ko-ordinátát, amely kivül esik a megjeleníthető kordinátákon, azaz a látható, konzolon belül pattog a "labda".

Tökéletes megoldás, ha a `ncurses` függvénykönyvtárat hívjuk segítségül, mely a terminálos interfészek (TUI-k) létrehozására lett készítve.

```
xj = (xj - 1) % mx;
```

Az xj változó értéke a a maradéka legyen a saját értékénél egyel kisebb értéknek és a terminál szélességének a kétszeresének a hányadosával.

```
xk = (xk + 1) % mx;
```

Ugyanezt xk változóval is.

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

Itt ugyanezt az y tengelyen is.

```
clear();
```

Képernyőt letisztítjuk, hogy ne maradjon fen az előző művelet labdája, így mindig tiszta lappal kezdünk.

```
mvprintw(abs((yj + (my - yk)) / 2),
abs((xj + (mx - xk)) / 2), "o");
```

Ezekben a sorokban rajzoljuk meg a labdát, itt használjuk a `mvprintw` függvényt, ami az adott karaktet, jelen esetben az *o*-t, kiija a képernyőre a megadott x és y ko-ordinátáknak megfelelően.

```
refresh();
usleep(50000);
```

Megtörténik a kiirás, majd várunk 50000 mikroszekundumnyi-t.

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:

Megoldás forrása:

```
#include <stdio.h>

int main()
{
  unsigned long int a = 1, count = 0;

  do
    count++;
  while (a <<= 1);

  printf("Szóhossz: %d.\n", count);
  return 0;
}
```

Ahoz hogy meghatározzuk a szóhoszt, kell használnunk először is egy `int` típusu változót, melynek 1-es értéket adunk, aztán shifteljük balra, ameddig csak lehet, majd megszámoljuk, hogy hány shiftelést tudtunk végre hajtani, és ez lessz a szóhossz az adott számítógépen.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

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

void kiir(double tomb[], int db);
double tavolsag(double pagerank[], double pagerank_temp[], int db);

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

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

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

    for (int i = 0; i < 4; i++) {
      double tmp = 0.0;

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

    if (tavolsag(PR, PRv, 4) < 0.000001)
      break;

  }

  kiir(PR, 4);

  return 0;
}
```

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

}

double tavolsag(double pagerank[], double pagerank_temp[], int db)
{
  double tav = 0.0;

  for (int i = 0; i < db; i++) {
    tav +=
        (pagerank[i] - pagerank_temp[i]) * (pagerank[i] -
              pagerank_temp
              [i]);
  }

  return sqrt(tav);
}
```

Eredet: Az algoritmust a googlenel fejlesztette, Larry Page és Sergey Brin, az algoritmus arra az elméletre épül, hogy egy oldal minnél több oldalra mutat, annak az értéke annál nagyobb, nyilván ez nem ennyire egyszerű, azt is figyelembe kell venni, hogy magára asz ítélt oldalra hány másik oldal mutat, stb...

Magától értetődően a googlenel manapság nem ez az algoritmus futt, hanem egy bustoltabb változata, ami nem publikus, érthető okokból, és ez az algoritmus magában hordoz megannyi kezdeti hibát. (pld zsákuca hiba...)

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

```
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
```

```
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>

library(matlab)

stp <- function(x){

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

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

Prímszámok --> olyan számok, melyek csak önmagukkal és eggyel osztva nem adnak maradékot.

Ikerprímek --> olyan prízmámok, melyek különbsége kettő.

A Brun tétel azt mondja ki, hogy ha vesszük az ikerprímek reciprokát, majd elkezdjük összeadni őket, pl. $\left(\frac{1}{3}+\frac{1}{5}\right)+\left(\frac{1}{5}+\frac{1}{7}\right)+...$, akkor ez a sor egy B_2 (Brun-)konstanshoz fog konvergálni. Bár ez nem oldja meg az ikerprímek számának problémáját, mivel arról nem nyilatkozik, hogy a sor véges, vagy végtelen.

A program az ikerprímeket ábrázolja egy ko-ordináta rendszerben, ahol jól megfigyelhető hogy valóban a B_2 konstanshoz tartunk.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

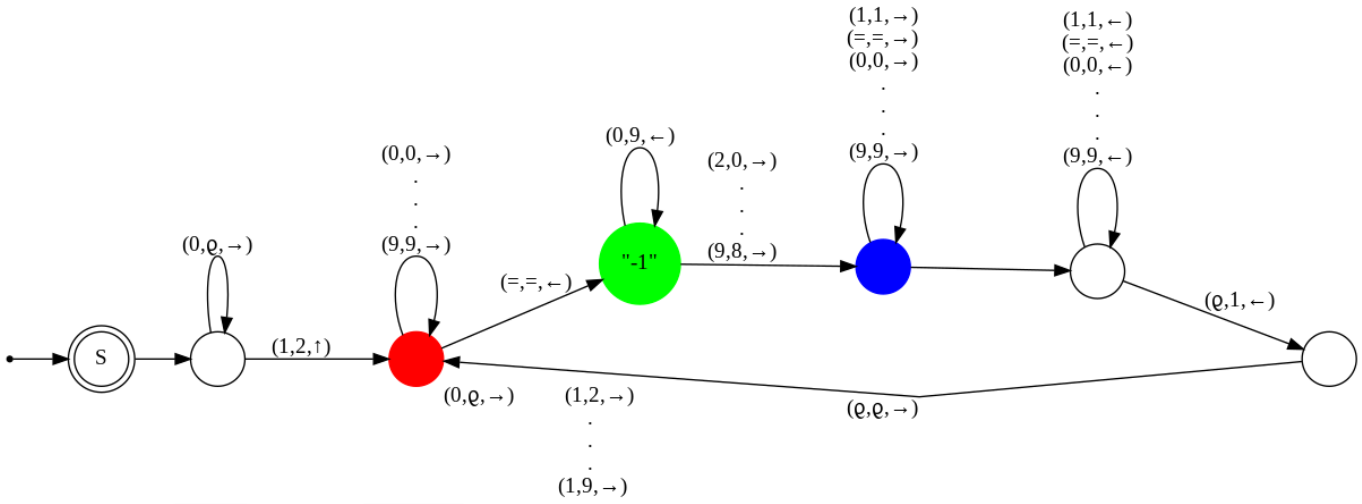
Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan

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

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép



3.1. ábra. Példa decimálisból unárisba átváltó Turing gépre.

Az unáris, azaz az egyes számrendszer a legegyszerűbb számrendszer, amelyel a természetes számokat ábrázolhatjuk. Lényege, hogy az adott számot, az 1 jelölésére alkalmas szimbólum ismétlésével ábrázoljuk. Pld.: `|||||` = 5

Ez a turig gép, ezt az átváltást végzi el.

Forrás:

```
#include <stdio.h>

int main(void)
{
  printf
```

```
      ("Írd be a számot decimálisan: ");

  int in = 0;
  scanf("%d", &in);

  printf("A szám unárisan:");

  for (int i = 0; i < in; ++i)
    (i % 5) ? printf("|") : printf(" |");

  printf("\n");
  return 0;
}
```

? : -> if-then-else operátor -> logikai-kif ? kifejezés : kifejezés

Csak annyi történik, hogy a beklért számnak megfelelően ki írjuk a vonalkákat, ötös csoport szerint.

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

A generatív nyelvtan szabályaival minden nyelv lehetséges jelsorozata előállítható, azaz leírja, hogyan lehet előállítani egy átírási eljárással a kitüntetett kezdő szimbólumból a többi jelsorozatot a szabályokat egymás után alkalmazásával.

Noam Chomsky négy csoportba (típusba) osztotta a felhasznált szabályok formája alapján: általános/mondatszerkezetű (0-s típus), környezetfüggő (1-es típus), környezetfüggetlen (2-es típus) és reguláris (3-as típus).

```
S, X, Y "változók"
a, b, c "konstansok"
S -> abc, S -> aXbc, Xb -> bX, Xc -> Ybcc, bY -> Yb, aY -> aaX, aY -> aa
S-ből indulunk ki
-----------------
  Első:

S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
aYbbcc (aY - aa)
aabbcc
-----------------
  Második:

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 -> aa)
aaabbbccc
```

3.3. Hivatkozási nyelv

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

Van olyan kódcsipet, amely a C egyik verzióval hibátlanul működik, míg más C verzióval hibát jelez.

Példa ahol C99-el lefordul, C89-el nem:

```
include <stdio.h>

int main()
{
  for (int i = 0; i < 5; i++) ;
  return 0;
}
```

Ez az egyik leglátványosabb példa. Fordítsuk ezt le c89 verzióval:

```
gcc -std=c89 c99c89.c
c99c89.c: In function ‘main’:
c99c89.c:5:2: error: ‘for’ loop initial declarations are only allowed in ↩
   C99 or C11 mode
  for (int i = 0; i < 5; i++) ;
  ^~~
c99c89.c:5:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 ↩
    to compile your code
```

Amennyiben a fordító által javasolt opciókkal fordítunk, minden okés:

```
gcc -std=c99 c99.c
```

3.4. Saját lexikális elemző

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

Forrás:

```
%{
// C kod, amit meghagyunk 1-az-1-ben
#include <stdio.h>
%}

%option noyywrap

%%
[-+]?([0-9]*\.[0-9]+|[0-9]+) {
      printf("Valost talaltam: %s\n", yytext);
    }
.|\n  {}
%%

int main(void)
{
  yylex();
  return 0;
}
```

Itt a lexnek köszönhetően csak meg adjuk a definiciót, majd hadjuk, hogy az "óriások" dolgozzanak helyettünk.

Fent a lex definiciónkat láthatjuk: `[-+]?([0-9]*\.[0-9]+|[0-9]+)` Ez a sor azt jelenti, hogy bármely szám, nullától kilencig, akármennyi előfordulása érvényes. `.|\n {}` -> minden más bmenetet ignorálunk

`[-+]?`

'-' vagy '+', egyszer, vagy nullaszor

`(`

Csoport kezdete

`[0-9]*`

Szám nullától kilencig, akárhányszor

`\.`

Szó szerint vett '.' (pont)

`[0-9]+`

Szám nullátor kilencig, akármennyiszer

`|`

Logikai vagy

`[0-9]+`

Szám nullától kilencig, bárhányszor

`)`

Csoport vége

Látszódik, hogy a reguláris kifejezésünk felépítése egyszerű. Elsőnek keresünk előjelet (negatív vagy pozitív) ÉS keressük az egészrészt valamint a törtrészt VAGY csak egészrészt.

Ebből a kódból a lex farag nekün kegy C-kódot...

```
$ lex lex.l
$ gcc lex.yy.c -o lex
$ ./lex
123ad
Valost talaltam: 123
123ad45
Valost talaltam: 123
Valost talaltam: 45
{1}{2}
Valost talaltam: 1
Valost talaltam: 2
3.1415 1.4142
Valost talaltam: 3.1415
Valost talaltam: 1.4142
...
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása:

A leet az egy internetes nyelv, mely azon alapszik, hogy a karaktereket, más karakterekre cseréljük fel, amelyek hasonlóan néznek ki mint az eredeti, ezáltal a mondatok ugyan olyan olvashatóak lesznek, csak a normától eltérő kinézetet adunk neki.

```
$ more l337d1c7.l
%{
  #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", "4", "4"}},
  {'b', {"13", "8", "|3", "|}"}},
  {'c', {"[", "(", "<", "{"}},
  {'d', {"|>", "|)", "|]", "|}"}},
  {'e', {"3", "€", "&#x20a4;", "£"}},
  {'f', {"f", "|=", "ph", "|#"}},
  {'g', {"C-", "6", "[", "[+"}},
  {'h', {"|+|", "4", "|-|", "[-]"}},
  {'i', {"1", "9", "|", "!"}},
  {'j', {"_)", "_7", "_|", "_/"}},
  {'k', {"I{", "|<", "1<", "|{"}},
  {'l', {"][", "1", "|", "|_"}},
  {'m', {"^^", "44", "|V|", "(V)"}},
  {'n', {"][\\][", "|\\\|", "/\\/", "/V"}},
  {'o', {"0", "oh", "()", "[]"}},
  {'p', {"|7", "/o", "|D", "|o"}},
  {'q', {"kw", "9", "O_", "(,)"}},
  {'r', {".-", "I2", "12", "|2"}},
  {'s', {"s", "5", "$", "§"}},
  {'t', {"+", "7", "7", "'|'"}},
  {'u', {"{_}", "|_|", "(_)", "[_]"}},
  {'v', {"\\//", "\\/", "\\/", "\\/"}},
  {'w', {"2u", "VV", "\\/\\/", "(/\\)"}},
  {'x', {"><", "%", ")(", ")("}},
  {'y', {"y", "y", "y", "y"}},
  {'z', {"5", "2", "7_", ">_"}},

  {'0', {"D", " ", "D", "0"}},
  {'1', {"I", "I", "L", "L"}},
  {'2', {"Z", "Z", "e", "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;
}
```

Fordítása:

```
$ lex -o l337d1c7.c l337d1c7.l
$ gcc l337d1c7.c -o l337d1c7 -lfl
$ ./l337d1c7
```

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);
```

Ha a SIGNT jelkezelés nincs ignorálva, akkor a `jelkezelo` végezze a jelkezelést.

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

For ciklus, az i nulla, megnézzük hogy kissebb-e mint 5, minden iterációban növeljük 1-el...

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

Végezzük el ötször...

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

Ez eléggé risky, mivel az i-t már használjuk egyszer és hivatkozunk rá mint tomb i.-re...

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

Nehezen olvashatóság mellett, az összehasonlító operátor helyett, értékadó operátort használunk, emiatt a `&&` operátor jobb oldalán nem egy logikai operandus áll.

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

Ez is hibás kód, mivel az f függvény két int-et kap, de azok kiértékelésének sorrendje nincs meghatározva. Itt az f függvény két intet kap, de a kiértékelési sorrendjük kérdéses...

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

f függvény a-ra való outputját és magát az a-t is irassuk ki.

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

Kiértékelési sorrenddel vannak gondok, nem tudju hogy az e reddeti a-t vagy a módosítottat fogja ki printelni.

3.7. Logikus

Ezekk az Ar nyelvű formulák hogyan néznek ki természetes nyelven?

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

Nézzük meg!

Ezeket a sorokat betesszük egy logikus.tex fájlba, majd lefordítjuk a forrást egy segédprogram használatával:

```
$ pandoc -t latex logikus.tex -o logikus.pdf
```

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})))$
Bármely számnál létezik nála nagyobb prím, azaz a prímek száma végtelen.
```

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})\wedge(SSy \text{ prím})) ↩
   )$
Bármely számnál létezik nála nagyobb prímszám, úgy hogy ennek a prímnek a  ↩
   rákövetkezőjének a rákövetkezője is prím legyen.
Azaz: az ikerprímek száma végtelen.
```

```
$(\exists y \forall x (x \text{ prím}) \supset (x<y)) $
A prímek száma véges.
```

```
$(\exists y \forall x (y<x) \supset \neg (x \text{ prím}))$
Itt a kvantorok nem minden változót kötnek, ezért ekvivalens átlakítások
elvégzése után azt kaptam, hogy : a prímek száma végtelen.
```

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];
  ```

  egészek tömbjére mutató mutató
- ```
  int *h ();
  ```
- ```
  int *(*l) ();
  ```
- ```
  int (*v (int c)) (int a, int b)
  ```
- ```
  int (*(*z) (int)) (int, int);
  ```

Elkészült a program ami a fentemlített elemeket tartalmazza:

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

fakt függvény -> szám faktoriálisát számolja ki, és visszaad erre az értékre mutató mutatót. `static` -> lehetővé teszi hogy a lokális változók benne maradjanak a memóriában, attól függetlenül, hogy a program már túlépett a függvényen.

```
int* sum(int egyik, int masik){
    static int sum = egyik + masik;
    return &sum;
}

int szorzat(int egyik, int masik){
    return egyik*masik;
}

int osztas(int egyik, int masik){
        return egyik/masik;
}
```

A statikus változó miatt csinálni kell egy egészre mutató mutatót visszadó függvényre mutató mutatót. Ez a pointer mutathatna a fakt-ra, csak az a gond, hogy akkor a sum értéke meg változik, ezért definiáljuk a sum függvényt, ami szintén egy egészre mutató mutatót ad vissza, szorzat és osztas függvények pedig csak segéd függvények.

```
int (*pfgv (int a)) (int,int){
    if (a){
        return &szorzat;
    }
    else
        return &osztas;
}
```

pfgv -> egy olyan függvény, ami függvényre mutató pointert ad vissza, egy olyan függvényre mutató pointert, ami két egészt kér paramétereül, maga a pfgv egy egészt kér, ami egy pointert ad vissza a szorzat vagy az osztas függvényre.

```
int main()
{
    int a = 10;
    int b = 5;
    int* pa = &a;
```

```
    int& ra = a;
    int tomb[a];
    int(& rtomb)[a] = tomb;
    int* ptomb[2];
    ptomb[0] = &a;
    ptomb[1] = &b;
    int* fakt_a = fakt(a);
    int*(*psum)(int,int) = &sum;
    int (*(*p_pfgv) (int valami))(int, int) = &pfgv;
    std::cout<<"a és b szorzata "<<(pfgv(1))(a,b)<<std::endl;
    std::cout<<"a és b hányadosa "<<(pfgv(0))(a,b)<<std::endl;
    std::cout<<"a és b hányadosa "<<(p_pfgv(0))(a,b)<<std::endl;
    std::cout<<"a értéke "<<a<<'\t'<<"a! értéke "<<*fakt_a<<std::endl;
    std::cout<<"a és b összege "<<*psum(a,b)<<std::endl;
}
```

A `a` és `b` -> egész

`int* pa = &a` -> egészre mutató mutatót vezet be

`int a` -> egy egész vezet be a programba.

`int &r = a` -> egy egésznek a referenciáját vezeti be, C++-ban

`int c[5]` -> egy egészekből álló tömbö dekralál `int (&tr)[5] = c` -> egészek tömbjének referenciáját vezeti be

`int *d[5]` -> egészre mutató mutatók tömbje

`int *h ()` -> egészre mutató mutatót visszadó függvény

`int *(*l) ()` -> egészre mutató mutatót visszaadó függvényre mutató mutató.

`int (*v (int c)) (int a, int b)` -> egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

`int (*(*z) (int)) (int, int)` -> két egészet kapó, egy egészet visszaadó függvényre mutató mutató

4. fejezet

Helló, Caesar!

4.1. int ** háromszögmátrix

Itt a C nyelv egyik nagy előnye mutatkozik meg, ami a dinamikus memóriakezelés. Ebben a példában létrehozunk egy duble ** háromszögmátrixot, ami lényegében egy két dimenziós tömb (C-ben a [] jelek használata a tömbök kezelésére csupán egy fordító adta kényelem, hogy ne kelljen mindig mutató-, és cím aritmetikával foglalkozunk).A különbség itt az lesz, hogy nem mondjuk meg előre, hogy hány elemű tömbjeink lesznek, hanem azokat majd dinamikusan foglaljuk le.

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

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

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

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

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

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

```
    }

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

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

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

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

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

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

    free (tm);

    return 0;
}
```

Először meg adjuk hogy hányas tömböt szeretnénk létrehozni, majd ezt eltároljuk egy váltözóba. Ezután le foglaljuk magát a hm Ezután lefoglaljuk magát a hm háromszögmátrixot, és a megértés érdekében kiíratjuk a memória címét is (példa lentebb). Ezután lefoglalunk `db` darab (esetünkben 5) tömböt (ezek lesznek a sorok), amelyek a következő lépésben hasonlóan tartalmazni fognak 5 db tömböt (ezek pedig az oszlopok).

Egy alsó háromszögmátrix egy olyan mátrix, melyben a főátló felett csupán nullák vannak. A programunk egy kvadratikus háromszögmátrixot készít el, olyan módon, hogy minden elemét úgy állítjuk be, hogy `(i+1)/(j+1)`, ahol az i a sor száma, a j pedig az oszlop száma. A +1 azért kell, hogy ne osszunk nullával véletlenül se.

```
$ gcc doubleharomszog.c  && ./a.out
Mutato cime: 0x7fffbc435de0
Sorok tombjenek cime: 0x5567bca69670
Elso sor cime: 0x5567bca696a0
```

```
|1.00  0.00  0.00  0.00  0.00  |
|2.00  1.00  0.00  0.00  0.00  |
|3.00  1.00  1.00  0.00  0.00  |
|4.00  2.00  1.00  1.00  0.00  |
|5.00  2.00  1.00  1.00  1.00  |
```

A feladat tanulsága, hogy a [] jelek valóban csak a fordító adta kényelmi funkció. C-ben azt a kifejezést, hogy `hm[1][1]` legalább háromféleképpen fejezhetjük ki, melyek a következők: `(*hm[1]+1)`, vagy `(*(*hm + 1)+1)`.

4.2. C EXOR titkosító

Az XOR titkosító programunk egyszerűen működik, az argumentumból kiolvassuk a kulcsot aztán a `while` ciklusban olvassuk az érkező bementet a sztenderd inputon. Majd megnézzük a bemenetünk méretét, és ennek a teljes tartalmát "titkosítjuk", azaz elvégezzük rajta a bitenkénti XOR utasítást `buffer[i] ^= key[key_index];`). Ezután vesszük a buffert és kinyomjuk az úgynevezett sztenderd outputon. Ez azért jó, mert a felhasználó egyszerűen inspektálhatja, vagy fájlba írhatja a kimenetet, legalábbis UNIX-jellegű operációs rendszerek alatt.

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

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

  int kulcs_index = 0;
  int olvasott_bajtok = 0;

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

  while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
    {

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

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

```
  }

      write (1, buffer, olvasott_bajtok);

    }
}
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

A Java egy objektum orientált nyelv, ami a C-vel nagyon hasonló szintaktikailag, de nincs benne semmi féle memóriakezelés.

```
import java.util.*;

class XorEncode {
  public static void main(String[] args) {
    String kulcs = "";

    if(args.length > 0) {
      kulcs = args[0];
    } else {
      System.out.println("Kulcs nelkul nem titkositok!");
      System.out.println("Hasznalat: java XorEncode.java [kulcs]");
      System.exit(-1);
    }

    Scanner sc = new Scanner(System.in);
    String str = "";

    while(sc.hasNext()) {
      str = sc.next();
      System.out.println(xor(kulcs, str));
    }
  }

  public static String xor(String kulcs, String s) {
    StringBuilder sb = new StringBuilder();

    for(int i = 0; i < s.length(); i++) {
      sb.append((char)(s.charAt(i) ^ kulcs.charAt(i % kulcs.length())));
    }

    return sb.toString();
```

```
    }
}
```

Hasonlóan járunk el mint a C változatban.

4.4. C EXOR törő

Megoldás forrása: https://github.com/Oszuszki/_owntextbookproject_/blob/master/_files_/_caesar_/exortoro.c

Ez a program csak olyan állományok feltörésére alkalmas, amelyeknek a kulcsa csak számokból áll, és 8 karakter hosszú. Ezt a kódban lehet módosítani.

A programunk egy bruteforce algoritmust használ, hogy visszafejtse a titkosított szöveget. A bemenetet olvasva a program meghatározza a szavak hosszát, és amennyiben ez megfelel egy bizonyos értékhatárnak, megnézi, hogy előfordulnak-e benne a leggyakoribb magyar szavak (*a, meg, vagy, van, volt, már, stb...*), és ha sikerül találni ilyeneket, akkor kiírja a kimenetre.

Ebben a példában ismét az OpenMP-t használjuk párhuzamosításra, hogy minél gyorsabban végezzünk a töréssel.

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 neuralis

A neuron az elektromos jelek összegyűjtéséért, feldolgozásáért és szétterjesztéséért felelős agysejt. Az agy információfeldolgozó kapacitása elsősorban neuronok hálózatából alakult ki.

Itt az R programban egy neutrális hálót építünk fel, amely úgy működik, hogy meg adjuk milyen bemenetre, milyen kimenetet várunk, amit a program meg próbál mesterségesen utánozni

1 és a2 értékeket tartalmaz, az OR pedig a logikai VAGY műveletet jelöli. A program az általunk meghatározott szabályok alapján elkezd tanulni. A `compute` parancs segítségével tudjuk leellenőrizni, hogy a megfelelő eredményeket kaptuk-e vagy sem. A logikai ÉS művelet (AND) betanítása is hasonló módon történik. Az EXOR műveletnél azonban csak többrétegű neuronokkal lehetséges a tanítás (hidden = 2).

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

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

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

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

plot(nn.or)

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

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

orand.data <- data.frame(a1, a2, OR, AND)

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

plot(nn.orand)

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

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

exor.data <- data.frame(a1, a2, EXOR)

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

plot(nn.exor)

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

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

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ↩
   output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

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

A Turing-fejezetnél már haaználtuk az R nyelevet (sudo apt install r-base) A neurális hálóhoz azonban szükség van egy új package, a neuralnet telepítésére is.

```
$ sudo -i
root@oszuszki-richard:~# R

R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.


> install.packages("neuralnet")
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
trying URL 'https://cloud.r-project.org/src/contrib/neuralnet_1.44.2.tar.gz ↩
   '
Content type 'application/x-gzip' length 27211 bytes (26 KB)
==================================================
downloaded 26 KB

* installing *source* package ‘neuralnet’ ...
** package ‘neuralnet’ successfully unpacked and MD5 sums checked
** R
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (neuralnet)

The downloaded source packages are in
  ‘/tmp/RtmpzKLwdC/downloaded_packages’
> q()
Save workspace image? [y/n/c]: y



$ Rscript neuralis.r
$neurons
$neurons[[1]]
       a1 a2
```

```
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1


$net.result
            [,1]
[1,] 0.001199652
[2,] 0.999127840
[3,] 0.999665104
[4,] 1.000000000

dev.new(): using pdf(file="Rplots1.pdf")
$neurons
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1


$net.result
             [,1]         [,2]
[1,] 1.352313e-05 2.324786e-09
[2,] 9.999951e-01 1.265366e-03
[3,] 9.999909e-01 1.201426e-03
[4,] 1.000000e+00 9.984769e-01

dev.new(): using pdf(file="Rplots2.pdf")
$neurons
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1


$net.result
          [,1]
[1,] 0.5000047
[2,] 0.4999999
[3,] 0.5000015
[4,] 0.4999967

dev.new(): using pdf(file="Rplots3.pdf")
$neurons
$neurons[[1]]
```

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

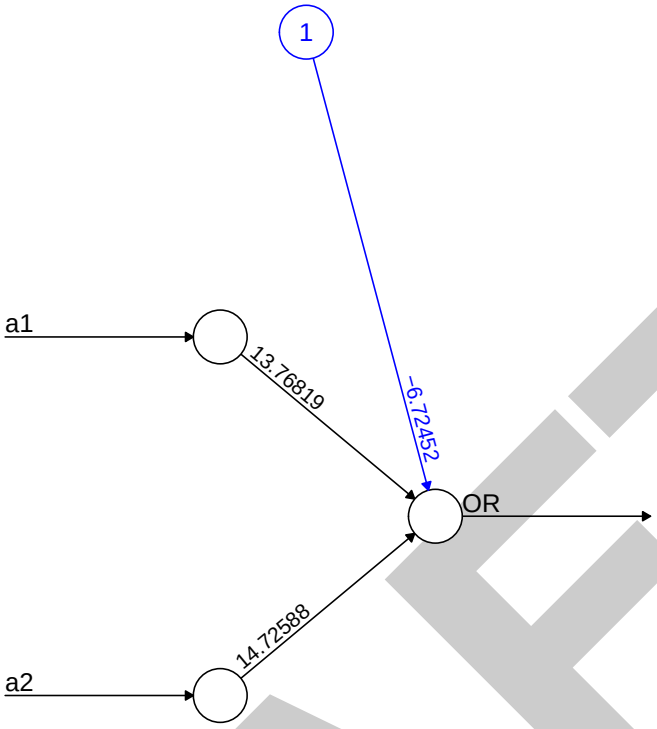
$neurons[[2]]
     [,1]       [,2]      [,3]       [,4]       [,5]       [,6]      [,7]
[1,]    1 0.04939826 0.9939279 0.02393772 0.01622474 0.08610279 0.9640993
[2,]    1 0.34475792 0.9925660 0.63066797 0.26255545 0.32004680 0.8650567
[3,]    1 0.30720762 0.9942785 0.11943347 0.15703431 0.44594585 0.5846982
[4,]    1 0.81784444 0.9929947 0.90424853 0.80085868 0.80084033 0.2515419

$neurons[[3]]
     [,1]       [,2]         [,3]         [,4]       [,5]
[1,]    1 0.02569708 9.687810e-01 0.0004236877 0.99693862
[2,]    1 0.98115879 5.192263e-03 0.0221381966 0.93650424
[3,]    1 0.92726506 2.611690e-02 0.0266950688 0.92936080
[4,]    1 0.99999993 6.730118e-09 0.9880119879 0.02325084

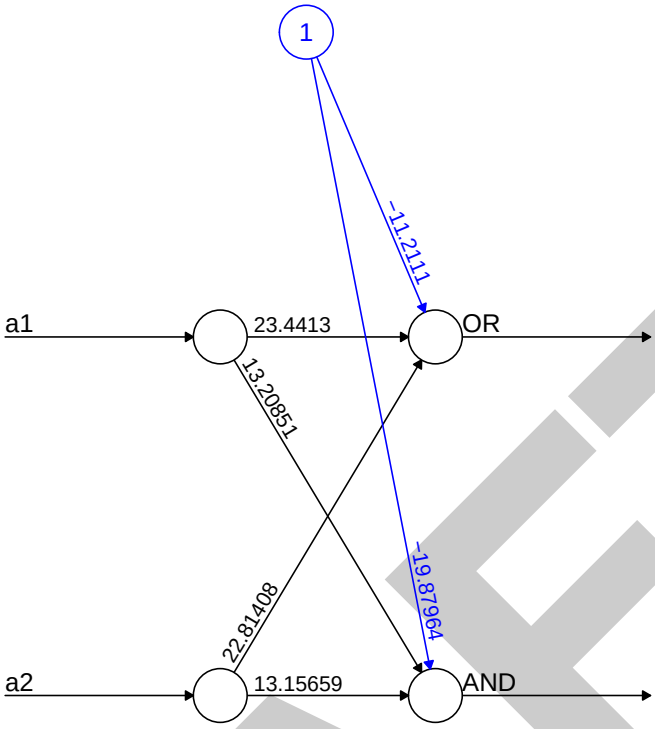
$neurons[[4]]
     [,1]       [,2]       [,3]       [,4]       [,5]       [,6]       [,7]
[1,]    1 0.86496705 0.01058845 0.92011779 0.87398264 0.11143327 0.81722131
[2,]    1 0.02156429 0.99951675 0.02011744 0.75885706 0.97758849 0.01096550
[3,]    1 0.02643377 0.99923565 0.02485400 0.76028586 0.97319306 0.01421735
[4,]    1 0.97727737 0.99999830 0.39296493 0.09925582 0.02637285 0.97012081


$net.result
             [,1]
[1,] 0.0002045972
[2,] 0.9994468628
[3,] 0.9994131333
[4,] 0.0007729238
```

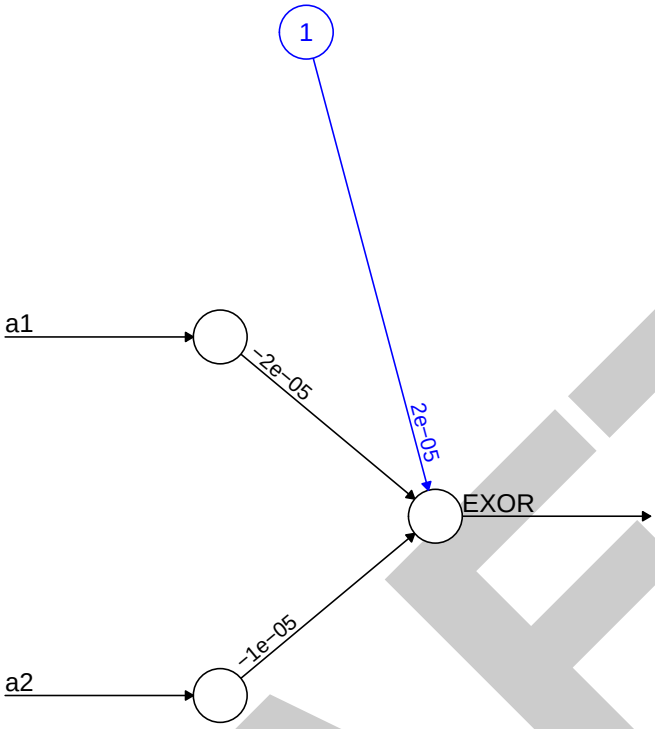
Ezután megkapjuk a szimuláció eredményeit szemléltető ábrákat:



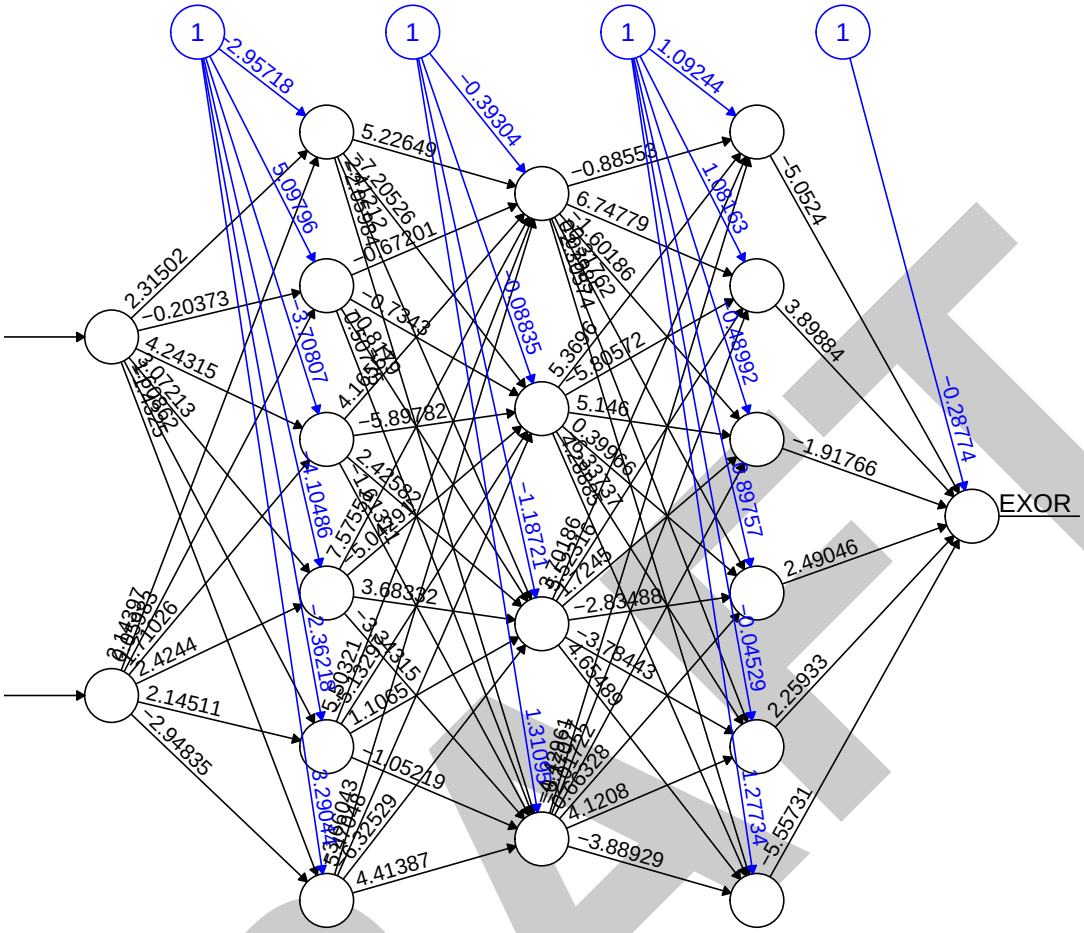
Error: 1e−06 Steps: 152



Error: 3e−06 Steps: 221



Error: 0.5 Steps: 84



4.6. Hiba-visszaterjesztéses perceptron

C++

Ebben a feladatban a mandelbrot halmaz által generált kép rgb kódjait áttesszük a neurális háló inputjába, egy három rétegű hálót csinálunk és végül különböző számítások alapján kapunk a 3. rétegben egy számot.

```
#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;



}
```

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

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

#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35

void GeneratePNG( int tomb[N][M])
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y ↩
               ]);
        }
    }
    image.write("kimenet.png");
}

struct Komplex
{
    double re, im;
};

int main()
{
    int tomb[N][M];
```

```
    int i, j, k;

    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;

    struct Komplex C, Z, Zuj;

    int iteracio;

    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            C.re = MINX + j * dx;
            C.im = MAXY - i * dy;

            Z.re = 0;
            Z.im = 0;
            iteracio = 0;

            while(Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < 255)
            {
                Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
                Zuj.im = 2 * Z.re * Z.im + C.im;
                Z.re = Zuj.re;
                Z.im = Zuj.im;
            }

            tomb[i][j] = 256 - iteracio;
        }
    }

    GeneratePNG(tomb);

    return 0;
}
```

A Mandelbrot halmaz illeszkedik a $f_c(z) = z^2 + c$ függvény képére, s nullától iterálva nem divergál azaz korlátos $f_c(0), f_c(f_c(0)), \ldots$ abszolútértékben.

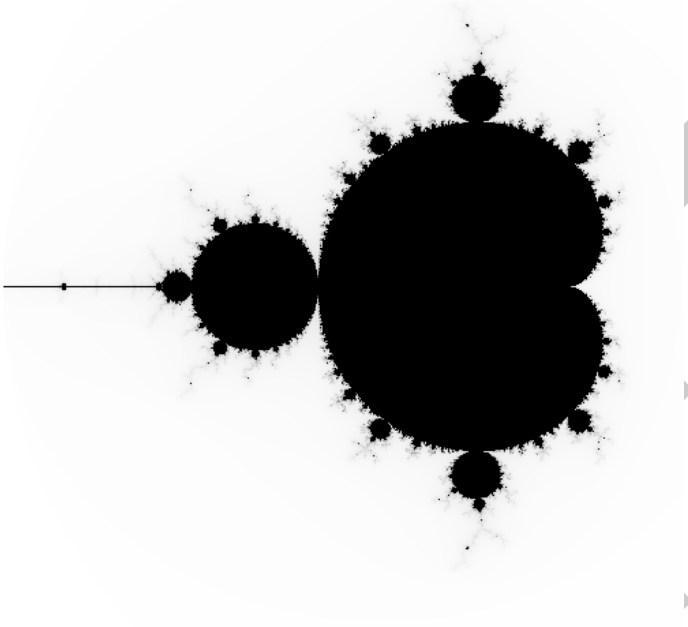
```
all: mandelbrot clean

mandelbrot.o: mandelbrot.cpp
  @g++ -c mandelbrot.cpp `libpng-config --cflags`

mandelbrot: mandelbrot.o
  @g++ -o mandelbrot mandelbrot.o `libpng-config --ldflags`

clean:
  @rm -rf *.o
```

```
  @./mandelbrot
  @rm -rf mandelbrot
```



5.1. ábra. A Mandelbrot halmaz a komplex síkon

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

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

const int N = 500;
const int M = 500;
const double MAXX = 0.7;
const double MINX = -2.0;
const double MAXY = 1.35;
const double MINY = -1.35;

void GeneratePNG(const int tomb[N][M])
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y ↩
               ]);
```

```
        }
    }
    image.write("kimenet.png");
}

int main()
{
    int tomb[N][M];

    double dx = ((MAXX - MINX) / N);
    double dy = ((MAXY - MINY) / M);

    std::complex<double> C, Z, Zuj;

    int iteracio;

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
      C = {MINX + j * dx , MAXY - i * dy};

            Z = 0;
            iteracio = 0;

            while(abs(Z) < 2 && iteracio++ < 255)
            {
                Zuj = Z*Z+C;
                Z = Zuj;
            }

            tomb[i][j] = 256 - iteracio;
        }
    }

    GeneratePNG(tomb);

    return 0;
}
```

Ugyan az mint az előző, csak itt `std::complex` osztályt használjuk, így nem kell saját struktúrát írni a komplex számok tárolására.

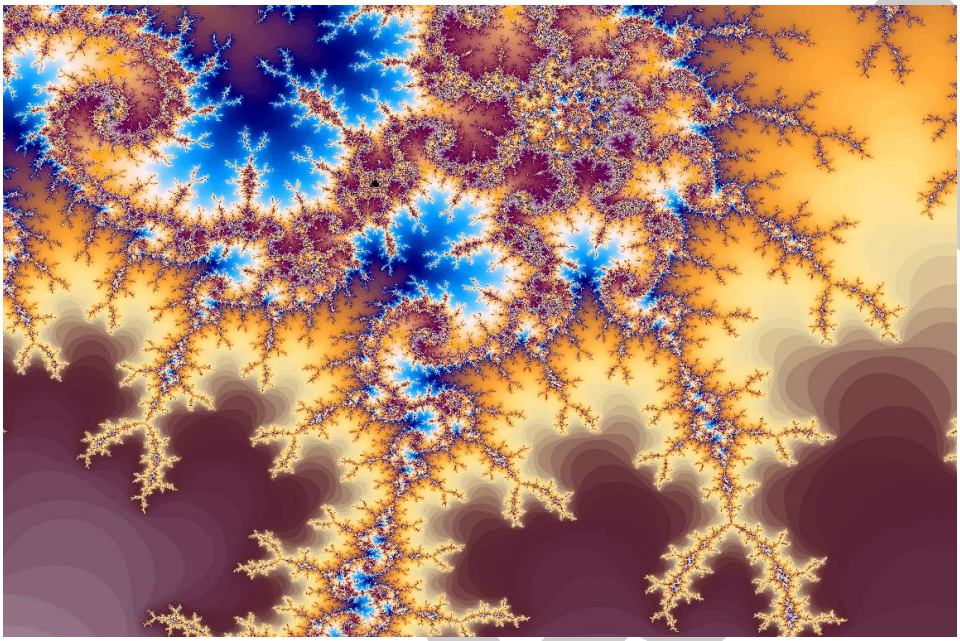
A `std::complex` köszönhetően sokkal kényelmesebb a változók definiálása Például: `C = {MINX + j * dx,` -> szám valós része, `MAXY - i * dy};` -> a szám imaginárius része.

5.3. Biomorfok

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

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

Ahogyan a Mandelbrot halmaznál, itt is egy komplex számsíkon ábrázolható függvényt nézünk meg.



Itt is Makefile-t használióunk, így a `make` parancs kiadásával lefuttatható a programunk.

```
        all: biomorf clean

biomorf.o: biomorf.cc
  @g++ -c biomorf.cc `libpng-config --cflags` -O3

biomorf: biomorf.o
  @g++ -o biomorf biomorf.o `libpng-config --ldflags` -O3

clean:
  @rm -rf *.o
  @./biomorf bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
  @rm -rf biomorf
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

A feladat megvalósításához a Qt Creator nevű szoftvert fogjuk használni, mely egy több platformon átívelő keretrendszer elsősorban grafikus alkalmazások készítésére. A Qt nagyon népszerű választás a nagyobb projektek körében is, például a Qt az alapja a VLC-nek, és még sok más **szabad szoftvernek**.

Ebben a programban alapul vesszük a már meglévő Mandelbrot-halmaz számító C++ programunkat, és egy más kontextusba ültetjük be. Természetesen ez még nem lenne elég, hanem el kell készítenünk hozzá Qt Creatorban a felületet, valamint némi extra kódot. Ez a kód a repóban megtalálható.

A program hasznáalta során az egérrel jelölünk ki egy adott területet, melyet az újra renderel és betölt. A programról használat közben beillesztek néhány képernyőképet.

5.5. Mandelbrot nagyító és utazó C++ nyelven

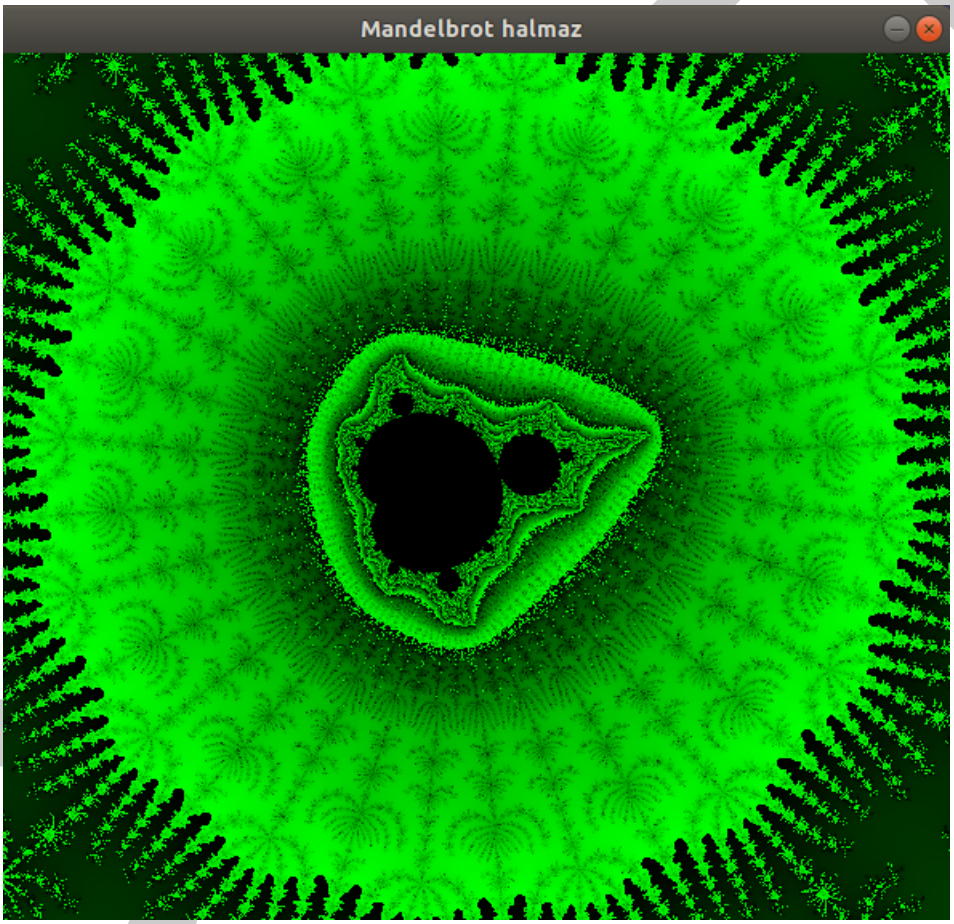
Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteréció bejárta z_n komplex számokat!

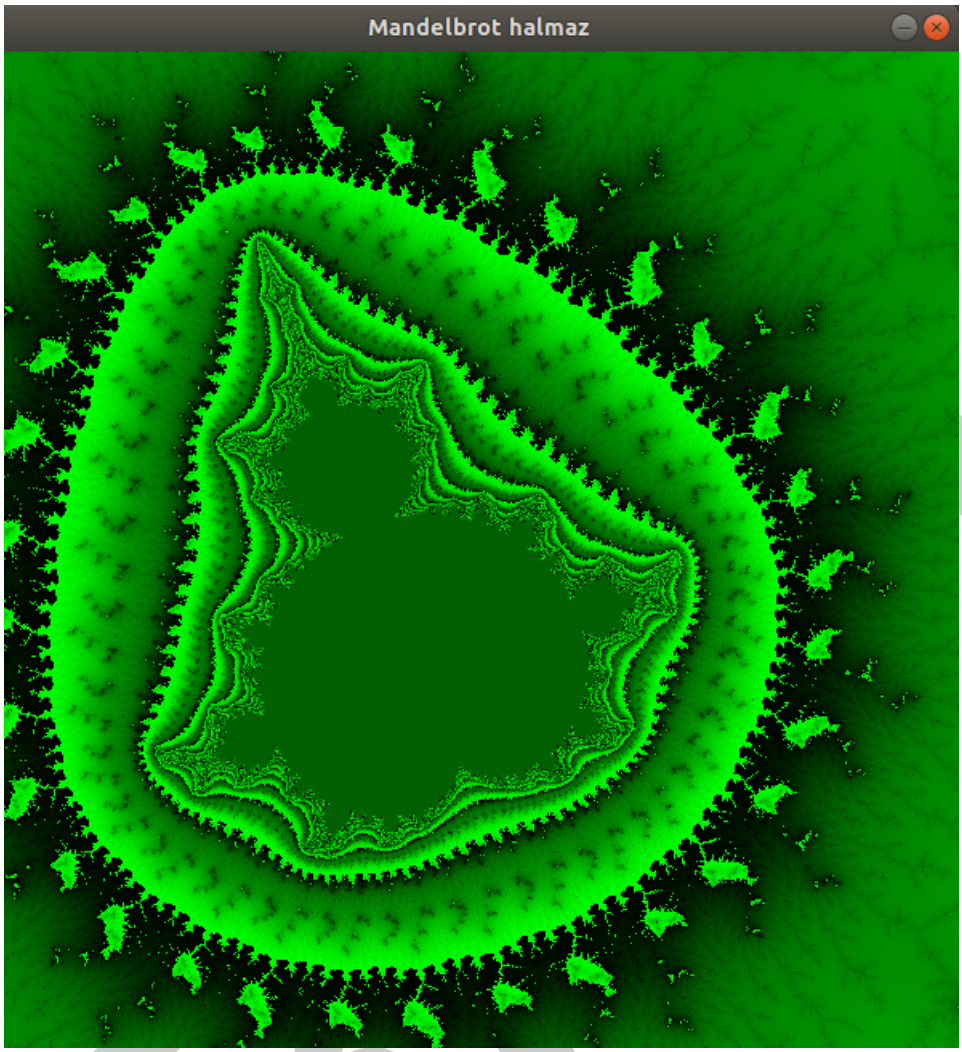
Mandelbot halmazt készítünk, csak itt bele tudunk nagyítani oda ahová szeretnénk.

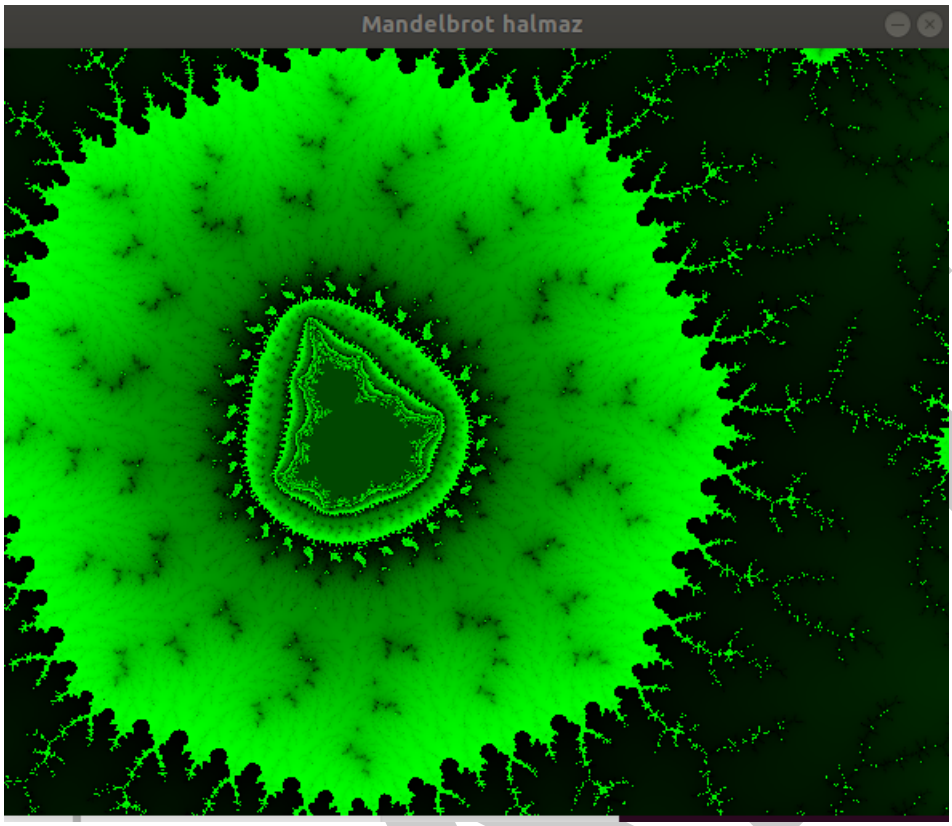
Telepítsük le a `libqt` könytárat, majd `qmake -project` parancsot írjuk be a terminálba, amely a mappa nevéhez alkalmaszkodva létre hoz egy .pro fájlt, ezek után elkészítjük a Makefilet.

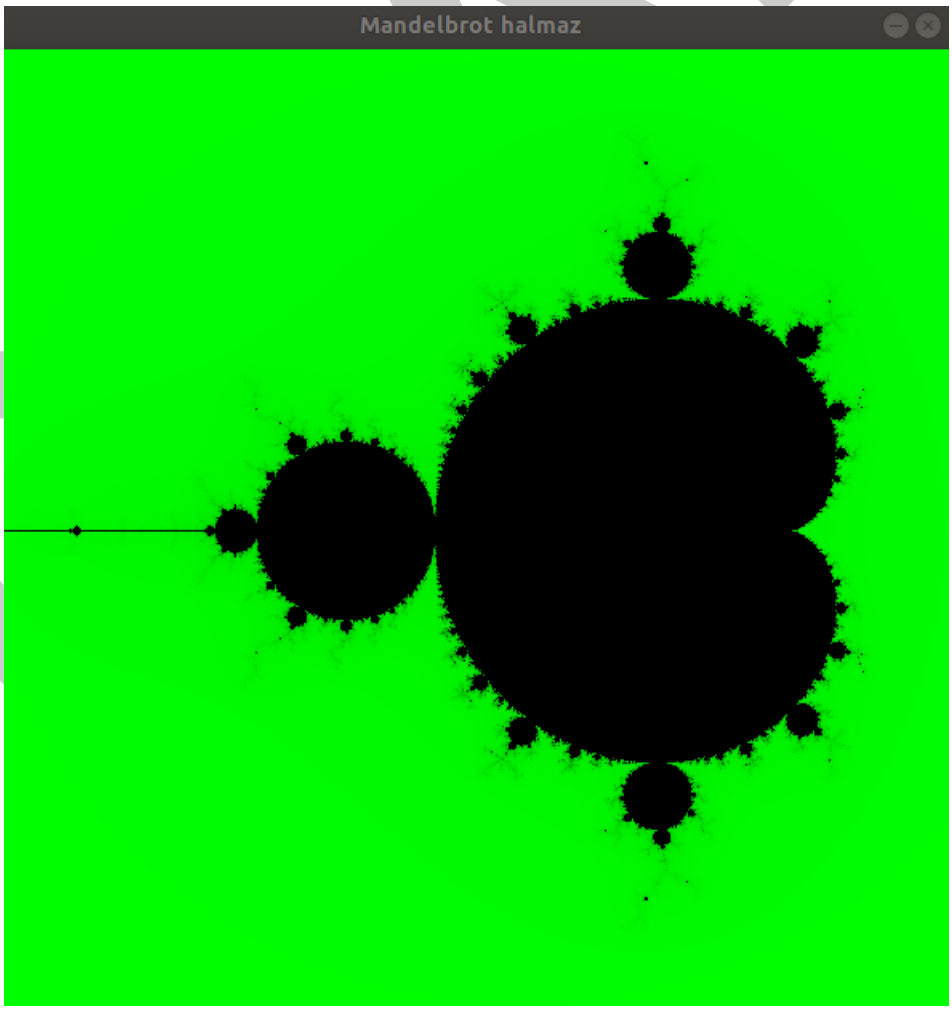
```
$ sudo apt-get install libqt5-dev
$ ls
frakablak.cpp  frakablak.h  frakszal.cpp  frakszal.h  main.cpp
$ qmake -project
$ ls
frakablak.cpp  frakablak.h  frakszal.cpp  frakszal.h  main.cpp  nagyito_c ↩
   ++.pro
$ qmake nagyito_c++.pro
$ ls
frakablak.cpp  frakszal.cpp  main.cpp  nagyito_c++.pro
frakablak.h    frakszal.h    Makefile
$ make
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB - ↩
   DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/ ↩
   usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. - ↩
   I. -o frakablak.o frakablak.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB - ↩
   DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/ ↩
   usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. - ↩
   I. -o frakszal.o frakszal.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB - ↩
   DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/ ↩
   usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. - ↩
   I. -o main.o main.cpp
/usr/lib/x86_64-linux-gnu/qt4/bin/moc -DQT_NO_DEBUG -DQT_GUI_LIB - ↩
   DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/ ↩
   usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. - ↩
   I. frakablak.h -o moc_frakablak.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB - ↩
   DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/ ↩
   usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. - ↩
   I. -o moc_frakablak.o moc_frakablak.cpp
```

```
/usr/lib/x86_64-linux-gnu/qt4/bin/moc -DQT_NO_DEBUG -DQT_GUI_LIB - ↩
   DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/ ↩
   usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. - ↩
   I. frakszal.h -o moc_frakszal.cpp
g++ -c -m64 -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB - ↩
   DQT_CORE_LIB -DQT_SHARED -I/usr/share/qt4/mkspecs/linux-g++-64 -I. -I/ ↩
   usr/include/qt4/QtCore -I/usr/include/qt4/QtGui -I/usr/include/qt4 -I. - ↩
   I. -o moc_frakszal.o moc_frakszal.cpp
g++ -m64 -Wl,-O1 -o nagyito_c++ frakablak.o frakszal.o main.o moc_frakablak ↩
   .o moc_frakszal.o    -L/usr/lib/x86_64-linux-gnu -lQtGui -lQtCore - ↩
   lpthread
$ ./nagyito_c++
```






Mandelbrot halmaz

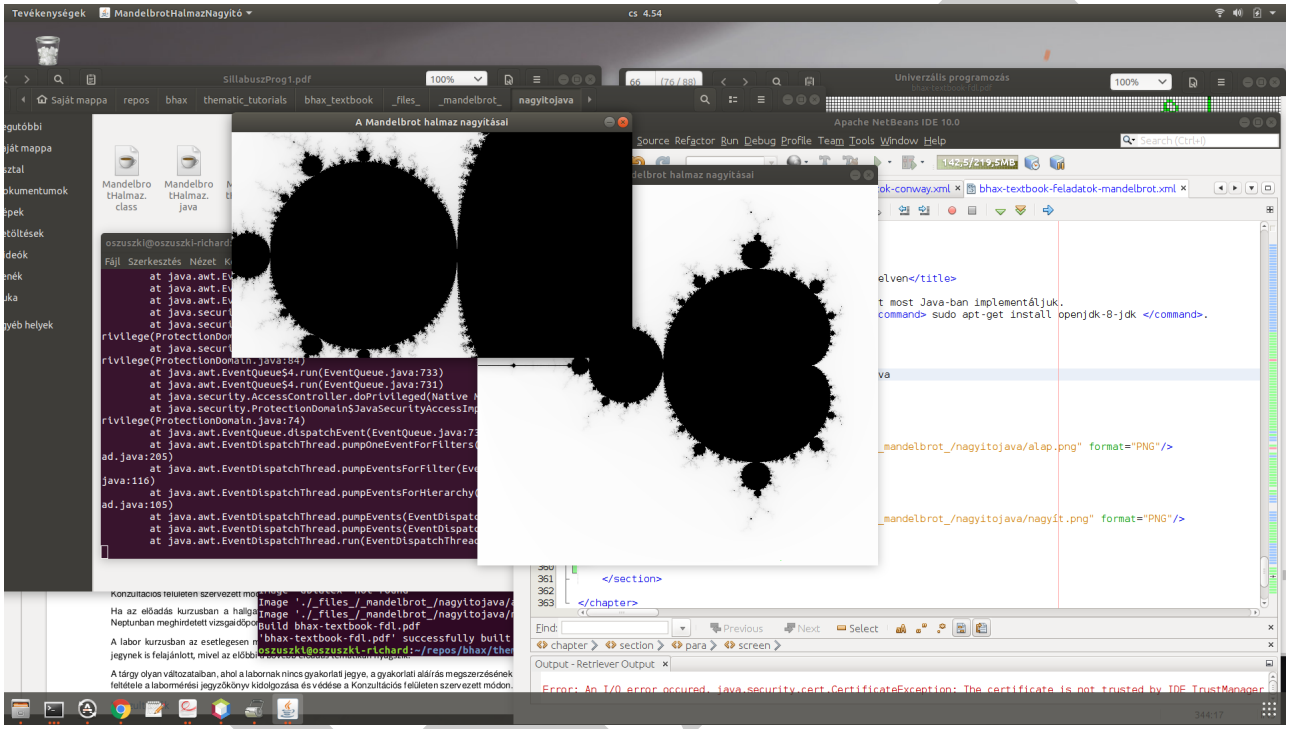

Mandelbrot halmaz

5.6. Mandelbrot nagyító és utazó Java nyelven

Az előző feladatban látott C++ nagyítót most Java-ban implementáljuk. A fordításhoz,futtatáshoz szükséges: **sudo apt-get install openjdk-8-jdk** .

Fordítás, futtatás:

```
$ javac MandelbrotHalmazNagyító.java
$ java MandelbrotHalmazNagyító
```



6. fejezet

Helló, Welch!

6.1. Első osztályom

```
public class PolarGenerator {
    boolean nincsTarolt = true;
    double tarolt;
    public PolarGenerator() {

        nincsTarolt = true;

    }
    public double kovetkezo() {
        if(nincsTarolt) {
            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();

                v1 = 2*u1 - 1;
                v2 = 2*u2 - 1;

                w = v1*v1 + v2*v2;

            } while(w > 1);

            double r = Math.sqrt((-2*Math.log(w))/w);

            tarolt = r*v2;
            nincsTarolt = !nincsTarolt;

            return r*v1;

        } else {
            nincsTarolt = !nincsTarolt;
            return tarolt;
```

```
        }
    }

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

```
$ oszuszki@oszuszki-richard:~$ java PolarGenerator
-0.5854006412057287
0.3294220378282633
-0.5618822511946688
-0.5162512889534413
-0.11869578322547197
1.210163412516636
0.16919466740119235
-0.16397819117423162
-0.6952737986265023
0.20957744539899742
oszuszki@oszuszki-richard:~$ java PolarGenerator
1.058675233600098
0.6988814229959112
-0.26362470577717434
-0.42547452177890543
-0.7557796238151275
0.35829458368626993
0.10611328155606277
1.6278775533208691
-1.6280313880302872
1.1994118333403905
```

A program 10 db random generált számot ad vissza, a 00 előnye itt mutatkozik meg, a matematikai háttér számunkra lényegtelen, mégis tökéletesen működik.

6.2. LZW

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

Megoldás forrása:

Z3A7.CPP

A LZWBinFa osztály felépíti a bemeneti fájl bináris fáját. Így használjuk: `./binfa [bemenő fájl] -o [kimenő fájl]`.

6.3. Fabejárás

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

Preorder bejárás = gyökér -> baloldali részfa -> jobboldali részfa

Postorder = gyökér -> jobboldali részfa -> baloldali részfa

Inorder = bal oldali részfa -> gyökér -> jobboldali részfa

Program futtatása: `./binfa befile -o kifile [o/r]` ahol az o postorder, az r pedig preorder bejárást jelent.

6.4. Tag a gyökér

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

A már megírt programban, a gyökér, már alápjáraton kompozícióban van a fával.

Z3A7.CPP

6.5. Mutató a gyökér

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

Megoldás forrása:

Z3A7.CPP

Itt a bineáris fánk, a gyökére mutat.

`gyoker`-t pointerre írjuk át,

```
                Csomopont *gyoker;
```

Futtatáskor rengeteg hibát ad vissza, de no problem, megyünk tovább, a konstruktort is változtassuk meg.

```
LZWBinFa ()
    {
        gyoker = new Csomopont ('/');
        fa = gyoker;
    }
```

A gyoker és a fa mutatónkat új tárterületre állítjuk, a foglalt területet fel szabadítjuk.

```
~LZWBinFa ()
    {
        szabadit (gyoker->egyesGyermek ());
        szabadit (gyoker->nullasGyermek ());
        delete(gyoker);
    }
```

A gyökér mutató lett, tehát a gyökér álltatl mutatott csomópont egyes és nullás gyermekére kell meghívnunk a szabadit függvényt. A gyokér által mutatott területet a `delete()`-el szabadítjuk fel. Töröljük mindenhol a & címképző operátort, azaz mostmár nem a gyökér memóriacímét kell átadni, hanem a gyökeret.

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

Myrmecologist program Qt C++ környezetben íródott. `ant.h` header fájlban vannak a koordináták. Ezek publikus elemek, ugyhogy nem csak a classon elül lestnek érvényesek. Deklaráltuk az ants vektort is, ebben a hangyákat tároljuk.

```
$ more ant.h
#ifndef ANT_H
#define ANT_H

class Ant
{

public:
    int x;
    int y;
    int dir;

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

        dir = qrand() % 8;

    }

};

typedef std::vector<Ant> Ants;

#endif
```

A szimuláció ablakának paraméteres adatait a public részben határoztuk meg, míg a hangyákat megjelenítő cellákét a privátban, a `antwin.h` header file AntWin osztályában.

`closeEvent` -> események kikapcsolását-,

`keyPressEvent` -> beadott gomblenyomásokat kezeli

```
$ more antwin.h
#ifndef ANTWIN_H
#define ANTWIN_H

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

class AntWin : public QMainWindow
{
    Q_OBJECT

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

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {

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

    void keyPressEvent ( QKeyEvent *event )
    {

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

    }

    virtual ~AntWin();
```

```
    void paintEvent(QPaintEvent*);

private:

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

public slots :
    void step ( const int &);

};

#endif
```

Az `antthread.h` header file AntThread osztályában láthatjuk többek között a párolgás mértékét (evaporation) és a feromonok számát (nbrPheromone) tároló változókat. Megfigyelhetjük még a hangyák cellán belüli maximális előfordulásának számát tároló változót is (cellAntMax). Private függvények közé soroljuk pl. a `newDir`-t, amely új irányt határoz meg, illetve a `moveAnts`-et, ami pedig a hangyák mozgatásáért felelős. AntThread osztályában találhatjuk meg a párolgás mértékét, és a feromonok számát (a hangyák magukután feromonnyomot hagynak, igazából ebben a fejezetbe ezt figyelhetjük meg, egy szimuláció álltal).

`newDir` -> új irányt határoz meg.

`moveAnts` -> hangyák mozgásáért felelős.

```
$ more antthread.h
#ifndef ANTTHREAD_H
#define ANTTHREAD_H

#include <QThread>
#include "ant.h"

class AntThread : public QThread
{
    Q_OBJECT

public:
    AntThread(Ants * ants, int ***grids, int width, int height,
              int delay, int numAnts, int pheromone, int nbrPheromone,
              int evaporation, int min, int max, int cellAntMax);

    ~AntThread();
```

```
    void run();
    void finish()
    {
        running = false;
    }

    void pause()
    {
        paused = !paused;
    }

    bool isRunnung()
    {
        return running;
    }

private:
    bool running {true};
    bool paused {false};
    Ants* ants;
    int** numAntsinCells;
    int min, max;
    int cellAntMax;
    int pheromone;
    int evaporation;
    int nbrPheromone;
    int ***grids;
    int width;
    int height;
    int gridIdx;
    int delay;

    void timeDevel();

    int newDir(int sor, int oszlop, int vsor, int voszlop);
    void detDirs(int irany, int& ifrom, int& ito, int& jfrom, int& jto );
    int moveAnts(int **grid, int row, int col, int& retrow, int& retcol,  ↩
       int);
    double sumNbhs(int **grid, int row, int col, int);
    void setPheromone(int **grid, int row, int col);

signals:
    void step ( const int &);

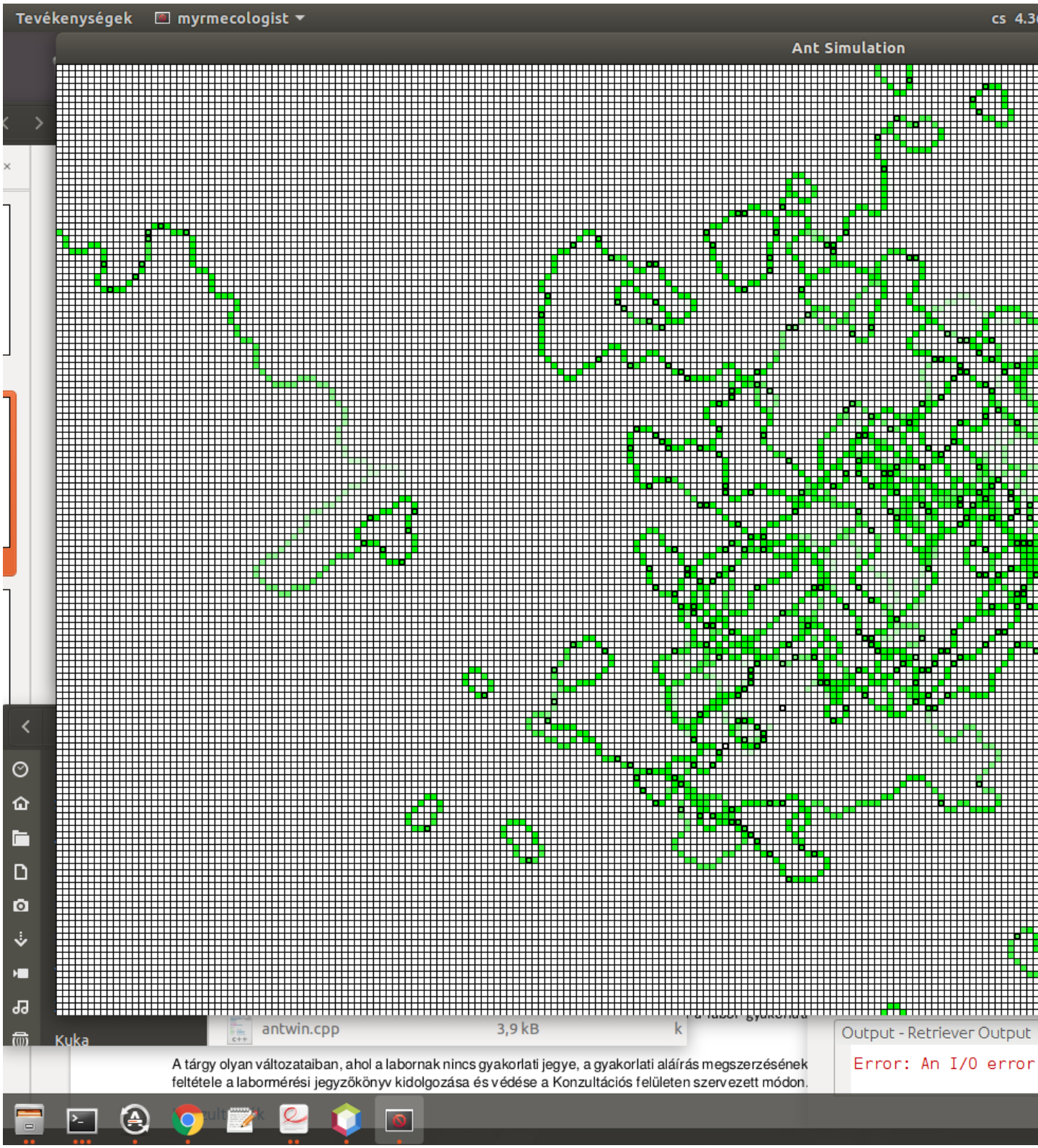
};

#endif
```

Fordítás, és futtatás:

```
$ ls
ant.h  antthread.cpp  antthread.h  antwin.cpp  antwin.h  main.cpp   ↩
   myrmecologist.pro
$ qmake myrmecologist.pro
Info: creating stash file /home/orszaghlev/Asztal/hangya/.qmake.stash
$ ls
ant.h  antthread.cpp  antthread.h  antwin.cpp  antwin.h  main.cpp  Makefile ↩
     myrmecologist.pro
$ make
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_NO_DEBUG -DQT_WIDGETS_LIB ↩
    -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/x86_64-linux- ↩
   gnu/qt5 -isystem /usr/include/x86_64-linux-gnu/qt5/QtWidgets -isystem / ↩
   usr/include/x86_64-linux-gnu/qt5/QtGui -isystem /usr/include/x86_64- ↩
   linux-gnu/qt5/QtCore -I. -isystem /usr/include/libdrm -I/usr/lib/x86_64- ↩
   linux-gnu/qt5/mkspecs/linux-g++ -o main.o main.cpp
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_NO_DEBUG -DQT_WIDGETS_LIB ↩
    -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/x86_64-linux- ↩
   gnu/qt5 -isystem /usr/include/x86_64-linux-gnu/qt5/QtWidgets -isystem / ↩
   usr/include/x86_64-linux-gnu/qt5/QtGui -isystem /usr/include/x86_64- ↩
   linux-gnu/qt5/QtCore -I. -isystem /usr/include/libdrm -I/usr/lib/x86_64- ↩
   linux-gnu/qt5/mkspecs/linux-g++ -o antwin.o antwin.cpp
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_NO_DEBUG -DQT_WIDGETS_LIB ↩
    -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/x86_64-linux- ↩
   gnu/qt5 -isystem /usr/include/x86_64-linux-gnu/qt5/QtWidgets -isystem / ↩
   usr/include/x86_64-linux-gnu/qt5/QtGui -isystem /usr/include/x86_64- ↩
   linux-gnu/qt5/QtCore -I. -isystem /usr/include/libdrm -I/usr/lib/x86_64- ↩
   linux-gnu/qt5/mkspecs/linux-g++ -o antthread.o antthread.cpp
g++ -pipe -O2 -Wall -W -dM -E -o moc_predefs.h /usr/lib/x86_64-linux-gnu/ ↩
   qt5/mkspecs/features/data/dummy.cpp
/usr/lib/qt5/bin/moc -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB - ↩
   DQT_CORE_LIB --include ./moc_predefs.h -I/usr/lib/x86_64-linux-gnu/qt5/ ↩
   mkspecs/linux-g++ -I/home/orszaghlev/Asztal/hangya -I/home/orszaghlev/ ↩
   Asztal/hangya -I/usr/include/x86_64-linux-gnu/qt5 -I/usr/include/x86_64- ↩
   linux-gnu/qt5/QtWidgets -I/usr/include/x86_64-linux-gnu/qt5/QtGui -I/usr ↩
   /include/x86_64-linux-gnu/qt5/QtCore -I/usr/include/c++/7 -I/usr/include ↩
   /x86_64-linux-gnu/c++/7 -I/usr/include/c++/7/backward -I/usr/lib/gcc/ ↩
   x86_64-linux-gnu/7/include -I/usr/local/include -I/usr/lib/gcc/x86_64- ↩
   linux-gnu/7/include-fixed -I/usr/include/x86_64-linux-gnu -I/usr/include ↩
    antwin.h -o moc_antwin.cpp
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_NO_DEBUG -DQT_WIDGETS_LIB ↩
    -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/x86_64-linux- ↩
   gnu/qt5 -isystem /usr/include/x86_64-linux-gnu/qt5/QtWidgets -isystem / ↩
   usr/include/x86_64-linux-gnu/qt5/QtGui -isystem /usr/include/x86_64- ↩
   linux-gnu/qt5/QtCore -I. -isystem /usr/include/libdrm -I/usr/lib/x86_64- ↩
   linux-gnu/qt5/mkspecs/linux-g++ -o moc_antwin.o moc_antwin.cpp
/usr/lib/qt5/bin/moc -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB - ↩
   DQT_CORE_LIB --include ./moc_predefs.h -I/usr/lib/x86_64-linux-gnu/qt5/ ↩
   mkspecs/linux-g++ -I/home/orszaghlev/Asztal/hangya -I/home/orszaghlev/ ↩
   Asztal/hangya -I/usr/include/x86_64-linux-gnu/qt5 -I/usr/include/x86_64- ↩
```

```
   linux-gnu/qt5/QtWidgets -I/usr/include/x86_64-linux-gnu/qt5/QtGui -I/usr ↩
   /include/x86_64-linux-gnu/qt5/QtCore -I/usr/include/c++/7 -I/usr/include ↩
   /x86_64-linux-gnu/c++/7 -I/usr/include/c++/7/backward -I/usr/lib/gcc/ ↩
   x86_64-linux-gnu/7/include -I/usr/local/include -I/usr/lib/gcc/x86_64- ↩
   linux-gnu/7/include-fixed -I/usr/include/x86_64-linux-gnu -I/usr/include ↩
    antthread.h -o moc_antthread.cpp
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -fPIC -DQT_NO_DEBUG -DQT_WIDGETS_LIB ↩
    -DQT_GUI_LIB -DQT_CORE_LIB -I. -I. -isystem /usr/include/x86_64-linux- ↩
   gnu/qt5 -isystem /usr/include/x86_64-linux-gnu/qt5/QtWidgets -isystem / ↩
   usr/include/x86_64-linux-gnu/qt5/QtGui -isystem /usr/include/x86_64- ↩
   linux-gnu/qt5/QtCore -I. -isystem /usr/include/libdrm -I/usr/lib/x86_64- ↩
   linux-gnu/qt5/mkspecs/linux-g++ -o moc_antthread.o moc_antthread.cpp
g++ -Wl,-O1 -o myrmecologist main.o antwin.o antthread.o moc_antwin.o  ↩
   moc_antthread.o   -lQt5Widgets -lQt5Gui -lQt5Core -lGL -lpthread
$ ls
ant.h          antthread.h  antwin.cpp  antwin.o  main.o    moc_antthread. ↩
   cpp  moc_antwin.cpp  moc_predefs.h  myrmecologist.pro
antthread.cpp  antthread.o  antwin.h    main.cpp  Makefile  moc_antthread.o ↩
       moc_antwin.o    myrmecologist
$ ./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s ↩
    3 -c 22
```



7.2. Qt C++ életjáték

Megoldás forrása: https://github.com/Oszuszki/_owntextbookproject_/tree/master/_files_/_conway_

A Qt-t már ismerjük, a Mandelbrotos fejezetnek köszönhetően, nos, most a Jonh H. Conway féle életjátékot üzemeljük be vele.

`./run` paranccsal futtatjuk.

A John Conway-féle életjáték szabályai:

- Azok a sejtek melyeknek két vagy három szomszédja van, életben maradnak
- Azok a sejtek melyeknek négy vagy öt szomszédja van, meghal
- Ha egy sejtnek kevesebb mint kettő szomszédja van, meghal
- Amenyiben egy üres cellát három élő sejt vesz körül, ott új selyt születik

7.3. BrainB Benchmark

Megoldás videó:

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

BrainB Benchmark szoftver, összeállít egy elméleti profilt, hogy az adott játékos hogyan teljesít abban a szituációbanamikor "elveszti" a karakterét egy egy intenzívebb pillanatban, pld vizuállis efektek ki takarják egymást, nagy a kavarodás stb stb. A feladatun a szimuláció alatt, hogy az egér kurzort a 'Samu Entropy' karakteren tartsuk, pontosabban a doboz közepébrn lévő körön.

Próbáljuk is ki:

```
$ git clone https://github.com/nbatfai/esport-talent-search
$ cd esport-talent-search
$ sudo apt-get install opencv-data
$ sudo apt-get install libopencv-dev
$ mkdir build && cd build
$ qmake ..
$ make
$ ./BrainB
```

Az én eredményem:

```
 NEMESPOR BrainB Test 6.0.3
time      : 6000
bps       : 54690
noc       : 44
nop       : 0
lost      :
43550 14420 1930 14290 2720 61970 31110 22890 16170 40150 45680 42910 64000 ↩
    52790 62090 51480 60170 40990 63460 61850
mean      : 39731
var       : 21055.9
```

```
found     : 0 11140 25410 38550 34750 33080 33290 23580 29210 32190 46480  ↩
   51880 23500 0 0 5240 0 16240 9580 6400 20330 16550 35330 27970 18090  ↩
   35500 38050 8490 13080 31770 24990 18910 38310 43510 41040 36520 12190  ↩
   12490 17000 18230 21350 31780 42780 46560 43120 44050 33510 28780 40510  ↩
   54300 53470 54410 54630 44960 41400 45810 43950 64000 44240
mean      : 29872
var       : 16086.5
lost2found: 29210 0 0 6400 8490 24990 12190 42780 44050 54630 44960 45810  ↩
   44240
mean      : 27519
var       : 19846.2
found2lost: 14420 14290 2720 61970 40150 45680 42910 52790 62090 51480  ↩
   63460 61850
mean      : 42817
var       : 21201.3
mean(lost2found) < mean(found2lost)
time      : 10:0
U R about 4.29297 Kilobytes
```

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

A TensorFlow nyílt forráskódu szabad szoftver, amely gépi algoritmusok leírását, és végrehajtását segíti elő. Nagyon rugalmas, sokfelé ágazódik a használhatósága pld robotika, orvoslás, információ kinyerés, beszédfelismerés stb.

A feladatban TensorFlow segítségével ismerünk majd fel kézi írással írt számjegyeket, amelyhez használnunk kell a MNIST adatbázist, itt vannak a kézírással írt számjegyek képei.

8.1. ábra. Minta

Az első python program összeszoroz két számot neurális hálók segítségével.

```
# TensorFlow Hello World 1!
# twicetwo.py
#
import tensorflow

node1 = tensorflow.constant(2)
node2 = tensorflow.constant(2)

node_twicetwo = tensorflow.mul(node1, node2, name="twicetwo")

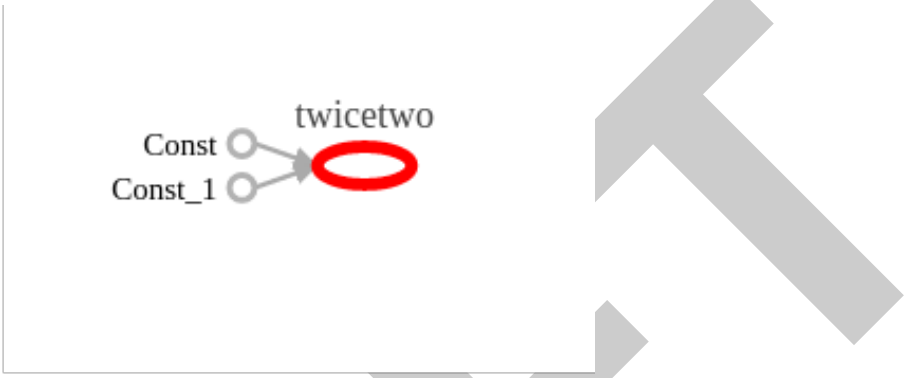
sess = tensorflow.Session()
print sess.run(node_twicetwo)

writer = tensorflow.train.SummaryWriter("/tmp/twicetwo", sess.graph)
# nbatfai@robopsy:~/Robopsychology/repos/tensorflow/tensorflow/tensorboard$ ↩
    python tensorboard.py --logdir=/tmp/twicetwo

tensorflow.train.write_graph(sess.graph_def, "models/", "twicetwo.pb",  ↩
   as_text=False)
# nbatfai@robopsy:~/Robopsychology/repos/tensorflow/tensorflow/twicetwo$  ↩
   bazel build :twicetwo
```

```
# nbatfai@robopsy:~/Robopsychology/repos/tensorflow/bazel-bin/tensorflow/ ↩
   twicetwo$ cp -r ~/Robopsychology/repos/tensorflow/tensorflow/twicetwo/ ↩
   models .
```

2*2 így néz ki:



https://www.tensorflow.org/tutorials.

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Itt olyan képeket szerkeszthetünk, a program segítségével, amit akkor látunk amikor valamilyen kábítószer, tegyükfel LSD hatása allatt állunk (tripp)

Itt a python virtuálin környezetét élvezzük, *virutalenv, venv* ami kimondottan hasznos, és rugalmas, és használata annál egyszerübb, mindössze követni kell a lent látható lépéseket, és már ki is próbálhatjuk.

Forrás:

```
'''
#Deep Dreaming in Keras.
Run the script with:
```python
python deep_dream.py path_to_your_base_image.jpg prefix_for_results
```
e.g.:
```python
python deep_dream.py img/mypic.jpg results/dream
```
'''
from __future__ import print_function

from keras.preprocessing.image import load_img, save_img, img_to_array
import numpy as np
import scipy
import argparse

from keras.applications import inception_v3
from keras import backend as K

parser = argparse.ArgumentParser(description='Deep Dreams with Keras.')
parser.add_argument('base_image_path', metavar='base', type=str,
                    help='Path to the image to transform.')
parser.add_argument('result_prefix', metavar='res_prefix', type=str,
                    help='Prefix for the saved results.')

args = parser.parse_args()
base_image_path = args.base_image_path
```

```
result_prefix = args.result_prefix

# These are the names of the layers
# for which we try to maximize activation,
# as well as their weight in the final loss
# we try to maximize.
# You can tweak these setting to obtain new visual effects.
settings = {
    'features': {
        'mixed2': 0.2,
        'mixed3': 0.5,
        'mixed4': 2.,
        'mixed5': 1.5,
    },
}


def preprocess_image(image_path):
    # Util function to open, resize and format pictures
    # into appropriate tensors.
    img = load_img(image_path)
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = inception_v3.preprocess_input(img)
    return img


def deprocess_image(x):
    # Util function to convert a tensor into a valid image.
    if K.image_data_format() == 'channels_first':
        x = x.reshape((3, x.shape[2], x.shape[3]))
        x = x.transpose((1, 2, 0))
    else:
        x = x.reshape((x.shape[1], x.shape[2], 3))
    x /= 2.
    x += 0.5
    x *= 255.
    x = np.clip(x, 0, 255).astype('uint8')
    return x

K.set_learning_phase(0)

# Build the InceptionV3 network with our placeholder.
# The model will be loaded with pre-trained ImageNet weights.
model = inception_v3.InceptionV3(weights='imagenet',
                                 include_top=False)
dream = model.input
print('Model loaded.')

# Get the symbolic outputs of each "key" layer (we gave them unique names).
```

```
layer_dict = dict([(layer.name, layer) for layer in model.layers])

# Define the loss.
loss = K.variable(0.)
for layer_name in settings['features']:
    # Add the L2 norm of the features of a layer to the loss.
    if layer_name not in layer_dict:
        raise ValueError('Layer ' + layer_name + ' not found in model.')
    coeff = settings['features'][layer_name]
    x = layer_dict[layer_name].output
    # We avoid border artifacts by only involving non-border pixels in the loss.
    scaling = K.prod(K.cast(K.shape(x), 'float32'))
    if K.image_data_format() == 'channels_first':
        loss += coeff * K.sum(K.square(x[:, :, 2: -2, 2: -2])) / scaling
    else:
        loss += coeff * K.sum(K.square(x[:, 2: -2, 2: -2, :])) / scaling

# Compute the gradients of the dream wrt the loss.
grads = K.gradients(loss, dream)[0]
# Normalize gradients.
grads /= K.maximum(K.mean(K.abs(grads)), K.epsilon())

# Set up function to retrieve the value
# of the loss and gradients given an input image.
outputs = [loss, grads]
fetch_loss_and_grads = K.function([dream], outputs)


def eval_loss_and_grads(x):
    outs = fetch_loss_and_grads([x])
    loss_value = outs[0]
    grad_values = outs[1]
    return loss_value, grad_values


def resize_img(img, size):
    img = np.copy(img)
    if K.image_data_format() == 'channels_first':
        factors = (1, 1,
                   float(size[0]) / img.shape[2],
                   float(size[1]) / img.shape[3])
    else:
        factors = (1,
                   float(size[0]) / img.shape[1],
                   float(size[1]) / img.shape[2],
                   1)
    return scipy.ndimage.zoom(img, factors, order=1)
```

```
def gradient_ascent(x, iterations, step, max_loss=None):
    for i in range(iterations):
        loss_value, grad_values = eval_loss_and_grads(x)
        if max_loss is not None and loss_value > max_loss:
            break
        print('..Loss value at', i, ':', loss_value)
        x += step * grad_values
    return x


"""Process:
- Load the original image.
- Define a number of processing scales (i.e. image shapes),
    from smallest to largest.
- Resize the original image to the smallest scale.
- For every scale, starting with the smallest (i.e. current one):
    - Run gradient ascent
    - Upscale image to the next scale
    - Reinject the detail that was lost at upscaling time
- Stop when we are back to the original size.
To obtain the detail lost during upscaling, we simply
take the original image, shrink it down, upscale it,
and compare the result to the (resized) original image.
"""


# Playing with these hyperparameters will also allow you to achieve new ↩
   effects
step = 0.01  # Gradient ascent step size
num_octave = 3  # Number of scales at which to run gradient ascent
octave_scale = 1.4  # Size ratio between scales
iterations = 20  # Number of ascent steps per scale
max_loss = 10.

img = preprocess_image(base_image_path)
if K.image_data_format() == 'channels_first':
    original_shape = img.shape[2:]
else:
    original_shape = img.shape[1:3]
successive_shapes = [original_shape]
for i in range(1, num_octave):
    shape = tuple([int(dim / (octave_scale ** i)) for dim in original_shape ↩
       ])
    successive_shapes.append(shape)
successive_shapes = successive_shapes[::-1]
original_img = np.copy(img)
shrunk_original_img = resize_img(img, successive_shapes[0])

for shape in successive_shapes:
    print('Processing image shape', shape)
```

```
    img = resize_img(img, shape)
    img = gradient_ascent(img,
                          iterations=iterations,
                          step=step,
                          max_loss=max_loss)
    upscaled_shrunk_original_img = resize_img(shrunk_original_img, shape)
    same_size_original = resize_img(original_img, shape)
    lost_detail = same_size_original - upscaled_shrunk_original_img

    img += lost_detail
    shrunk_original_img = resize_img(original_img, shape)

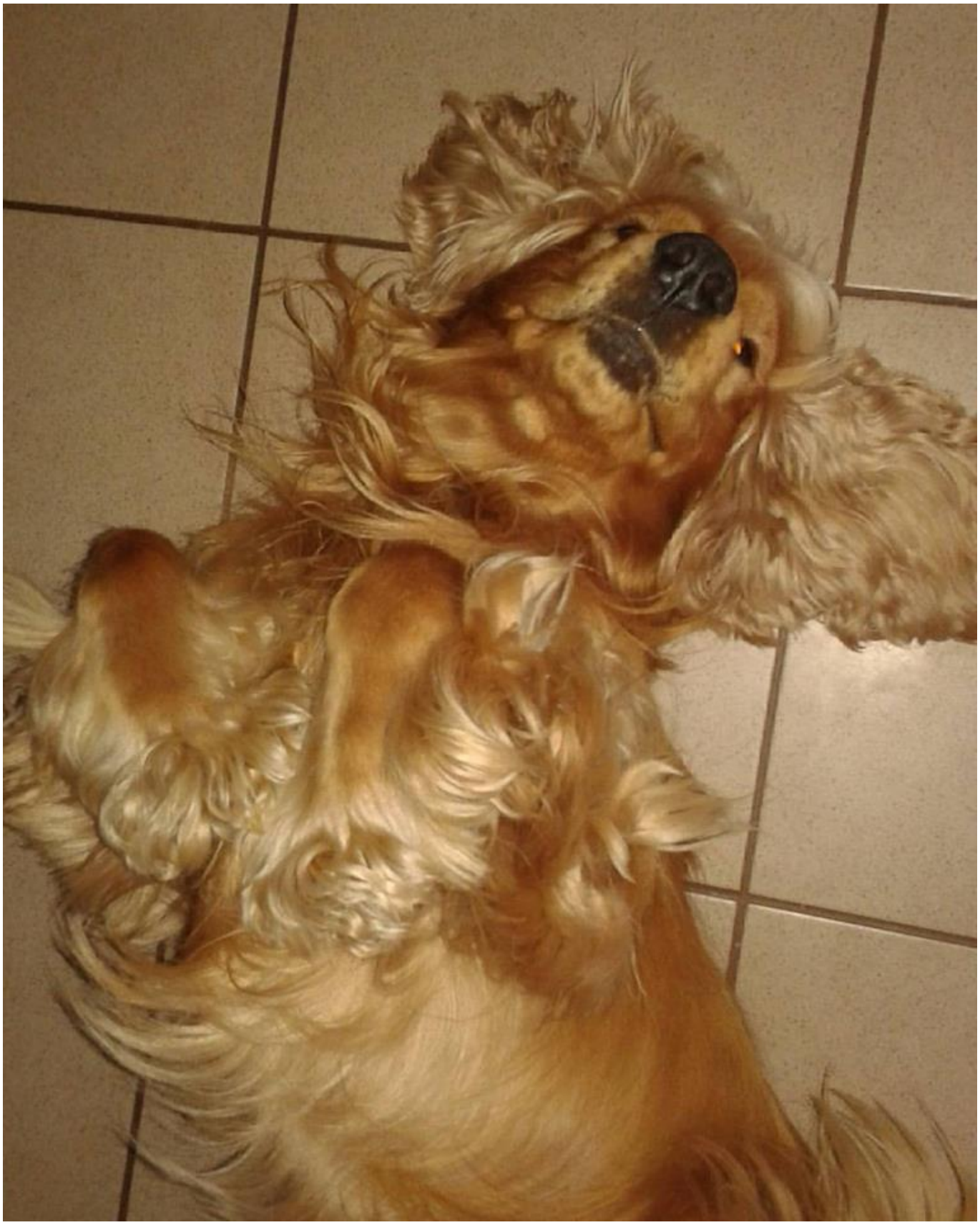
save_img(result_prefix + '.png', deprocess_image(np.copy(img)))
```

Üzemeljük be a virtuális környezetet:

```
$ virtualenv -p /usr/bin/python3 venv # Készítsük el a virtuális  ↩
   környezetet
Running virtualenv with interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /home/b1/Repos/textbook/files/svajci/deepd/venv/ ↩
   bin/python3
Also creating executable in /home/b1/Repos/textbook/files/svajci/deepd/venv ↩
   /bin/python
Installing setuptools, pip, wheel...
done
$ source venv/bin/activate # Aktiváljuk a virtuális környezetet
(venv) $ pip3 install tensorflow keras pillow # Telepítsük a Keras-t
[...]
(venv) $ pip3 freeze > requirements.txt # Mentsük el a dependenciákat
[...]
```

Csapjunk bele:

```
      $ python deep_dream.py bemeneti_kép_név.jpg kimeneti_kép_név
```



8.2. ábra. A képen a kutyusom láththató, Viliem

8.3. ábra. Itt szintén ő, a deep dream változata

A programban a paramétereket kedvünkre változtathatjuk, ezáltal erősítve, vagy gyengíve a deep dreamet

8.5. Minecraft-MALMO

Megoldás videó:

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

A Minecraft-MALMO projeckt kisérletezés a mesterséges inteligenciával, és a fiatalabb generáció ösztönzése az új poblémák megoldására, mind ezt a Minecraft segítségével. A fentebbi link a projeckt GitHub oldalának linkje, ahol remek dokumentációkat találhatunk.

```
$ cd Minecraft
$./launchClient.sh
```

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

A LISP nylevcsalád a rekurziz függvények absztakt ábrázolására lett alkotva, az 1960-as években.

LISP érdekessége, hogy prefix alakban kéri be a kifejezéseket, azaz (2 + 2) -t, (+ 2 2) módon.

A LISP használatához először is le kell telepítenünk a Lisp futtató programot, a `clisp`-t. Amit így futtatunk: `clisp fájlnév.lisp`

iteralt -> az algoritmus annyiszor hajtódik végre, amennyi a megadott szám (4)

```
            (defun factorial (N)
       (let ((R 1))
            (do ((i 1 (+ i 1) )) ((> i N) R)
                (setf r (* r i))
            )
       )
)

(print (factorial 4))
)
```

rekurziv -> addig hívja meg önmagát, míg a beadott számból ki nem jön az 1 (4 -> 1)

```
  (defun factorial (N)
       (if (= N 1)
           1
           (* N (factorial (- N 1)))
       )
)

(print (factorial 4))
)
```

```
$ clisp iter.lisp
```

```
24
$ clisp rekurz.lisp
24
```

Mindkettő esetében az eredmény okés!

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

1966, Joseph Weizenbaum megalkotta a chatboot elődjét, jóval egyszerübben működött mint az említett chatoot, tartalmazott rengeteg kifejezést, ezekre mintaválaszokat, és ezek az adatok által próbált komunikálni a felhasználóval a program

Szedjük le aprogramot: http://norvig.com/paip/README.html Indítsuk onnan ahová ki csomagoltuk.

```
$ clisp
> (load "auxfns.lisp")
> (load "eliza.lisp")
> (eliza)
ELIZA>
```

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

PICI

Programozásra kialakult nyelveknek szintjei: gépi-, assembly szintű-, és magas szintű nyelv. Egy magas szintű programozási nyelv a szintaktikai, és a szemantikai szabályok együttese határoz meg. Ahoz hogy egy programot a gép értelmezni tudjon át kell alakítani gépi nyelvé, erre két módot ismerünk meg: fordítóprogramos és interpreteres. Fordítóprogramos (- tetszőleges nyelvről tetszőleges nyelvre) módszer esetén ezek a lépések történnek meg: lexikális elemzés -> szintaktikai elemzés -> szemantikai elemzés -> kódgenerálás. interpreteres teknika esetén meg van az első három lépé (lexikális elemzés -> szintaktikai elemzés -> szemantikai elemzés), de nem készít tárgyprogramot. Hivatkozási nyelv - az adott programozási nyelv szabályrendszere. IDE - Integrated Development Enviroment -> programok írásához grafikus integrált fejlesztii környezetek, ezek magukba foglalják a szövegszerkesztőt fordítót, kapcsolatszerkeszztőt, betöltőt, futtatórendszert és betöltőt. Programozási nyelveket fel lehet bontani két csoportra (ezek mellett alcsoportokra): Imperatív- és Dekleratív nyelvek.

Imperatív nyelvek -> algoritmikus nyelv, processzort működtetik. (alcsoportok: eljárásorientált-, objektumorientált nyelvek)

Deklaratív nylevek -> nem algoritmikus nyelv, a programozó konkrétan csak a problémát adja meg.

Máselvű nylevek -> ide olyan nyelvek tartoznak, amelyek egyéb más helyre nem sorolhatóak be, egységes jellemzőjük, nincs.

Az adattipus, komponensként megjelenő programozási eszköz, amit a művelet, a tartomány (típusa) és a reprezentáció (tárban való megelenése) határoz meg. A beépített típusok megtalálhatóak minden nyelvben, de emelett mi magunk is definiálhatunk sajátot. Két nagyobb csoportja van: egyszerű, és az ősszetett. ||(skalár és strukturált)

A tömb lehet egy vagy több dimenziós, elemeire, az indexeik segítségével hivatkozunk, elemeinek típusai minden képpen ugyan olyanok, ezzel szemben a rekord elemei lehetnek különböző típusuak. A rekordoknál minden mezőnek van neve, és típusa. (Pascal, C -> a rekordoknak nincsenek almezői, egyszerű, de a mezők típusa lazán lehet összetett!) Egyes mezőkre "eszköznév.mezőnév" alakban hivatkozunk, az eszköznévre azért van szükség, mert a mező neve nem minden esetben egyedi. A mutató tárcímeket ad vissza, ez egy egyszerű típus, minden mutató egy címre mutat, kivéve a NULL speciális elem, ez nem mutat sehová.

Nevesített konstans három komponense: név, típus, érték. A nevesített konstans mindig dekralálni kell. Beszélő név szerepének köszönhetően nagyon rugalmas, és egyszerű a használata, könyen tudunk rá utalni, és ha meg akarjuk változtatni, elég csak egy helyen elvégezni a változtatásokat.

Változó négy komponense: név, cím, attributum, érték. Név -> ezzel azonosítjuk a programban. Attributum -> futtás közbeni viselkedését határozza meg a változónak, (három féle képpen lehet dekralálni: explicit, implicit, automatikus) A változóhoz cím rendelhető -> statikus tárkiosztás: futás előtt már meg van a változó címe, ami futtás alatt nem változik; dinamikus tárkiosztás: futtató rendszer végzi a cím hozzárendelését, a futás alatt változhat.

A kifejezés szintaktikai eszköz, két komponens: érték, típus. Megnéztük a felépítését. Unáris, ternális, bináris operátor, függően az operandusok számától. Prefix, infix, postfix - operátorok, és operandusok sorrendjétől függ. Bemutatja a kifejezés kiértékelését. A C kifejezéseit, precedenciatáblázatát, operátorait példák álltal mutatja be a könyv.

Negyedik fejezet bemutatja az utasítások feladatát, két nagy csoportját (deklarációs, végrehajtható). Majd be mutatta csoportosításait hét alfejezeten keresztül (értékadó utasítás, üres utasítás, ugró utasítás, elágaztató utasítások, ciklusszervező utasítások, hívó utasítás, vezérlésátadó utasítások, I/O utasítások, egyéb utasítások) ahol kitér minden esetre, szemléltetve azokat példákkal.

Az eljárás orientált nyelvekben a program szövege független szuverén részekre tagolható. Három esetet vázol a könyv: ahol a program önálló részekbél áll, ezek külön kúlön forrdíthatóak; van olyan is ahol a programot egy nagy egységként kell lefordítani, a programok fizikailag, nem függetlenek; és van ennek a kettőnek a keveréke, ahol fizikailag független, de tetszőleges belső struktúrával rendelkező programegységek léteznek.

Eljárás orientált nyelvek programegységei: alprogram, blokk, csomag, taszk. Az alprogram egy olyan eszköz ahol a bemeneti adatcsoport leképződik a kimeneti adatcsoportra, de csak a specifikációt ismerjük. Az alprogram egy nagyon rugalmasan felhasználható eszköz, elég csak a programban hivatkozni rá. Alprogramfajták: függvény, eljárás. Hívási lánc -> egy program egység meg hív egy másikat; Rekurzió -> egy programegység megívja ömagát.

Az i/o egymástól eltérő területe a programozási nyeleknek, operációs rendszer-, platform-, implementációfüggőek. Középpontban az állomány áll, lehet logikai (programozási eszköz, atributumként jelenik meg), fizikai (oprendszer szintű). Az álomány lehet inpu-, output-, input-output- állomány. I/O folyamatok alatt adatátvitel történik a tár és a perifériák között (folyamatos, rekord módú).

Ahoz hogy egy programban állományokkal dolgozzunk, a következőket kell végrehajtani: deklaráció, összerendelés, állomány megnyítása, feldolgozás, lezár.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Első fejezet, egy egyszerű C program:

```
#include <stdio.h>

main()
{
```

```
                printf("Figyelem, emberek!\n");
                }
```

Kiírja a "Figyelem, emberek!" szöveget ez a program. A fejezet meg tanítja hogyan kell futtatn, fordítani, és láthatjuk is a végereedményt.

Ezután alapvető dolgokat vettünk át: kommentelés, változók használata, while, for, if használatát, változók típusait, függvénykönyvtárak beágyazása, tömbök, stb...

Második fejezetben változóneveket veszünk át (char, int, dpuble, float); aritmetikai-, relációs- és logikai operátorok (+, -, %, -, /, =, ==, !=);

Harmadik fejezetben az utasításokat vizsgáljuk (if, if-else, case, switch), helyes használatukat, szituációs példák stb, majd do-while utasításokat. Ezekután a break, continue utasításokat nézzük meg.

Negyedik fejezet: fügvényeket rágjuk át, jó alaposan. Header álományok vizsgálata. Speciális változótípusok; Blokkstruktura, inicializálás és a rekurzió megismerése.

Ötödik fejezetben a tömbök, mutatók, függvényargumentumokról, illetve többdimenziós tömbökkel való ismerkedés.

Hatodik fejezet strukturák alapfogalmai. Aztán struktúrák és függvények kapcsolata, struktúra tömbök. Ezután "typedef" utasítást vizsgáljuk.

Hetedik fejezetben a főszerepet az adatbevitel, és az adatkivitel játsza, hogyan férhetünk hozzá más állományokhoz stb. Aztán a hibakezelésbe vetjük bele magunkat.

A nyolcadik, és egyben utolsó fejezetünkben az UNIX alapú operációs rendszerek a téma; renszerhívások, függvények, kiiratások megismerése, ebben a környezetben.

10.3. Programozás

[BMECPP]

A C++ a C nyelvből fejlesztett általános célú programozási nyelv, amelyben tudunk generikusan, és objektum orientáltan programozni.

A második fejezet bemutatja az olvasónak a C és a C++ közötti külömbségeket, a C++ újításait.

Mint például a függvényparaméterek és visszatérési érték, a main függvény, bool típus (ami a C++ nyelvben került bevezetésre), függvénynevek túlterhelése majd a paraméterátadás részletezi referenciatípussal.

A harmadik fejezetben a könyv részletezi az objektumorientáltság alapelveit, kitér a láthatóságra, konstruktorokra és destruktorokra, osztályokra, dinamikus memóriakezelésre, másolókontruktorokra, emelett friend függvények és osztálokkal if foglalkozik.

A hatodik fejezet az operátorokat általánosságban, és azok túlterhelését magyarázza. A tízedik részleteigmenően veszi a kivételkezelést, bemutatja az egymásba ágyazott try-catch működését, és használatát.

III. rész

Második felvonás

11. fejezet

Helló, Berner-Lee!

11.1. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven VS. Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II

*

A bevezetésben ki lett fejtve, hogy a Java sok mindent át vett a C++ nyelvtől, sok figyelmet fordított a magbízhatóságra és felhívta a figyelmet a nyelvi szerkezetben felfedezhető újdonságokra, pld.: a mutatók helyett a referenciák használatára, futató környezet korlátozásai.

*

A Java nylev teljesen objektum orientált (objektumok, és ezek mintáinak tekinthető ostályok összesége).

Egy osztály két dologból állhat: mezőkből, azaz változókból, valamint metódusokból, melyeket erőszerettel hívunk függvényeknek, holott nem mindig van visszatérési értékük. Mezőkben tároljuk az a datokat, a metódusokkal pedig az adatokon végezhető műveletek kódját adjuk meg.

Megnéztük a HelloVilág programot, Javaban:

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello Világ");
  }
}
z objektumorientált paradigma alapfoglamai
```

És azt le is fordítottuk:

```
oszuszki@oszuszki-richard:~/javakonyvszarnyprobalgatasok$ javac Hello.java
oszuszki@oszuszki-richard:~/javakonyvszarnyprobalgatasok$ java Hello
```

Megfigyelhettük, hogy a Java fordítóprogram egy bájtkódnak nevezett formátumra fordítja le a forráskódot, melyet a Java Virtuális Gép önálló interpretensként fog értelmezni. Biztonsági szempontból előnyös, de sebesség terén hátrányos, ez az eljárás.

Aztán kielemeztük minden részét a példa kódnak, `main`, a program fő metódusa, az azt követő `static` kulcs szó lehetővé teszi, hogy a Hello ostály futtatása során, nem jön létere egytlen objektum sem, `public`, ha nem szerepelne a kódban, akkor a Java meggátolná a program futását, ez teszi lehetővé, hogy a metódus, a külvilág számára is látható legyen, `void`, nem-függvény jellegű metódusra, azaz a visszatrési érték elhagyására utal, `String[]`, szövegtömbön keresztül kerülnek átadásra a parancssor argumentumai.

`System` osztály `out` objektumának `println` metódusa kerül meghívásra, mely a paraméterként kapot szöveget, kiírja a szabványos kimenetre, egy újsor karakterrel kiegészítve azt.

Java nyelv egy figyelemre méltó újdonsága, az ASCII karrakterészletből hiányzó karaktereket, nemcsak a szövegliterátorokba, hanem a megjegyzésekben, sőt az azonosítókban is lehet használni.

*

A Java nyelv teljesen objektumorientált. Egy osztály két dologból tevődik össze: mezők(adatok tárolása), metódusok(adatokon végezhető műveletek kódja.

Futtatáshoz szükségünk van egy Java nevű fordítóprogramra. Ez egy bájtkódnak nevezett formátumra fordítja le a forráskódot, amelyet majd a Java Virtuális Gép önálló interpreterként fog értelmezni. Ez lassabb futást eredményez. A legnagyobb különbség a Java illetve a C++ között itt mutatkozik. A C++ kézenfekvőbb, ha olyan programokat szeretnénk írni, amelyeknél a futás sebessége elsődleges szempont. A Java inkább a web-es téren elterjedt.

A Java nyelvnek vannak egyszerű típusai is, melyeket az adatok egyszerű reprezentálására lehet használni. Ezeknek értéket adni az '=' operátorral lehet. Ilyenkor ez valódi értékadást jelent, összetett típusok esetében csak egy referencia átmásolását jelenti.

Eltérés még a C++ és a Java között, hogy Java-ban már 16 bites Unicode karaktereket is lehet használni változók vagy konstansok deklarálásához, tehát használhatunk ékezetes karaktereket, görög ábécé betűit, stb.

Megjegyzéseket ugyanúgy adhatunk hozzá a kódhoz, mint a C++ esetében.

A Java nyelveben semmilyen explicit eszköz nincs egy objektum megszüntetésére, egyszerűen nem kell hivatkozni rá és magától meg fog szűnni. Hivatkozás megszüntetéséhez az eddigi referencia helyett a null referenciát adjuk neki értékül.

A Java nyelvben tömböket a [] jelöléssel lehet megadni. A C++-tól eltérően ez egy igazi típus lesz és nem csak a mutató típus egy másik megjelenítési formája. A tömb típusok nem primitív típusok, a tömb típusú változók objektumhivatkozást tartalmaznak. Eltérés még a C++ és a Java között, hogy a Java-ban nincsenek többdimenziós tömbök, erre a tömb a tömbben megoldást használja.

További eltérés még a Goto utasítás teljes hiánya, ez a Java nyelvből kimaradt.

Java-ban egy objektumot a következő képen példányosítunk.

```
#Adott egy objektum
public class Alkalmazott {
String nev; int fiztes;
void fizetesemeles (int novekmeny) {
    fizetes += novekmeny
    }
```

```
}
#Példányosítás
Alkalmazott a = new Alkalmazott();
```

A new operátor mögött adjuk meg, h melyik osztályt példányosítjuk. A zárójelek közék közé az adott esetben a konstruktornak szánt paramétereket írjuk. Csakúgy, mint a C++-ban, itt is vannak nyilvános (public) és privát (private) tagok. A private tagokhoz az adott osztályon kívül nem férhet hozzá semmi.

Java megkülönbözteti a referenciát és az imperatív programozási nyelvekben használt mutatót aképpen, hogy a kifejezésekben automatikusan a mutatott objektumot jelenti, nem pedig a címet.

11.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba.

A Python egy általános célú programozási nyelv. Alkotója Guido van Rossum. Leginkább prototípus készítésre és tesztelésre szokás alkalmazni. Képes együttműködni más nyelveken íródott modulokkal. Elterjedtségét főként az imént említett tulajdonságának, valamint a rövid tanulási ciklusnak köszönheti. A Python-ban megírt programok terjedelmükben jóval rövidebbek, mint a C, C-ben, C++-ban vagy Java-ban készült, velük ekvivalens társaik. A kódok tömörek és könnyen olvashatóak. A kódcsoportosításhoz új sort és tabulátort használ (behúzásalapú), nincs szükség nyitó és zárójelekre, nincs szükség változó vagy argumentumdefiniálásra. Az utasítások a sorok végéig tartanak, nincs szükség ';'-re, ha egy adott utasítást a következő sorban szeretnénk folytatni, akkor ezt a '\' jellel tehetjük meg.

Példa:

```
if feltétel1 és feltétel2
    alapfeladat()
egyébfeladat()
```

Az értelmező a sorokat tokenekre bontja, amelyek közt tetszőleges üres (whitespace) karakter lehet. A tokenek lehetnek: azonosító(változó, osztály, függvénymodul), kulccszó, operátor, delimiter, literál.

Lefoglalt kulcsszavak: and, del, for, is, raise, assert, elif, from, lambda, return, break, else, global, not, try, class, except, if, or, while, continue, exec, import, pass, yield, def, finally, in, print.

Típusok

A Pythonban minden adatot típusok reprezentálna. Az adatokon végezhető műveleteket az objektum típusa határozza meg. A változók típusait a rendszer futási időben "kitalálja".

Számok lehetnek:egészek(decimális, oktális, hexadecimális), lebegőpontosak és komplex számok.

Sztringeket aposztrófok közé írva adhatunk meg, illetve a 'u' betű használatával Unicode szövegeket is felvehetünk.

Példa:

```
print u"Hello %s, kedves %s!"
```

Az ennesek (tuples) objektumok gyűjteményei vesszővel elválasztva.

Példa:

```
('a','b','c') #három elemű ennes
tuple('abc') #tuple generálás kulcsszóval (ugyanaz, mint az előző)
() #üres ennes
(1, "szia", 3,) #három elemű ennes, az utolsó vessző elhagyható
```

A lista különböző típusú elemek rendezett szekvenciája, elemeit szögletes zárójelek közé írjuk, dinamikusan nyújtózkodik. Az elemeket az indexükkel azonosítjuk.

Példa:

```
[a','b','c'] #három elemű lista
list('abc') #lista generálás kulcsszóval (ugyanaz, mint az előző)
[] #üres ennes
[1, "szia", 3,] #három elemű lista, az utolsó vessző elhagyható
```

A szótár kulcsokkal azonosított elemek rendezetlen halmaza. Kulcs lehet: szám, sztring stb.

Példa:

```
{'a':1, 'b':5, 'e':1982}
{1:1, 2:1, } #az utolsó vessző elhagyható
{} #üres szótár
```

Változók Pythonban - egyes objektumokra mutató referenciák. Típusaik nincsenek. A hozzárendelést az '=' karakterrel végezzük. A del kulcsszóval törölhető a hozzárendelés, a mögöttes objektum törlését a garbage collector fogja elvégezni. Léteznek globális és lokális változók (akárcsak a C++-ban). A változók közötti típuskonverzió támogatott.

A szekvenciákon (sztringek, listák, ennesek) műveletek végezhetőek el (összefűzés, szélsőértékhelyek meghatározása stb).

Az elágazások és ciklusok működése megegyezik a többi programnyelvével.

Függvényeket a def kulccszóval definiálunk, tekinthetünk rájuk úgy, mint értékekre, mivel továbbadhatóak más függvényeknek illetve objektumkonstruktoroknak.

```
def hello();
    print "Hello world!"
    return
```

A Python támogatja a klasszikus, objektumorientált fejlesztési eljárásokat (osztályokat definiálunk, amelyek példányai az objektumok. Osztályoknak lehetnek attribútumaik(objektumok, függvények) illetve örökölhetnek más osztályokból.

A kivételkezelés:

Példa

```
try:
    utasítások
except [kifejezés]: #akkor fut le,ha az történik, amit a try után leírtunk
[else:
    utasítások]
```

12. fejezet

Helló, Arroway!

12.1. OO szemlélet

Az objektumorientált programozás fő alapelve, hogy osztályokat hozunk létre, ezekkel dolgozunk, az osztályok elemeit, objektumoknak nevezzük, és az az objektumok rendelkeznek az osztály öszess tulajdonságával. Ezzel megkönyítjük a munka menetét, mivel az osztály létrehozásakor megadott tulajdonságokkal az összes benne megadott elem fog rendelkezni.

A lenti példa elején létre hozunk egy PolarGenerator nevű osztályt, amiben két változó, és egy függvény található. A nincsTarolt változó lesz felelős azért, hogy van e változó eltárolva, amennyiben nincs, a függvény generál két véletlenszerű számot, az egyiket eltárolja, a másikat visszaadja, viszont, ha viszont van eltárolt változó, akkor azt fogja vissza adni. A feltételes utasítás mindkét ágában negálja a nincsTarolt változó értékét.

```
public class PolarGenerator {

  boolean nincsTarolt = true;
  double tarolt;

  public PolarGenerator(){
    nincsTarolt = true;
  }

  public double kovetkezo(){
    if(nincsTarolt){
      double u1, u2, v1, v2, w;
      do{
        u1 = Math.random();
        u2 = Math.random();
        v1 = 2 * u1 - 1;
        v2 = 2 * u2 - 1;
        w = v1 * v1 + v2 * v2;
        } while (w>1);
        double r = Math.sqrt((-2 * Math.log(w))/w);
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;
```

```
        return r * v1;
        } else {
        nincsTarolt = !nincsTarolt;
        return tarolt;
        }
      }

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

12.2. "Gagyi"

Megoldás forrása:

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

```
            public class Gagyi
{

public static void main (String[]args)
{

Integer x = 333;
Integer t = 333;

System.out.println (x);
System.out.println (t);

while (x <= t && x >= t && t != x);

}

}
```

A while ciklusba deklarált értekek, amennyiben 128-tól nagyobb vagy -128-tól kissebb értéket vesznek fel, a ciklus egy végtelen ciklusba fordul át, ellenkező esetben leáll. A két értéknek ugyan annyinak kell lennie.

```
            public class GagyiL
{

public static void main (String[]args)
{

Integer x = 66;
Integer t = 66;

System.out.println (x);
System.out.println (t);

while (x <= t && x >= t && t != x);

}

}
```

12.3. Yoda

Írtunk egy programot, amely java.lang.NullPointerEx-el leáll, ha eltérünk a Yoda conditiontől, ami arról szól, hogy a feltétel sorrendje megfordul, baloldalon, a konstans foglal helyet.

```
public class Yoda {
    public static void main(String[] args)
    {
        String myname = null;
        if (myname.equals("Ricsi")) {
      System.out.println(""BingBengBong"");
    }
    }
}
```

```
            oszuszki@oszuszki-richard:~/bhax-textbook/_files_/_arroway_$  ↩
               javac Yoda.java
            oszuszki@oszuszki-richard:~/bhax-textbook/_files_/_arroway_$  ↩
               java Yoda
            Exception in thread "main" java.lang.NullPointerException
            at Yoda.main(Yoda.java:6)
```

A myname, nem változó, hanem egy null pointer, ezáltal adja a NullPointerExeption-t, mivel a nullpointert akarjuk hasonlítani a "Ricsi" konstanshoz.

Viszont, ha megfordítjuk a feltétel sorrendjét, akkor fog működni a Yoda condition, String literátorhoz már hasonlíthatunk null értéket, ami a mi kódcsipetünkben false értéket fog adni. Ezzel a módszerrel értékadás-hasonlítás vagy a null értékekből eredő problémákra tudunk megoldást találni...

```
public class Joda {
    public static void main(String[] args)
    {
        String myname = "Ricsi";
        if ("Ricsi".equals(myname)) {
      System.out.println("BingBengBong");
    }
    }
}
```

12.4. Kódolás from scratch

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

```
public class PiBBP {

    String d16PiHexaJegyek;

    public PiBBP(int d) {

        double d16Pi = 0.0d;

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

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

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

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

        while(d16Pi != 0.0d) {

            int jegy = (int)StrictMath.floor(16.0d*d16Pi);

            if(jegy<10)
                sb.append(jegy);
            else
```

```
                sb.append(hexaJegyek[jegy-10]);

            d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
        }

        d16PiHexaJegyek = sb.toString();
    }

    public double d16Sj(int d, int j) {

        double d16Sj = 0.0d;

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



        return d16Sj - StrictMath.floor(d16Sj);
    }

    public long n16modk(int n, int k) {

        int t = 1;
        while(t <= n)
            t *= 2;

        long r = 1;

        while(true) {

            if(n >= t) {
                r = (16*r) % k;
                n = n - t;
            }

            t = t/2;

            if(t < 1)
                break;

            r = (r*r) % k;

        }

        return r;
    }

    public String toString() {

        return d16PiHexaJegyek;
```

```
    }

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

1995-ben felfedezett, Baliey-Browein-Plouffe formula, a pi kiszámítására létrehozott algoritmus, mely a szerzőiről kapta a nevét.

A fenti program a pi szám hexadecimális értékét fogja visszaadni.

Ezzel az algoritmussal működik:

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right].$$

Láthatóan le is fut, és kiírja a pi hexadecimális értékét:

```
oszuszki@oszuszki-richard:~/bhax-textbook/_files_/_arroway_$ javac  ↩
   PiBBP.java
oszuszki@oszuszki-richard:~/bhax-textbook/_files_/_arroway_$ java PiBBP
6C65E5308
```

IV. rész

Irodalomjegyzék

12.5. Általános

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

12.6. C

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

12.7. C++

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

12.8. Lisp

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

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

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