Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert	2019. szeptember 16.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-02-27	Hello Turing!	mariannranyhoczki
0.0.6	2019-03-06	Hello Chomsky!	mariannranyhoczki
0.0.7	2019-03-13	Hello Caesar!	mariannranyhoczki
0.0.8	2019-03-20	Hello Mandelbrot!	mariannranyhoczki

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-03-27	Hello Welch!	mariannranyhoczki
0.0.10	2019-04-03	Hello Conway!	mariannranyhoczki

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

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!

Egy mag 100 százalékban:

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

  return 0;
}
```

Azt,hogy a megírt ciklus valóban 1 magot használ 100 százalékon egy másik terminálban top paranccsal ellenőrizhetjük. A folyamatot a Ctrl+C billentyűkombinációval lőhetjük le legegyszerűbben.

Egy mag 0 százalékban:

```
#include <unistd.h>
int
main ()
{
  for (;;)
    sleep(10000);

  return 0;
}
```

Ha azt szeretnénk hogy egy végtelen ciklus ne használja a magot akkor "elaltathatjuk" a sleep függvénnyel mely után a zárójelben megadott szám az altatás idejét jelzi másodpercekben. Másik terminálban a top paranccsal ellenőrizhetjük hogy valóban nem használ-e egy magot sem.

Minden mag 100 százalékban.

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

Ha azt szeretnénk hogy a program több szálat használjon használhatjuk az OpenMp könyvtárat de ekkor a fordítás a -fopenmp kapcsolóval történik. Ilyenkor a programot a parancs szerint párhuzamosan , egyszerre több szálon futtaja. A helyes működést a rendszerfigyelőben ellenőrizhetjük és ha jól működik a program láthatjuk hogy több CPU-t is használ.

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:

```
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

```
#include<stdio.h>

int main()
{
int a=7, b=19;
  printf("a=%d b=%d\n",a,b);

  a=(a-b);
  b=(a+b);
  a=(b-a);

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

return 0;

}
```

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

Két változó értékének felcserélését segédváltozó nélkül is könnyen megtehetjük, csupán egy kis matematika szükséges hozzá. Először az 'a' változót egyenlővé teszem 'a-b'-vel, majd a 'b' változót 'a+b'-vel, végül pedig az 'a' változót 'b-a'-val. Így az értékek felcserélődnek és a printf segítségével kiirathatjuk őket a terminálba. A % jel után pedig d betűt írunk, ezzel jelezve hogy a kiírt számok decimális, előjeles egészek lesznek.

2.4. Labdapattogás

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

```
Program T1000

#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;
}
```

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

Ennél a vizuális feladatnál a labdánkat egy karakterrel ,a 0-val jelöljük. A feladat az hogy ha a képernyő széléhez ér a labda akkor visszapattanjon, ezért használjuk az ncurses könyvtárat,amely tartalmazza a getmaxyx-et amellyel a terminál kiterjedéseit adjuk meg.Az if feltételekkel adjuk meg a labdának azt ,hogyha elérte a terminál kiterjedését akkor pattanjon vissza.

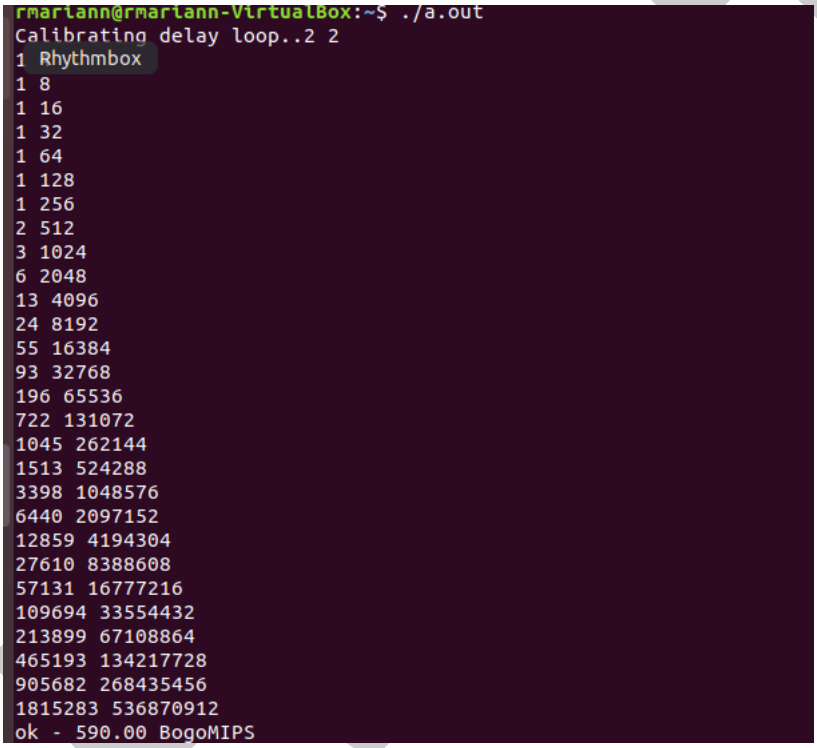
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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!

```
Program T100

#include <time.h>
#include <stdio.h>

void
delay (unsigned long long int loops)
{
unsigned long long int i;
for (i = 0; i < loops; i++);
}

int
main (void)
{
unsigned long long int loops_per_sec = 1;
unsigned long long int ticks;

printf ("Calibrating delay loop..");
fflush (stdout);

while ((loops_per_sec <<= 1))
{
ticks = clock ();
delay (loops_per_sec);
ticks = clock () - ticks;

printf ("%llu %llu\n", ticks, loops_per_sec);

if (ticks >= CLOCKS_PER_SEC)
{
loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
(loops_per_sec / 5000) % 100);

return 0;
}
}

printf ("failed\n");
return -1;
}
```

A MIPS a Million Instructions Per Secon rövidítése. Ez a számítógép számítási sebességének a mértékegysége. A kernelnek szüksége van egy időzítő ciklusra, amit a processzor sebessége alapján határoz meg.Ezért a kernel megméri hogy egy bizonyos ciklus mennyi idő alatt fut le a számítógépen. A Bogo az angol bugus szóból ered amelynek jelentése hamis és azért adták neki ezt a nevet mert csak egy megközelítő értéket ad de nem pontos. A program futásakor az értéket a loops_per_sec változóba teszi.

Nekem a következő eredmény jött ki



2.6. Helló, Google!

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

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

void
kiir (double tomb[], int db)
{
  int i;

  for (i = 0; i < db; ++i)
    printf ("%f\n", tomb[i]);
}
```

```
double
tavolsag (double PR[], double PRv[], int n)
{
  double osszeg = 0.0;
  int i;

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

  return sqrt(osszeg);
}

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 };

  int i, j;

  for (;;)
    {

      for (i = 0; i < 4; ++i)
  {
    PR[i] = 0.0;
    for (j = 0; j < 4; ++j)
      PR[i] += (L[i][j] * PRv[j]);
  }
      if (tavolsag (PR, PRv, 4) < 0.00000001)
  break;

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

    }

  kiir (PR, 4);

  return 0;
}
```

A Page-Rank egy olyan a Google által létrehozott program amely a weboldalakat minőségük, értékelésük szerint rangsorolja. Megnézi, hogy milyen weboldalról mennyi link mutat és annak szavazata mennyit ér. Minnél több és erősebb linkek mutatnak egy weboldalról annál jobb és fontosabb lesz az oldal.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

A Brun-tétel szerint az ikerprímek (azaz az olyan prímek melyeknek különbsége 2 és egymás után következnek) reciprokának az összege konvergál egy értékhez, melyet Brun-konstansnak nevezünk.

A következő program/szimulácuó R nyelven íródott.

```
sumTwinPrimes <- 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 = sumTwinPrimes)
plot(x,y,type="b")
```

A prímszámokkal való számítás miatt meghívjuk a matlab csomagot. Egy függvény létrehozásával kezdünk, majd a primes(x) fügvénnyel a prészámokat adjuk meg. A diff vektor tartalmazza azoknak a prímszámoknak a különbségét amelyek egymás után következnek. Az idx vektor az ikerprímeket tartalmazza. A t1primes és a t2 primesben vektorokban tároljuk a 2 ikerprímet. A returnben sum fügvénnyel visszaadjuk az eredményt. A plottal megrajzolva látható, hogy a számok ténylegesen egy értékhez tartanak.

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

A Monty Hall probléma arról szól hogy ha van 3 ajtóm és az egyik mögött nyeremény van akkor ha választunk egyet és a választásunk után kinyitnak egy másik ajtót amiben nem található nyeremény akkor

ugyanannyi esélyünk marad e nyerni az elsőre választott ajtóval mintha megváltoztatnánk a választásunkat. A számítógépes szimuláció alapján számít, hogy másik ajtót választunk, ugyanis ekkor megduplázódik az esélyünk a nyerésre.

Az első ajtó kinyitásakor 1/3 az esélye, hogy a kinyitott ajtó a nyertes és 2/3, hogy nem az. Ezután egy üres ajtó kinyitásakor ha nem változtatunk a döntésünkön, akkor 1/3-ad marad az esélye a nyerésnek,de ha változtatunk akkor a szimuláció alapján az esélyünk 2/3-adra növekszik.

Ennek bizonyítását láthatjuk R nyelven megfogalmazva, egy televíziós műsorra levetítve:

```
kiserletek_szama=1000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)
for (i in 1:kiserletek_szama) {
    if(kiserlet[i]==jatekos[i]){

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

    }else{

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

    }
    musorvezeto[i] = mibol[sample(1:length(mibol),1)]
}
nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)
for (i in 1:kiserletek_szama) {
    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]

}
valtoztatesnyer = which(kiserlet==valtoztat)
sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Először meg kell adnunk az elvégezendő kísérletek számát, melyeket a for ciklusban vizsgalunk majd. Ezután megadjuk az ajtó számát melyet a játékos választott, majd for ciklussal vizsgáljuk ,hogy a játékos eltalálta-e elsőre a jó ajtót, ez akkor történik meg ha a kisérlet tömb első eleme megegyezik a játékos tömb első elemével. Ha a játékosnak ez sikerült, akkor a mibol vektorba beletesszük azokat az értékekekt amelyeket a játékos nem választott. Ha nem sikerült eltalálni a helyes ajtót akkor a mibol vektorba szintén 2 érték kerül, az egyik az ajtó a nyereménnyel a másik pedig az az ajtó ,ami nem nyer ,de nem is választotta a játékos.

A nemvaltoztatesnyer vektor tárolja majd az eseteket amikor első próbálkozásra sikerült a nyereményt választani a játékosnak, azaz ha a kísérlet tömb eleme megegyezik a játékos elemével. A valtoztatesnyer

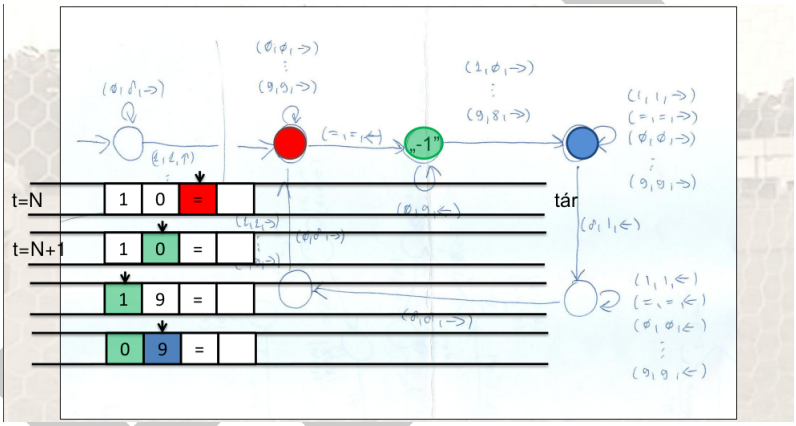
vektorba pedig megkapjuk azokat az eseteket amikor a játékos változtatással nyert. Eztuán a lenght-el kiiratjuk az esetek számait.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

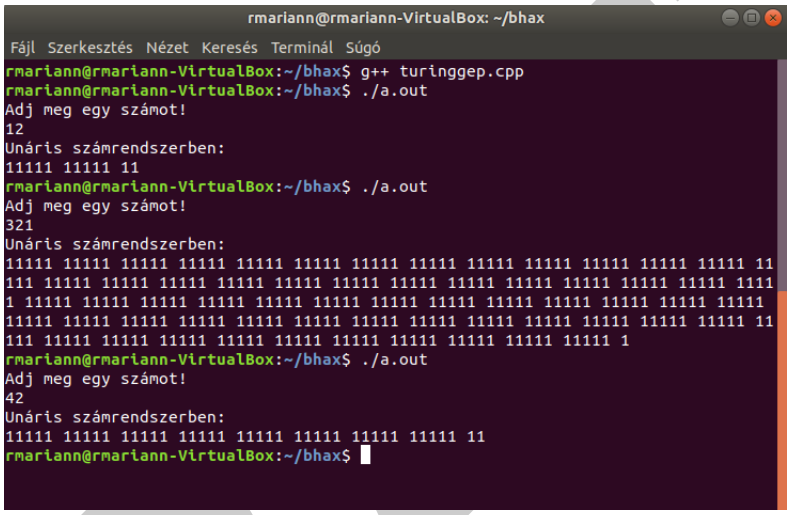


A feladatom egy decimális ,azaz tízes számrendszerből egy unáris, azaz egyes számrendszerbe váltó gép írása. Az unáris a legegysezreűbb számrendszer amely a természetes számok ábrázolására szolgál. A célja hogy egy szám beírása után annyi vonal jelenjen meg amennyi maga a szám értéke. A következő Turing gépben a vonalak helyett 1-esek fognak megjelenni. Működési elve hogy a számok beolvasása után ha talál egyenlőségjelet akkor az előtte lévő számból kivon 1-et, amíg az 0 nem lesz. Minden kivont 1-es után kiír egy vonalat (esetünkben egy 1-est). A végső sorozatban az egyesek számának pontosan meg kell egyezni a a megadott szám értékével.Az itt látható programban minden 5 darab 1-es után szóköz jelenik meg de ez nem számít egy kivont értéknek.

```
#include <iostream>
using namespace std;
int main() {
    int szam;
    int szamlalo = 0;

    cout << "Adj meg egy számot!\n";
    cin >> szam;
    cout << "Unáris számrendszerben:\n";
```

```
    for(int i = 0; i < szam; ++i) {
        cout << "1";
        ++szamlalo;
        if(szamlalo%5 == 0)
            cout << " ";
    }
    cout << '\n';
    return 0;
}
```

A következő képen a kód tesztelése látható:



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

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

Első előadás feladat passzolása

3.3. Hivatkozási nyelv

A C nyelvnek rengeteg szabványa van,de leggyakrabban a C18-as szabványt használjuk. Mivel több verzió van ezért van hogy az egyik verzióban megírt program nem biztos hogy kompatibilis lesz a másik verzió fordítójával.Például a C89-es szabványban a for ciklus fejében deklarált változót a fordító nem fordítja le, valamint a // kommentjelet sem ismeri ezért a /**/ komment jelölést kell használni és a printf függvény itt sprintként ismert.

```
#include <stdio.h>
#include <math.h>
int main ()
{
   char str[80];
```

```
    sprintf(str, "A Pi értéke: %f", M_PI);
    puts(str);
    return(0);
}   //Ez a program kiírja a standard kimenetre a pi ↩
    értékét
```

3.4. Saját lexikális elemző

Lexikális elemzőre főként akkor van szükségünk hogyha nagyon adattömeggel dolgozunk. Egy elemzővel könyebben kinyerhetők statisztikai adatok,szavakat tudunk osztályozni és például grafikus ábrázolásra is alkalmas.

```
import nltk
from nltk.tokenize import word_tokenize
text = open('elemzo.txt').read()
tokens = word_tokenize(text)
digit_count = len(set(word for word in tokens if word.isdigit() ↩
   ))
print('The number of digits in the text: ')
print(digit_count)
```

Miután az elem.txt tartalmát odaadtuk a text változónak szavakra bontjuk a stringet és megkeressük azokat a stringeket amik csak számból állnak.A set függvény miatt azokat a megegyező (duplikált) stringeket csak egynek veszi. Ezután a len() függvénnyel megkapjuk a számból álló stringek számát melyet beleírunk a digit_count változóba.A végén pedig printf-el kiíratjuk az eredményt.

3.5. l33t.l

Egy új írási mód terjedt el a 80-as években annak érdekében hogy bizonyos fájlokra kulcsszavas keresés során nehezebb legyen rátalálni.Ekkor az ABC betűit olyan karakterekre cserélték amellyel a szöveg olvasható maradt,de nehezebben megtalálható. Például az A betűt 4-esre cserélték vagy a C-t nyitó zárójelre. Ezt az írásmódot, leetspeak-nek nevezték el.

A következő egy példa az 1337-es kódolóra.

```
import nltk
text = open('text.txt').read()
text = text.lower()
text = text.replace('a', '4')
text = text.replace('b', '8')
text = text.replace('c', '(')
text = text.replace('e', '3')
text = text.replace('g', '6')
```

```
            text = text.replace('i', '1')
            text = text.replace('l', '|')
            text = text.replace('o', '0')
            text = text.replace('s', '5')
            text = text.replace('t', '7')
            text = text.replace('x', '8')
            text = text.replace('z', '2')
            print(text)
```

Az előző program pythonban íródott .Először importáljuk az nltk könyvtárat.A kódunk a kapott inputban kicseréli a betűket a megadott karakterekre, majd ezeket elmenti és a printf segítségével megjeleníti az outouton. Erre példa: "informatika -> 1nf0rm471k4"

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?

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

Egy for ciklus, melyben a változó értékét 4-ig növelem. A ++i-vel a változó értéke mindig eggyel nő.

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

Egy for ciklus, melyben a változó értékét 4-ig növelem. A i++-al a változó értéke mindig eggyel nő. Ha egy kifejezésben csak egy változó szerepel, akkor használhatjuk a ++i-t és az i++-ot is,de kifejezés kiértékelésénél már számít melyiket használjuk.

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

Egy for ciklus 0-tól 4-ig,de a végrehajtás sorrendje hibás, mert értékadás történik az i változó növelésénél.

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

For ciklus, mely n-szer fut le és a d pointer értékét odaadja az s pointernek, majd növeli a változók értékét

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

A printf-el az outputra kiírunk két fügvényvisszatérési értéket. De ez nem ajánlott mert nem tudhatjuk melyik fog előbb lefutni és melyik érték melyik függvényhez tartozik.

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

Kiíratjunk az outputra decimális formában az f(a) függvény visszatérési értékét és az a változót.

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

Kiíratjunk az outputra decimális formában az f(a) függvény visszatérési értékének referenciáját és az a változót.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

I. Minden x-re létezik olyan y, amely nagyobb nála és egy prímszám.

II. Minden x-re létezik olyan y, amely nagyobb nála, és y is és y+2 is egy prímszám.

III. Minden x-re létezik olyan y , hogy x prím, ha x kisebb y-nál.

IV. Minden x-re létezik olyan y, amely kisebb nála ha x nem prím.

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;
  ```

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

  Egész típusú változóra mutató pointer.
- ```
  int &r = a;
  ```

  Egy egész típusú referencia, mely megkapja a értékét.
- ```
  int c[5];
  ```

  Egy 5 elemű tömb mely egészeket tartalmaz.
- ```
  int (&tr)[5] = c;
  ```

  Egy ötelemű tömb referenciája mely a c tömb címére hivatkozik.
- ```
  int *d[5];
  ```

  5 elemű tömb, mely pointereket tartalmaz.
- ```
  int *h ();
  ```

  Függvény, melynek a visszatérérsi érteke egy pointer.
- ```
  int *(*l) ();
  ```

  Egy pointer mutat egy mutató pointert visszaadó függvényre.
- ```
  int (*v (int c)) (int a, int b)
  ```

  Függvény, melynek ha megadunk két értéket, visszaad egy egész típusú változót, majd visszatér a változóra mutató pointerrel.
- ```
  int (*(*z) (int)) (int, int);
  ```

Mivel az előző kódok referenciák ezért a g++ paranccsal fordíthatóak.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

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

A háromszögmátrixok olyan mátrixok melyek ugyan négyzet alakúak de főátlójuk alatt vagy felett csak 0 érték található. Jelen esetben alsó háromszögmátrixról van szó, melynek fátlója FELETT vannak a nullértékek.

```
#include <stdio.h>
#include <stdlib.h>
int
main ()
{
    int nr = 5;
    double **tm;
    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }
    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL ↩
           )
        {
            return -1;
        }
    }
    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");
    }
    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;
}
```

Az nr változó megadja a mátrix magasságát, majd a malloccal megadott méretű területet foglalunk. Ezt úgy tesszük hogy a double* méretét megszorozzuk az nr változóval és az így kapott számú bitnyi helyet foglalunk(jelen esetben 5*8=40).Ezután a for ciklusban a mutatók értékét mindig 1- el növeljük, majd értékeket adunk a változóknak.Végül a printf segítségével kiíratjuk őket majd a free-vel felszabadítjuk a területeket.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Ebben a feladatban a kizáró vagyos (XOR) titkosítás van bemutatva.A kizáró vagy művelet 1 értéket ad, ha a két bit különböző és 0-t ha megegyező. Például:

A xor titkosítás nem a legbiztonságosabb titkosítási mód, de kevésbé fontos adatok titkosítására tökéletesen megfelel.

```
#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);
    }
}
```

A program elején meg kell adnunk a kulcs és a buffer méretét.VAlamint létrehozunk egy kulcs változót, amellyel nyomon tudjuk követni hogy a titkosításnál melyik karakternél járunk és létrehozunk egy másik változót is amely megadja hogy hány bájtot olvasunk be az inputról. Ezután a while ciklusban karakterenként megtörténik a titkosítás. Ha a folyamatban a kulcs végéhez értünk kezdődik előről mindaddig amíg a szövegünk titkosítva nem lesz. , majd kiirathatjuk. Az algoritmusnak az a feladata hogy összehasonlítsa az előre megadott kulcsban lévő értékeket a bitsorozatban lévőkével.Ha az érték különböző akkor 0-t kapunk, ha pedig ugyanaz akkor 1-et, mely bitsorozatból a xor művelet használata után a titkosítatlan adatokat kapjuk meg.

aAz előző algoritmus fordításkor a c99 szabványt kell használnunk:

```
            gcc exor.c -o e -std=c99
        x
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

```
public class ExorTitkosító {

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

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;
        while((olvasottBájtok =
                bejövőCsatorna.read(buffer)) != -1) {

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

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

```
            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

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

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Mivel a Java egy objektumorientált nyelv ezért itt classokkal dolgozuk. A String[] args-el átadhatóak a programnak a parancssori argumentumok.A try-ban tároljuk a ülönböző utasításokat és ha hibát talál azt jelzi.A catch "elkapja" a hibát és egy hibaüzenetet ad vissza. A try -ban helyet foglalunk az ExorTitkosito függvénynek amely eztán megkapja a kimenetet a bemenetet és a kulcsot is. Létrehozunk egy kulcs és egy buffer tömböt majd a kulcs tömbbe beolvassuk magát a kulcsot a getBytes segítségével. A while ciklus addig tart amíg már nem tud többet beolvasni vagy addig ameddig el nem éri a karaktersorozat a buffer méretét (ezt előzőleg 256 bájtban adtuk meg) .A for ciklusban megszoroztuk a kulccsal a buffer elemeit, majd maradékos osztást végzünk a % operátorral.A kulcs hosszúságát elérve az érték 0 lesz.

4.4. C EXOR törő

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

Megoldás videó:

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <string.h>
```

```
#include <stdlib.h>
double atlagos_szohossz(const char *titkos, int titkos_meret) {
    int sz = 0;
    for(int i = 0; i < titkos_meret; ++i)
        if(titkos[i] == ' ')
            ++sz;
    return(double) titkos_meret / sz;
}
int tiszta_lehet(const char *titkos, int titkos_meret) {
    // a tiszta szöveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket
    double szohossz = atlagos_szohossz (titkos, titkos_meret);
    return szohossz > 6.0 && szohossz < 9.0
           && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
           && strcasestr (titkos, "az") && strcasestr (titkos, "ha");
}
void exor(const char kulcs[], int kulcs_meret, char titkos[],
        int titkos_meret, char *buffer) {
    int kulcs_index = 0;
    for (int i = 0; i < titkos_meret; ++i) {
        buffer[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}
void exor_tores(const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret) {
    char *buffer;
    if((buffer = (char *)malloc(sizeof(char)*titkos_meret)) == NULL) {
        printf("Memoria (buffer) faliora\n");
        exit(-1);
    }
    exor (kulcs, kulcs_meret, titkos, titkos_meret, buffer);
    if(tiszta_lehet (buffer, titkos_meret)) {
        printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
               kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],
               kulcs[6],kulcs[7], buffer);
    }
    free(buffer);
}
int main(void) {
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;
    // titkos fajt berantasa
    while((olvasott_bajtok =
               read(0, (void *) p,
                     (p - titkos + OLVASAS_BUFFER <
                      MAX_TITKOS) ? OLVASAS_BUFFER :
```

```
                        titkos + MAX_TITKOS - p)))
        p += olvasott_bajtok;
    // maradek hely nullazasa a titkos bufferben
    for(int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\0';
    // osszes kulcs eloallitasa
    #pragma omp parallel for private(kulcs)
    for(int ii = '0'; ii <= '9'; ++ii)
        for(int ji = '0'; ji <= '9'; ++ji)
            for(int ki = '0'; ki <= '9'; ++ki)
                for(int li = '0'; li <= '9'; ++li)
                    for(int mi = '0'; mi <= '9'; ++mi)
                        for(int ni = '0'; ni <= '9'; ++ni)
                            for(int oi = '0'; oi <= '9'; ++oi)
                                for(int pi = '0'; pi <= '9'; ++pi) {
                                    kulcs[0] = ii;
                                    kulcs[1] = ji;
                                    kulcs[2] = ki;
                                    kulcs[3] = li;
                                    kulcs[4] = mi;
                                    kulcs[5] = ni;
                                    kulcs[6] = oi;
                                    kulcs[7] = pi;
                                    exor_tores (kulcs, KULCS_MERET, titkos,
                                    p - titkos);
                                }
    return 0;
}
```

Az Exor törő elején definiáljuk a kulcs méretét és a titkosított szöveg maximális méretét.Megadjuk az átlagos szóhosszot majd a for ciklus végrehajtásával az sz változóhoz mindig hozzáadunk 1-et.Ha a tiszta_lehet függvényben azt kapjuk hogy a kódunk tiszta akkor a következő exor függvényel a tiszta kódot vissza is kapjuk. Az exor_tores függvény a szöveg tisztaságától függően 1-et vagy 0-át ad vissza. A main függvényben 2 tömböt deklarálunk kulcs és titkor néven. A két több mérete a korábban megadott értékeket veszi fel.A while ciklusban olvassuk a bájtokat mindaddig még a bemenet végére nem érünk vagy a Buffer el nem éri a maximális méretét. Ezután for ciklusokkal előállítjuk az összes lehetséges kulcsot alkalmazva a xor műveletet.Találat esetén a kulcs és a feltört szöveg kiíratásra kerül

4.5. Neurális OR, AND és EXOR kapu

R

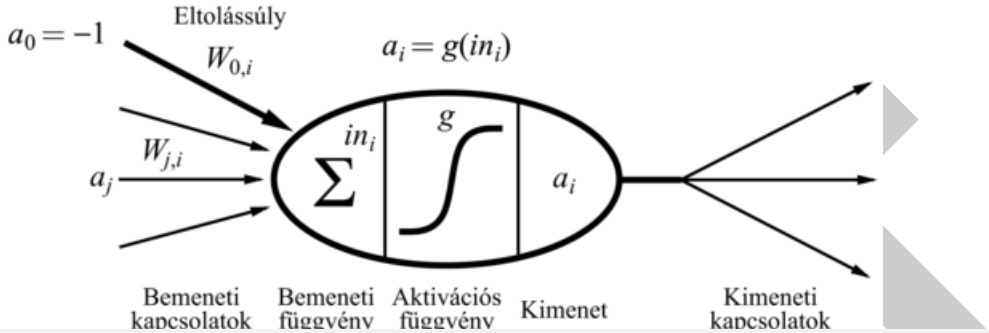
Tutor:Szűcs Gergő

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

A neurális hálóknak manapság egyre nagyobb szerepe van az informatikában ,hiszen a merterséges intelligencia erre épül.A neurális háló képes tanulni és különböző adatokat tulajdonságokat megjegyezni mellyekkel a bejövő adatokat értékeli és reagál rájuk. Az alábbi ábrán a neuron egyszerű matematikai modellje

látható. "Az mondható, hogy a neuron akkor "tüzel", amikor a bemenetek súlyozott összege meghalad egy küszöböt. A kimeneti értéket az aktivációs függvény fogja megadni."

A következő képen a neuron egyszerű matematikai modellje látható. Forrás: link




```
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])
```

R nyelvben dolgozunk, így szükségünk lesz a neuralnet csomagra. Az a1-be és az a2-be értékeket teszünk majd az OR-ban meghatározzuk hogy logikai vagy esetén milyen értéket kell visszakapnunk, azaz itt tanít-

juk a programot ezután már magát fogja. A neuralnet egy súlyozást számol minden bemenethez.Ezután a súlyokat beállítja, melyeknek a helyességét a compute paranccsal ellenőrzünk.

AND használata esetén majdnem ugyanez a feladatom,de ott más lesz a logikai művelet igazságértéke.

EXOR használata esetén rejtett neuronokkal kell dolgoznunk.

4.6. Hiba-visszaterjesztéses perceptron

C++

Tutor:Szűcs Gergő

```
#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;
}
```

Az előbb ,megadott program inputja egy kép,melyben pixelenként vizsgálja a piros szín méretét.AZ adatok neurális hálóba kerülnek. Ezeket az adatokat véletlenszerű számokkal súlyozzuk.Ezeknek az értékeknek a vizsgálatakor kapunk egy számot -1 és 1 között súlyozástól függően.

5. fejezet

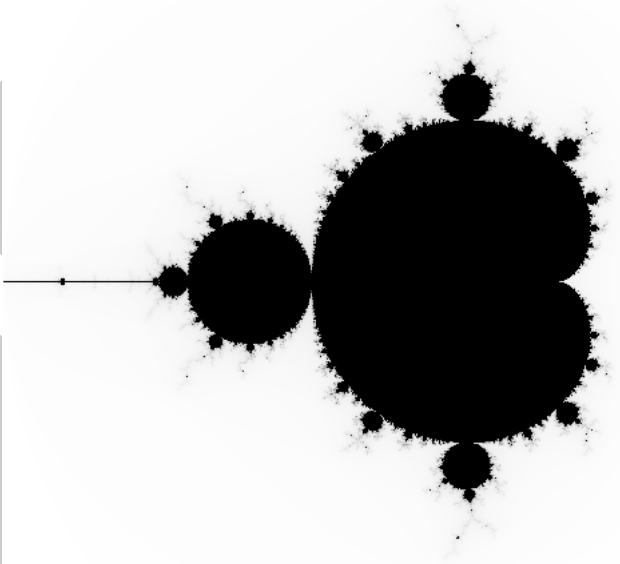
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

A Mandelbrot halmaz a komplex számoknak egy meghatározott részhalmaza. Mandelbrot halmaz ábrázolásánál a képernyő megfelel a síknak a pixelek pedig az oda tartozó komplex számoknak.



```
#include <stdio.h>
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>
#define MERET 600
#define ITER_HAT 32000
void
mandel (int kepadat[MERET][MERET]) {
```

```
    clock_t delta = clock ();
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

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

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

        for (int k = 0; k < szelesseg; ++k)
        {
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;
                ++iteracio;
            }
            kepadat[j][k] = iteracio;
        }
    }
    times (&tmsbuf2);
    std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
              + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;
    delta = clock () - delta;
    std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}
int
main (int argc, char *argv[])
{
    if (argc != 2)
    {
        std::cout << "Hasznalat: ./mandelpng fajlnev";
        return -1;
    }
    int kepadat[MERET][MERET];
```

```
    mandel(kepadat);
    png::image < png::rgb_pixel > kep (MERET, MERET);
    for (int j = 0; j < MERET; ++j)
    {
        for (int k = 0; k < MERET; ++k)
        {
            kep.set_pixel (k, j,
                png::rgb_pixel (255 -
                (255 * kepadat[j][k]) / ITER_HAT,
                 255 -
                (255 * kepadat[j][k]) / ITER_HAT,
                255 -
                (255 * kepadat[j][k]) / ITER_HAT));
        }
    }
    kep.write (argv[1]);
    std::cout << argv[1] << " mentve" << std::endl;
}
}
```

A kódunk elején definiáljuk a kép méretét és az iterációs határt.Egy megadott lépésközzel végig megyünk a rácspontokon, majd a C és a Z komplex számokat változókban tároljuk. Mivel komplex számokkal dolgozunk ezért létrejön egy koordinátarendszer és minden számnak 2 koordinátája lesz (x és y).A while ciklusban szűmoljuk a halmaz következő elemeit mindaddig, amíg a z négyzete kisebb, mint 4. Az iterációs határ elérése után az iteráció konvergens ,Ezért a c a mandelbrot halmaz eleme. Már csak meg kell adnunk a pixelek színét és ki is rajzolhatjuk az ábrát.

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

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

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

```
#include <iostream>
#include "png++/png.hpp"
#include <complex>
int
main ( int argc, char *argv[] )
{
  int szelesseg = 1920;
  int magassag = 1080;
  int iteraciosHatar = 255;
  double a = -1.9;
  double b = 0.7;
  double c = -1.3;
  double d = 1.3;
  if ( argc == 9 )
    {
      szelesseg = atoi ( argv[2] );
```

```
      magassag =  atoi ( argv[3] );
      iteraciosHatar =  atoi ( argv[4] );
      a = atof ( argv[5] );
      b = atof ( argv[6] );
      c = atof ( argv[7] );
      d = atof ( argv[8] );
    }
  else
    {
      std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↩
         " << std::endl;
      return -1;
    }
  png::image < png::rgb_pixel > kep ( szelesseg, magassag );
  double dx = ( b - a ) / szelesseg;
  double dy = ( d - c ) / magassag;
  double reC, imC, reZ, imZ;
  int iteracio = 0;
  std::cout << "Szamitas\n";
  // j megy a sorokon
  for ( int j = 0; j < magassag; ++j )
    {
      // k megy az oszlopokon
      for ( int k = 0; k < szelesseg; ++k )
        {
          // c = (reC, imC) a halo racspontjainak
          // megfelelo komplex szam
          reC = a + k * dx;
          imC = d - j * dy;
          std::complex<double> c ( reC, imC );
          std::complex<double> z_n ( 0, 0 );
          iteracio = 0;
          while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
            {
              z_n = z_n * z_n + c;
              ++iteracio;
            }
          kep.set_pixel ( k, j,
                          png::rgb_pixel ( iteracio%255, (iteracio*iteracio ↩
                             )%255, 0 ) );
        }
      int szazalek = ( double ) j / ( double ) magassag * 100.0;
      std::cout << "\r" << szazalek << "%" << std::flush;
    }
  kep.write ( argv[1] );
  std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

Mint az előző programban , itt is meg kell adnunk a kép adatait, az iterációs határt és az adatokat amelyek a számításhoz szükségesek.Ebben a kódban 2 for ciklussal járjuk be a rácspontokat, majd a kép pixeleit

beállítjuk. Figyeljünk arra hogy ne azokat az értékeket adjuk meg mint az előző feladatban, hiszen így egy színesebb képet kaphatunk.A while ciklus megmutatja hogy mennyire távolodott el egymástól a z_n és a z_0.Ha a while ciklusból való kilépés az iterációs határ elérése miatt történik, akkor az iteráció konvergens ,Ezért a c a mandelbrot halmaz eleme.

5.3. Biomorfok

Első gyakorlat passzolási lehetőség

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

"A CUDA jelentése Compute Unified Device Architecture, vagyis egy új hardver és szoftverarchitektúra amelynek célja, hogy ellássa és kezelje a GPU-n történő számításokat anélkül,hogy azokat le kellene képezni valamelyik grafikus API-ra." Forrás : https://dea.lib.unideb.hu/dea/bitstream/handle/2437/4529/Szakdolgozat.pdf;jsessionid=4DE14EBBC2F036722B7A6CA9355DD3?sequence=1

A CUDA tartalmaz egy rácsot ahol minden sor és oszlop 60 blokkot tartalmaz, 1-1 blokk pedig 100 szálat így egyszerre 60x60x100-at fog számolni. Ha például egy 600x600 pixeles képet szeretnél kiszámolni akkor minden pixelnek megfelel egy szál. A CPU-nál viszont csak egy szálunk volt

```
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>
#include <sys/times.h>
#include <iostream>
#define MERET 600
#define ITER_HAT 32000
__device__ int
mandel (int k, int j)
{
  // Végigzongorázza a CUDA a szélesség x magasság rácsot:
  // most eppen a j. sor k. oszlopaban vagyunk
  // számítás adatai
  float a = -2.0, b = .7, c = -1.35, d = 1.35;
  int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;
  // a számítás
  float dx = (b - a) / szelesseg;
  float dy = (d - c) / magassag;
  float reC, imC, reZ, imZ, ujreZ, ujimZ;
  // Hány iterációt csináltunk?
  int iteracio = 0;
  // c = (reC, imC) a rács csomópontjainak
  // megfelelő komplex szám
  reC = a + k * dx;
  imC = d - j * dy;
```

```
  // z_0 = 0 = (reZ, imZ)
  reZ = 0.0;
  imZ = 0.0;
  iteracio = 0;
  while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
      // z_{n+1} = z_n * z_n + c
      ujreZ = reZ * reZ - imZ * imZ + reC;
      ujimZ = 2 * reZ * imZ + imC;
      reZ = ujreZ;
      imZ = ujimZ;
      ++iteracio;
    }
  return iteracio;
}
/*
__global__ void
mandelkernel (int *kepadat)
{
  int j = blockIdx.x;
  int k = blockIdx.y;
  kepadat[j + k * MERET] = mandel (j, k);
}
*/
__global__ void
mandelkernel (int *kepadat)
{
  int tj = threadIdx.x;
  int tk = threadIdx.y;
  int j = blockIdx.x * 10 + tj;
  int k = blockIdx.y * 10 + tk;
  kepadat[j + k * MERET] = mandel (j, k);
}
void
cudamandel (int kepadat[MERET][MERET])
{
  int *device_kepadat;
  cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));
  // dim3 grid (MERET, MERET);
  // mandelkernel <<< grid, 1 >>> (device_kepadat);

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

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

```
main (int argc, char *argv[])
{
  // Mérünk időt (PP 64)
  clock_t delta = clock ();
  // Mérünk időt (PP 66)
  struct tms tmsbuf1, tmsbuf2;
  times (&tmsbuf1);
  if (argc != 2)
    {
      std::cout << "Hasznalat: ./mandelpngc fajlnev";
      return -1;
    }
  int kepadat[MERET][MERET];
  cudamandel (kepadat);
  png::image < png::rgb_pixel > kep (MERET, MERET);
  for (int j = 0; j < MERET; ++j)
    {
      //sor = j;
      for (int k = 0; k < MERET; ++k)
  {
    kep.set_pixel (k, j,
       png::rgb_pixel (255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT));
  }
    }
  kep.write (argv[1]);
  std::cout << argv[1] << " mentve" << std::endl;
  times (&tmsbuf2);
  std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
    + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;
  delta = clock () - delta;
  std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}
```

A fordítást más paranccsal kell elvégeznünk:

```
nvcc mandelpngc_60x60_100.cu -lpng16 -03 -o mandelpngc
```

Hogy a programot futtatni tudjuk szükségünk lesz nvidia kártyára és akár 50-70x-es gyorsulás is elérhető.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

A következő kód QT GUI-t használ.Ahhoz hogy fordítani tudjunk létr kell hoznunk egy mappát és belehelyezni mind a 4 fájlt amelyre szükségünk lesz. A qmake -project parancs futtatásával létrejönegy *.pro fájl. FOntos hogy a parancsot helyes mappában futtassuk.Írjuk a fájlba a következő sort: QT += widgets.HA ez megtörtént akkor a qmake*.pro parancs futtatásával létrejön egy Makefile , ami után már létrehozható egy bináris fájl a make paranccsal.

```
// frakablak.cpp
#include "frakablak.h"
FrakAblak::FrakAblak(double a, double b, double c, double d,
                     int szelesseg, int iteraciosHatar, QWidget *parent)
                         : QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");
    szamitasFut = true;
    x = y = mx = my = 0;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    magassag = (int)(szelesseg * ((d-c)/(b-a)));
    setFixedSize(QSize(szelesseg, magassag));
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag,  ↩
       iteraciosHatar, this);
    mandelbrot->start();
}
FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}
void FrakAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);
    }
    qpainter.end();
}
void FrakAblak::mousePressEvent(QMouseEvent* event) {
    // A nagyítandó kijelölt területet bal felsõ sarka:
```

```
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;
    update();
}
void FrakAblak::mouseMoveEvent(QMouseEvent* event) {
    // A nagyítandó kijelölt terület szélessége és magassága:
    mx = event->x() - x;
    my = mx; // négyzet alakú
    update();
}
void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {
    if(szamitasFut)
        return;
    szamitasFut = true;
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double a = this->a+x*dx;
    double b = this->a+x*dx+mx*dx;
    double c = this->d-y*dy-my*dy;
    double d = this->d-y*dy;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ↩
       iteraciosHatar, this);
    mandelbrot->start();
    update();
}
void FrakAblak::keyPressEvent(QKeyEvent *event)
{
    if(szamitasFut)
        return;
    if (event->key() == Qt::Key_N)
        iteraciosHatar *= 2;
    szamitasFut = true;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ↩
       iteraciosHatar, this);
    mandelbrot->start();
}
void FrakAblak::vissza(int magassag, int *sor, int meret)
{
    for(int i=0; i<meret; ++i) {
        QRgb szin = qRgb(0, 255-sor[i], 0);
        fraktal->setPixel(i, magassag, szin);
    }
```

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

```
// frakablak.h
#ifndef FRAKABLAK_H
#define FRAKABLAK_H
#include <QMainWindow>
#include <QImage>
#include <QPainter>
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"
class FrakSzal;
class FrakAblak : public QMainWindow
{
    Q_OBJECT
public:
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
              double d = 1.35, int szelesseg = 600,
              int iteraciosHatar = 255, QWidget *parent = 0);
    ~FrakAblak();
    void vissza(int magassag , int * sor, int meret) ;
    void vissza(void) ;
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;
protected:
    void paintEvent(QPaintEvent*);
    void mousePressEvent(QMouseEvent*);
    void mouseMoveEvent(QMouseEvent*);
    void mouseReleaseEvent(QMouseEvent*);
    void keyPressEvent(QKeyEvent*);
private:
    QImage* fraktal;
    FrakSzal* mandelbrot;
    bool szamitasFut;
    // A nagyítandó kijelölt területet bal felsõ sarka.
    int x, y;
    // A nagyítandó kijelölt terület szélessége és magassága.
    int mx, my;
```

```
};
#endif // FRAKABLAK_H
```

```
// frakszal.cpp
#include "frakszal.h"
FrakSzal::FrakSzal(double a, double b, double c, double d,
                   int szelesseg, int magassag, int iteraciosHatar,  ↩
                      FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;
    egySor = new int[szelesseg];
}
FrakSzal::~FrakSzal()
{
    delete[] egySor;
}
void FrakSzal::run()
{
    // A [a,b]x[c,d] tartományon milyen sûrû a
    // megadott szélesség, magasság háló:
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság hálót:
    for(int j=0; j<magassag; ++j) {
        //sor = j;
        for(int k=0; k<szelesseg; ++k) {
            // c = (reC, imC) a háló rácspontjainak
            // megfelelõ komplex szám
            reC = a+k*dx;
            imC = d-j*dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
```

```
            // Mandelbrot halmaz eleme
            while(reZ*reZ + imZ*imZ < 4 && iteracio < iteraciosHatar) {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ*reZ - imZ*imZ + reC;
                ujimZ = 2*reZ*imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;
                ++iteracio;
            }
            // ha a < 4 feltétel nem teljesült és a
            // iteráció < iterációsHatár sérülésével lépett ki, azaz
            // feltesszük a c-rõl, hogy itt a z_{n+1} = z_n * z_n + c
            // sorozat konvergens, azaz iteráció = iterációsHatár
            // ekkor az iteráció %= 256 egyenlõ 255, mert az esetleges
            // nagyítasok során az iteráció = valahány * 256 + 255
            iteracio %= 256;
            //a színezést viszont már majd a FrakAblak osztályban lesz
            egySor[k] = iteracio;
        }
        // Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.
        frakAblak->vissza(j, egySor, szelesseg);
    }
    frakAblak->vissza();
}
```

```
// frakszal.h
#ifndef FRAKSZAL_H
#define FRAKSZAL_H
#include <QThread>
#include <math.h>
#include "frakablak.h"
class FrakAblak;
class FrakSzal : public QThread
{
    Q_OBJECT
public:
    FrakSzal(double a, double b, double c, double d,
             int szelesseg, int magassag, int iteraciosHatar, FrakAblak * ↩
                frakAblak);
    ~FrakSzal();
    void run();
protected:
    // A komplex sík vizsgált tartománya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sík vizsgált tartományára feszített
    // háló szélessége és magassága.
    int szelesseg, magassag;
    // Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
    // (tk. most a nagyítási pontosság)
    int iteraciosHatar;
```

```
    // Kinek számolok?
    FrakAblak* frakAblak;
    // Soronként küldöm is neki vissza a kiszámoltakat.
    int* egySor;
};
#endif // FRAKSZAL_H
```

```
// main.cpp
#include <QApplication>
#include "frakablak.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    FrakAblak w1;
    w1.show();
    /*
    FrakAblak w1,
    w2(-.08292191725019529, -.082921917244591272,
       -.9662079988595939, -.9662079988551173, 600, 3000),
    w3(-.08292191724880625, -.0829219172470933,
       -.9662079988581493, -.9662079988563615, 600, 4000),
    w4(.14388310361318304, .14388310362702217,
       .6523089200729396, .6523089200854384, 600, 38655);
    w1.show();
    w2.show();
    w3.show();
    w4.show();
*/
    return a.exec();
}
```

5.6. Mandelbrot nagyító és utazó Java nyelven

Második előadás passzolási lehetőség.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat... térj ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked!

Az objektumorientált programozásnál elengedhetetlen a classok használata.Egy classban, másnéven osztályban,megjelennhet több függvény és több változó ia melyeket egy egységként kezelhetünk.Osztály megadásánál gyakran szerepel a private vagy a public szavak melyek szerepe meghatározni hogy az adott elemekhez hozzá férhetnek e az osztályon kívüli elemek

Java-ban:

```
public class PolárGenerátor {

    boolean nincsTárolt = true;
    double tárolt;

    public PolárGenerátor() {

        nincsTárolt = true;

    }

    public double következő() {

        if(nincsTárolt) {

            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);

            tárolt = r*v2;
            nincsTárolt = !nincsTárolt;

            return r*v1;

        } else {
            nincsTárolt = !nincsTárolt;
            return tárolt;
        }
    }

    public static void main(String[] args) {

        PolárGenerátor g = new PolárGenerátor();

        for(int i=0; i<10; ++i)
            System.out.println(g.következő());

    }

}
```

C++-ban:

```
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iostream>
class PolarGen
{
public:
  PolarGen ()
  {
    nincsTarolt = true;
    std::srand (std::time (NULL));
  }
```

```
   ~PolarGen ()
  {
  }
double kovetkezo ()
{
  if (nincsTarolt)
    {
      double u1, u2, v1, v2, w;
      do
  {
    u1 = std::rand () / (RAND_MAX + 1.0);
    u2 = std::rand () / (RAND_MAX + 1.0);
    v1 = 2 * u1 - 1;
    v2 = 2 * u2 - 1;
    w = v1 * v1 + v2 * v2;
  }
      while (w > 1);
      double r = std::sqrt ((-2 * std::log (w)) / w);
      tarolt = r * v2;
      nincsTarolt = !nincsTarolt;
      return r * v1;
    }
  else
    {
      nincsTarolt = !nincsTarolt;
      return tarolt;
    }
}
private:
  bool nincsTarolt;
  double tarolt;
};
int main (int argc, char **argv)
{
  PolarGen pg;
  for (int i = 0; i < 10; ++i)
    std::cout << pg.kovetkezo () << std::endl;
  return 0;
}
```

6.2. LZW

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

Az LZW egy ,veszteségmentes tömörítési algoritmus,melyeket bináris adatok tömörítésére használnak. Az algoritmus az adatokat fa struktúrában tárolja. A beérkező adatok(melyek 1-esek és 0-ák) feldolgozása a következőképpen zajlik: a gyökértől kiindulunk majd vizsgáljuk hogy van e már 1-es vag 0-ás gyermek.

Ha van addig lépkedünk amíg nem lesz, ha nincs akkor beszúrunk egyet majd visszalépünk a gyökérre és beolvassuk a következő beérkező adatot és az elejéről kezdjük a folyamatot.

Először létrehozunk egy binfa struktúrát és egy binfa típust.A struktúrában megtalálható egy érték és a fa bal valamint a jobb oldali gyermekeire mutató mutató. A binfa struktúra három részből áll: egy értékből és a fa bal és jobb gyermekeire mutató mutatókból. Létrehozunk egy mutatót, mely a binfa típusra mutat. Az uj_elem() függvény segítségével memóriaterületet foglalunk.A main függvényben létrehozzuk a gyökérmutatót melynek a '/' értéket adjuk. Binfa struktúrában a gyökérelemet általában valamiylen speciális karakterrel jelöljük.Ezután elkezdjük az adatok beolvasását.Ha az érték 0 akkor megvizsgáljuk hogy a fa bal gyermeke null értékű-e. Ha nem akkor az azt jelenti hogy a gyökérelemnek nincs bal oldali gyermeke.Ekkor a fa mutató bal gyerekének lefoglalunk egy helyet és 0-ra állítjuk az értékét.Ha a gyökér bal oldalán már van egy 0 és az inputon érkezik a következő akkor ebben az esetben ez a 0 tölti be a gyökérelem szerepét. Ugyanez a helyzet a jobb oldalon is , Ezután pedig csak kiíratjuk a fát.

```
 #include <stdio.h>
#include <stdlib.h>0
#include <unistd.h>
typedef struct binfa
{
  int ertek;
  struct binfa *bal_nulla;
  struct binfa *jobb_egy;
} BINFA, *BINFA_PTR;
BINFA_PTR
uj_elem ()
{
  BINFA_PTR p;
  if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
      perror ("memoria");
      exit (EXIT_FAILURE);
    }
  return p;
}
extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);
int
main (int argc, char **argv)
{
  char b;
  BINFA_PTR gyoker = uj_elem ();
  gyoker->ertek = '/';
  BINFA_PTR fa = gyoker;
  while (read (0, (void *) &b, 1))
    {
      write (1, &b, 1);
      if (b == '0')
  {
    if (fa->bal_nulla == NULL)
      {
        fa->bal_nulla = uj_elem ();
```

```
        fa->bal_nulla->ertek = 0;
        fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
        fa = gyoker;
      }
    else
      {
        fa = fa->bal_nulla;
      }
  }
      else
  {
    if (fa->jobb_egy == NULL)
      {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
      }
    else
      {
        fa = fa->jobb_egy;
      }
  }
    }
  printf ("\n");
  kiir (gyoker);
  extern int max_melyseg;
  printf ("melyseg=%d", max_melyseg);
  szabadit (gyoker);
}
static int melyseg = 0;
int max_melyseg = 0;
void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
  max_melyseg = melyseg;
      kiir (elem->jobb_egy);
      for (int i = 0; i < melyseg; ++i)
  printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
          ,
        melyseg);
      kiir (elem->bal_nulla);
      --melyseg;
    }
}
```

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

6.3. Fabejárás

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

Egy fa struktúrának három bejárási módja van, inorder, preorder és postorder.

Minden bejárás előtt meg kell vizsgálnunk hogy a bejárandó fa tartalmaz e elemeket vagy egy üres fáról van szó. Preorder bejárásnál először a gyökérelemet dolgozzuk fel,ezután bejárjuk a bal oldali részfáját, és ezután a jobb oldali részfáját is.

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

Postorder bejárásnál pedig a gyökérelem bal oldali részfát járjuk be először, utána a jobb oldali részfát és ezután dolgozzuk fel a gyökérelemet.

```
void
kiir (BINFA_PTR elem)
```

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

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

A feladat az hogy ez előző feladatban C-ben megírt binfát írjuk meg C++-ban is ISmét egy class al kezdünk. Amikor elkezdjük bekérni az elemeket akkor C++-ban a new-val tudunk helyet foglalni a beilleszteni kívánt elemnek. A C++ struktúrában 2 kiir() függvényünk is van. ezt azért tehetjük meg mert a c++ már paraméterlista alapján is meg tud különböztetni 2 azonos nevű függvényt.Használni fogjuk a get függvényt is a mélység, a szórás és az átlag kiszámítására.

Az LZWBinFa class-on kívül definiáljuk az osztályon belüli függvényeket. A class belsejében lévő függvényeket a LZWBinFa:: előtaggal érjük el. A get függvényekkel a private részben lévő elemeket tudjuk elérni.

```
#include <iostream>
#include <cmath>
#include <fstream>
class LZWBinFa {
public:
    LZWBinFa():fa (&gyoker) {
    }
    ~LZWBinFa() {
        szabadit(gyoker.egyesGyermek());
        szabadit(gyoker.nullasGyermek());
    }
    void operator<< (char b) {
        if (b == '0') {
```

```
            if (!fa->nullasGyermek()){
                Csomopont *uj = new Csomopont('0');
                fa->ujNullasGyermek(uj);
                fa = &gyoker;
            }
            else {
                fa = fa->nullasGyermek ();
            }
        }
        else {
            if (!fa->egyesGyermek ()) {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else {
                fa = fa->egyesGyermek ();
            }
        }
    }
    void kiir(void) {
        melyseg = 0;
        kiir(&gyoker, std::cout);
    }
    int getMelyseg(void);
    double getAtlag(void);
    double getSzoras(void);
    friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf) {
        bf.kiir (os);
        return os;
    }
    void kiir(std::ostream & os) {
        melyseg = 0;
        kiir (&gyoker, os);
    }
private:
    class Csomopont {
    public:
        Csomopont(char b = '/'):betu (b), balNulla (0), jobbEgy (0) {
        };
        ~Csomopont() {
        };
        Csomopont *nullasGyermek() const {
            return balNulla;
        }
        Csomopont *egyesGyermek() const {
            return jobbEgy;
        }
        void ujNullasGyermek(Csomopont * gy) {
            balNulla = gy;
```

```
        }
        void ujEgyesGyermek(Csomopont * gy) {
            jobbEgy = gy;
        }
        char getBetu() const {
            return betu;
        }
    private:
        char betu;
        Csomopont *balNulla;
        Csomopont *jobbEgy;
        Csomopont (const Csomopont &);
        Csomopont & operator= (const Csomopont &);
    };
    Csomopont *fa;
    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;
    LZWBinFa (const LZWBinFa &);
    LZWBinFa & operator = (const LZWBinFa &);
    void kiir(Csomopont * elem, std::ostream & os) {
        if (elem != NULL) {
            ++melyseg;
            kiir (elem->egyesGyermek (), os);
            for (int i = 0; i < melyseg; ++i)
                os << "---";
            os << elem->getBetu () << "(" << melyseg - 1 << ")" << std:: ↩
               endl;
            kiir (elem->nullasGyermek (), os);
            --melyseg;
        }
    }
    void szabadit(Csomopont * elem) {
        if (elem != NULL) {
            szabadit(elem->egyesGyermek ());
            szabadit(elem->nullasGyermek ());
            delete elem;
        }
    }
protected:
    Csomopont gyoker;
    int maxMelyseg;
    double atlag, szoras;
    void rmelyseg(Csomopont * elem);
    void ratlag(Csomopont * elem);
    void rszoras(Csomopont * elem);
};
int LZWBinFa::getMelyseg (void) {
    melyseg = maxMelyseg = 0;
    rmelyseg(&gyoker);
    return maxMelyseg - 1;
```

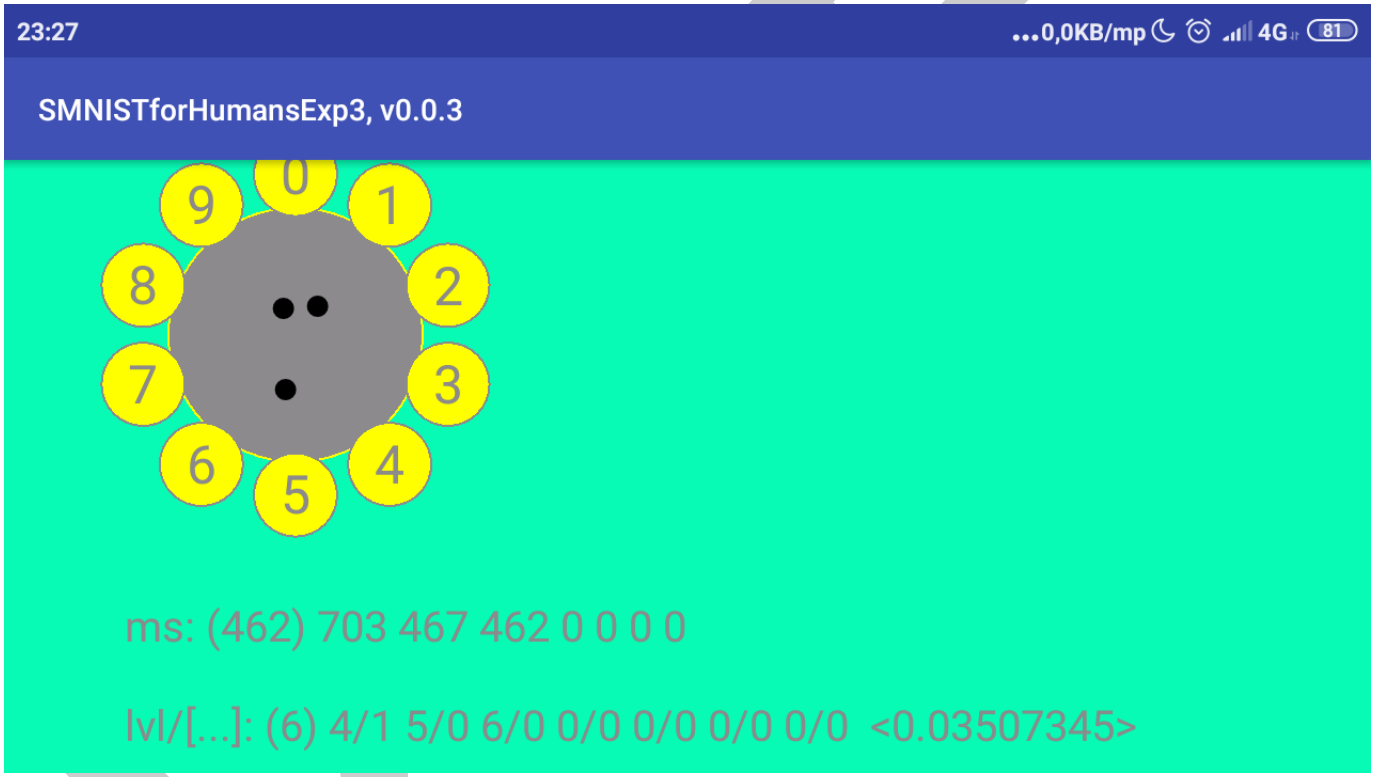
```
}
double LZWBinFa::getAtlag(void) {
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBinFa::getSzoras (void) {
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;
    rszoras (&gyoker);
    if (atlagdb - 1 > 0)
        szoras = std::sqrt (szorasosszeg / (atlagdb - 1));
    else
        szoras = std::sqrt (szorasosszeg);
    return szoras;
}
void LZWBinFa::rmelyseg(Csomopont * elem) {
    if (elem != NULL) {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek());
        rmelyseg (elem->nullasGyermek());
        --melyseg;
    }
}
void LZWBinFa::ratlag(Csomopont * elem) {
    if (elem != NULL) {
        ++melyseg;
        ratlag (elem->egyesGyermek());
        ratlag (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)  ↩
           {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}
void LZWBinFa::rszoras(Csomopont * elem) {
    if (elem != NULL) {
        ++melyseg;
        rszoras (elem->egyesGyermek());
        rszoras (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek () == NULL) ↩
            {
            ++atlagdb;
```

```
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}
void usage(void) {
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}
int main (int argc, char *argv[]) {
    if (argc != 4) {
        usage ();
        return -1;
    }
    char *inFile = *++argv;
    if (*((*++argv) + 1) != 'o') {
        usage ();
        return -2;
    }
    std::fstream beFile (inFile, std::ios_base::in);
    if (!beFile) {
        std::cout << inFile << " nem letezik..." << std::endl;
        usage ();
        return -3;
    }
    std::fstream kiFile (*++argv, std::ios_base::out);
    unsigned char b;
    LZWBinFa binFa;
    while (beFile.read ((char *) &b, sizeof (unsigned char)))
        if (b == 0x0a)
            break;
    bool kommentben = false;
    while (beFile.read ((char *) &b, sizeof (unsigned char))) {
        if (b == 0x3e) {
            kommentben = true;
            continue;
        }
        if (b == 0x0a) {
            kommentben = false;
            continue;
        }
        if (kommentben)
            continue;
        if (b == 0x4e)
            continue;
        for (int i = 0; i < 8; ++i) {
            if (b & 0x80)
                binFa << '1';
            else
                binFa << '0';
            b <<= 1;
        }
```

```
    }
    kiFile << binFa;
    kiFile << "depth = " << binFa.getMelyseg() << std::endl;
    kiFile << "mean = " << binFa.getAtlag() << std::endl;
    kiFile << "var = " << binFa.getSzoras() << std::endl;
    kiFile.close();
    beFile.close();
    return 0;
}
```

6.5. Mutató a gyökér

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



6.6. Mozgató szemantika

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

Második labor passzolási lehetőség

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

3. labor passzolási lehetőség

7.2. Java életjáték

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

A John Horton Conway féle életjáték szabályai a következők : A sejtek cellákban élnek(crllánként 1 sejt).Egy sejt akkor él ha van 2 vagy 3 szomszédja , különben meghal. ha meghalt egy sejt addig halott is marad amig pont 3 szomszédja van.

```
public void időFejlődés() {

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

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

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

                if(rácsElőtte[i][j] == ÉLŐ) {
                /* Élő élő marad, ha kettő vagy három élő
                 szomszedja van, különben halott lesz. */
                    if(élők==2 || élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
```

```
                } else {
                /* Halott halott marad, ha három élő
                 szomszedja van, különben élő lesz. */
                    if(élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
                }
            }
        }
        rácsIndex = (rácsIndex+1)%2;
    }
}
```

A sejtek összessége a sikló. Adott irányba halad,és lemásolja önmagát.

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

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

    }
```

A sikló ágyú bizonyos iránybalövöldözi a többi siklót. Az x és y érték a bal felso sarkának határoló élei.

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

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

        rács[y+ 3][x+ 13] = ÉLŐ;

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

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

        rács[y+ 6][x+ 11] = ÉLŐ;
        rács[y+ 6][x+ 15] = ÉLŐ;
        rács[y+ 6][x+ 16] = ÉLŐ;
        rács[y+ 6][x+ 22] = ÉLŐ;
        rács[y+ 6][x+ 23] = ÉLŐ;
```

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

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

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

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

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

        rács[y+ 11][x+ 25] = ÉLŐ;

    }
```

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

    }
```

Fordításkor,java fordítót használunk.

Ha megnyomjuk az S-t akkor készül egy felvétel a sejttérről.N-el mő, K-val pedig csökken a sejtméret.A G gyorsít az i pedig lassít.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

A szabály ugyanúgy működik mint javaban:

```
void SejtSzal::idoFejlodes() {
    bool **racsElotte = racsok[racsIndex];
    bool **racsUtana = racsok[(racsIndex+1)%2];
    for(int i=0; i<magassag; ++i) { // sorok
        for(int j=0; j<szelesseg; ++j) { // oszlopok
            int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);
            if(racsElotte[i][j] == SejtAblak::ELO) {

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

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

Sikló:

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

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

}
```

Siklókilövő

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

  racs[y+ 6][x+ 0] = ELO;
  racs[y+ 6][x+ 1] = ELO;
  racs[y+ 7][x+ 0] = ELO;
  racs[y+ 7][x+ 1] = ELO;
```

```
  racs[y+ 3][x+ 13] = ELO;

  racs[y+ 4][x+ 12] = ELO;
  racs[y+ 4][x+ 14] = ELO;

  racs[y+ 5][x+ 11] = ELO;
  racs[y+ 5][x+ 15] = ELO;
  racs[y+ 5][x+ 16] = ELO;
  racs[y+ 5][x+ 25] = ELO;

  racs[y+ 6][x+ 11] = ELO;
  racs[y+ 6][x+ 15] = ELO;
  racs[y+ 6][x+ 16] = ELO;
  racs[y+ 6][x+ 22] = ELO;
  racs[y+ 6][x+ 23] = ELO;
  racs[y+ 6][x+ 24] = ELO;
  racs[y+ 6][x+ 25] = ELO;

  racs[y+ 7][x+ 11] = ELO;
  racs[y+ 7][x+ 15] = ELO;
  racs[y+ 7][x+ 16] = ELO;
  racs[y+ 7][x+ 21] = ELO;
  racs[y+ 7][x+ 22] = ELO;
  racs[y+ 7][x+ 23] = ELO;
  racs[y+ 7][x+ 24] = ELO;

  racs[y+ 8][x+ 12] = ELO;
  racs[y+ 8][x+ 14] = ELO;
  racs[y+ 8][x+ 21] = ELO;
  racs[y+ 8][x+ 24] = ELO;
  racs[y+ 8][x+ 34] = ELO;
  racs[y+ 8][x+ 35] = ELO;

  racs[y+ 9][x+ 13] = ELO;
  racs[y+ 9][x+ 21] = ELO;
  racs[y+ 9][x+ 22] = ELO;
  racs[y+ 9][x+ 23] = ELO;
  racs[y+ 9][x+ 24] = ELO;
  racs[y+ 9][x+ 34] = ELO;
  racs[y+ 9][x+ 35] = ELO;

  racs[y+ 10][x+ 22] = ELO;
  racs[y+ 10][x+ 23] = ELO;
  racs[y+ 10][x+ 24] = ELO;
  racs[y+ 10][x+ 25] = ELO;

  racs[y+ 11][x+ 25] = ELO;

}
```

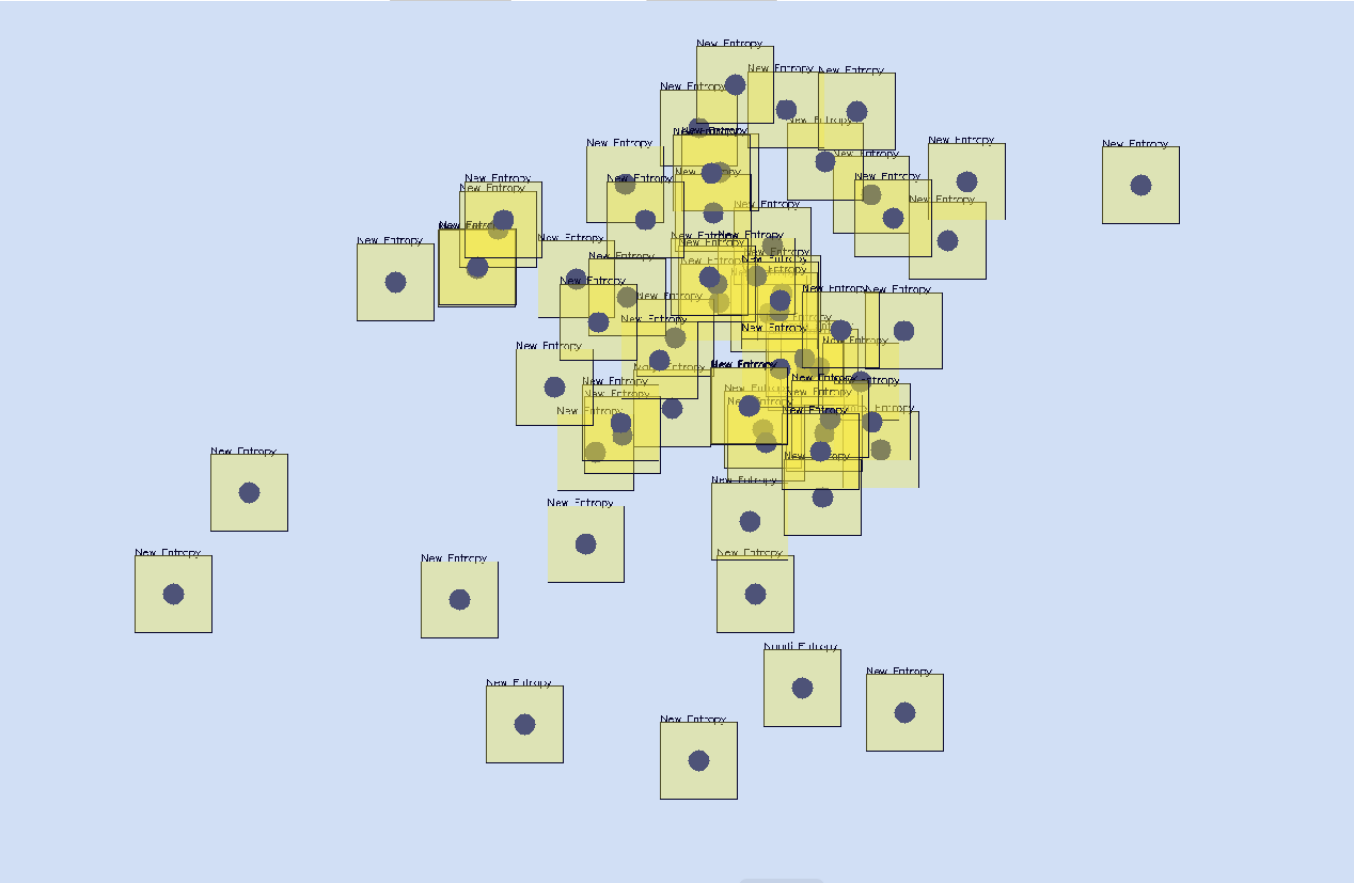
```
#include <QApplication>
#include "sejtablak.h"
#include <QDesktopWidget>
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  SejtAblak w(100, 75);
  w.show();

  return a.exec();
}
```

C++-nál használmi kell a QT-t, ezért fordításkor először Qmake -project , ez majd legenerálja a -pro fájlt és ezután még alkalmaznom kell egy make-et. A futtatás menete ugyanaz mit Javaban

7.4. BrainB Benchmark

A program azért íródott hogy a karakterek mozgása követését figyeljék meg e sportolókon. A Samu Entropy dobozon belül, a mozgó körben kell tartani az egérmutatót és eközben újabb dobozok is megjelennek hogy összezavarjanak. Ha 1 másodpercet meghaladja az az idő amikor az egérmutató a körön kívül van akkor romlik az eredmény és a dooboz mozgása lassul.

Fordításkor az OpenCv könyvtárat használjuk.



A mérés 10 percig tart és statisztika is készül a teljesítményről, amit a program könyvtárában egy txt fájlban találunk.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Minecraft-MALMÖ

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

Juhász István - Magas szintű programozási nyelvek 1

Az informatikában léteznek különböző dekleratív és imperatív nyelvek. Az imperatív nyelvekben a legfontosabb a változó melyeket különböző algoritmusokban használunk fel.Nagyon meghatározó szempontjukból a Neumann-architektúra. A deklaratív nyelveknek nincs igazán meghatározott közös jellemzőjük,de nem algoritmikusak.

Karakterkészlet

A programok mindig karakterekből állnak ezért érdemes ha ismerjük a karakterkészletünket, azaz hogy milyen karakterekkel dolgozhatunk.A karakterek 3 fő csoportja a betűk a számok és az egyéb karakterek.

Adattípusok

Minden adattípusnak kell rendelkezni névvel és azonosítóval, hogy el lehessen róla mondani hogy egy adattípus.VAnnak programnyelvek ,amelyek ismerik a típusokat és vannak amelyek nem ,így különböztetünk meg típusos és nem típusos nyelveket. Vannak olyan nyelvek melyek engedélyezik a saját adattípus megadását, ekkor meg kell adnunk a tartományát a műveleteit és a reprezentációját.

Kifejezések

A kifejezések olyan szintaktikai eszközök amelyek operandusokból operátorokból és kerek zárójelekből állnak. Az operandus lehet literál, konstans, változó vagy függvényhívás. Az operátorok műveleti jelek. A kerek zárójelek a műveletek végrehajtási sorrendjét befojásolják, rendundásan alkalmazhatóak.

Utasítások

Az utasítások segítségével meg tudjuk adni az alggoritmusk egyes részeit és még a tárgyprogramok is ezek segítségével készülnek. Rengeteg utasítás létezik melyeket 9 nagyobb csoportra osztunk (értékadó utasítás,üres, ugró, elágazó, ciklusszervező, hívó, vezérlésátadó,I/O utasítások és más utasítások)

A programok szerkezete

Egy program szövege programegységekből áll. Van olyan nyelv hogy egy programot programegységenként kell fordítanunk de vn olyan is amelyben magá a kész programot egyben kell fordítanunk Vannak különböző alprogramuk is .A leghasznosabbnak és a leggyakrabban alkalmazottnak akkor bizonyulnak amikor egy egy hosszabb programban egy adott programrész többször ismétlődik. Ha alprogramban írjuk ezt a részt elég

egyszer megírnunk és nem kell többször a kódba gépelni. Az alprogramnak két fajtája van: eljárás és függvény. Az eljárássaleredményt kapunk amelyet fel tudunk használni,de a függvény csak egy értéket ad vissza.

A folyamatot mikor az alprogram hívásánál egymáshoz rendelődnek a formális és aktuális paraméterek paraméterkiértékelésnek nevezzük. A blokk olyan programegység amely egy másik programegységbe van beágyazva. A blokknak nincs paramétere és bárhol elhelyezhető.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Vezérlési szerkezetek

Egy kifejezés akkor válik utasítássá ha pontosvessző követi.A Kerighanritchie könyv vezérlési szerkezetek fejezete föként utasítás elemzéssel és értelmezéssel foglalkozik. Az if-else utasítást valamilyen választás vagy döntés leírására használhatjuk.Van amikor ez az utasítás az else ág nélkül is helyesen működik,de erre oda kell figyelnünk mert hiányzó else ág esetén nem tudni hogy a meglévő else ág melyik ifhez tartozik.Ennek párja az else-if, mellyel többszörös választást, döntédt írhatunk le. A switch utasítás is a többirányú programelágaztatás egyik eszköze, mely egy kifejezés értékét hasonlitja több állandó értékű kifejezéshez.Ha ki szeretnénk lépni a switch utasításból csak alkalmaznunk kell a break utasítást. A while és for utasításokat ciklusokban használjuk. A while működési elve hogy egy kifejezést ciklikusan addig értékel ki amíg az nullává nem válik.A for utasítás is hasonlóan működik,de itt nem 1 hanem 3 kifejezésünk van. Az első és a harmadik általában függvényhívás és a második egy relációs kifejezés. Ha a kifejezéseket elhagyjuk és csak a pontos vesszőket írjuk ki a for ciklusban egy végtelen ciklust hozunk létre. A do-while utasításban a kifejezés kiértékelése a ciklustörzs végrehajtása után történik,tehát egyszer biztod végrehatódik a törzs. A break utasításról már volt szó, a legfontosabb jellemzője ,hogy lehetővé teszi hogy egy ciklust a befejezése előtt elhagyjunk. A goto utasítás egy megadott címre ugorhatunk.Leggyakoribb használata az anikor egymásba épülő ciklusok vannak a kódban és valamelyik ciklust ki szeretném hagyni de nem akarom törölni.

10.3. Programozás

[BMECPP]

A C++ nem objektumorientált tulajdonságai

A C nyelvet továbbfejlesztették annak érdekében, hogy kényelmesebb és biztonságosabb legyen a használata és így jött létre a C++ programozási nyelv.Az első fontos különbség a paraméterek értelmezése C++ban a void függvény üres paraméterlistát jelent még a C nyelvben ez azt jelentené hogy a függvény nem rendelkezik paraméterrel.A C++ -ban bevezették a bool típust ami nagyon hasznos tud lenni kiértékelések esetén. A C-ben ez a típus nem létezik. C nyelvben egy függvényt csaupán a neve azonosítja mé C++ ban a neve és az argumentumlistája, ezért C++-ban már létrehozható több ugyanolyan nevű függvény is. Ez a funkció hosszabb kódoknál nagyon hasznos lehet.C++ban a A címszerinti paraméterátadás abban változott hogy a paraméter deklarációjában csak egy jelet kell írnunk a paraméter elé.

III. rész

Második felvonás

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

11. fejezet

Helló, Berners-Lee!

11.1. 0. hét: Az objektumorientált paradigma alapfoglamai. Osztály,objektum, példányosítás.

A Python egy objektumorientált,dinamikus, platformfüggetlen programozási nyelv amelyet még a 90-es években alkottak meg. Egy olyan szkriptnyelv, amelyet a kiegészítő csomagok és eljárások lehetővé tesznek bonyolultabb programok megírásához. Mivel egyszerű ,de mégis megbízható ezért ezt használva ajánlott prototípus-alkalmazások és különböző tesztelések megírása. Ezek mellett a Pythonra magas szintű programozási nyelvként tekintünk. A Python nyelv legfőbb jellemzője hogy behúzásalapú a szintaxisa, ninca szükség kapcsos zárójelekre vagy explicit kulcsszavakra.Fontos hogy a behúzásokat egységesen kezeljük,tehát vagy mindenhol tabot vagy mindenhol szóközt használjunk. Továbbá fontos megjegyezni hogy egy utasítás mindig a sor végéig tart,nincs ';' a sorok végén. Ha egy utasítás nem fér el egy sorban akkor '\'-jelet teszünk a sor végére. Az értelemző minden sort tokenekre bont melyek lehetnek : azonosító, kulcsszó, operátor,delimiter és literál. Pythonban minden adatot objektumok reprezentálnak.Különböző adattípusok vannak, melyek a következők: számok (egészek, lebegőpontoska,komplexek), sztringek (Ezeket idézőjelek és aposztrófok közt is megadhatjuk vagy az "u" betű használatával Unicode szöveget is felvehetünk), ennesek (zárójellel közé zárt objektumok gyűjteménye vesszővel elválasztva, listák(elemek rendezett szekvenciája), szótár(kulcsokkal azonosított elemek rendezetlen halmaza) Pythonban a változók az objektumokra mutató referenciákat jelentik. A változónak értéket a "=" -lel adunk. Pythonban függvényeket a del szóval definiálunk. A függvények rendelkeznek paraméterekkel melyek érték szerint adhatók át kivéve a mutable típusok. A python nyelvben is megjelenik a kivételkezelés. Ekkor a try kulcsszó után található a kódblokk melyben a kivételes eset előállhat. Az elméleti bemutatás után a könyvben különböző példákat találhatunk.

11.2. C++ és Java összehasonlítása

A feladat a C++ és a Java nyelv összehasonlítása volt. A 2 nyelvben nagyon sok a hasonló szintaxis. Ez annak köszönhető hogy a C és a C++ nyelvet vették alapul a Java tervezői. A sok szintaktika nem jelenti mindig azt hogy a szintaktikák jelentése is megegyezik. A Java objektumorientált programozási nyelv , míg a C++ multiparadigmás nyelv. Nagyon nagy és fontos különbség az ,hogy a Java nyelv platformfüggetlen, azaz először létrehozza a Java Virtual Machine számára az érthető kódot, amelyet minden olyan

rendszer értelmezni tud majd amelyre a JVM meg van írva. Ezzel szemben a C++- nál szükségünk van fordítóra hogy a manuálisan megírt kódot gépi kódra fordítsa. Az is elég fontos különbség, hogy Java-nál van automatikus garbage collector , c++ -nál a programozónak kell megírnia a destruktorokat/felszabadítani a memóriát. Játékfejlesztésnél ez elég fontos, Java-ban nem igazán írnak nagyobb teljesítményigénnyel rendelkező játékot mert nagyon sok memóriát használ, így kisebb eszközöknél, amiknél limitált a memória jobb a c++. Java-ban a típusokat két fo csoportra oszthatjuk, ezek a primitív és a nem primitív típusok. A primitív típusok magát az értéket tárolják. Ezzel szemben a nem primitív típusok objektumhivatkozások. A Java nyelv lehetové teszi a primitív típusok "objektumosítását". Tehát az int típust helyettesíthetjük Integer típussal. Jelen esetben az Integer egy csomagoló osztály. A Java 5-ös verziójától lehetové vált az ezen típusok közötti automatikus be- és kicsomagolás. Nem létezik a char* Java-ban, ehelyett egy beépített az-osztályt használhatunk, a String-et. A C++-al ellentétben a tömbök nem a pointerek másik megnyilvánulási formái, hanem ezek is valódi típusok. Ennek köszönhetoen elérhet ˝ o a méretüket lekérdező függvény is, így nem szükséges azt tárolni egy másik változóban.

12. fejezet

Helló, Arroway!

12.1. 1.hét

Az objektumorientált paradigma: Az OOP, azaz objektum-orientált programozás, olyan zárt programegységek (objektumok) halmazából építi fel a program egészét, amelyek egyedi tulajdonságokkal rendelkeznek és önmagukban is működőképesek. Legfontosabb alapelvek: egységbezárás, adatrejtés, öröklés, absztrakció, polimorfizmus. Egységbe zárás során az osztályok állapotai és viselkedései egy helyen, adatstruktúraként egy egységben jelenik meg. Az adatrejtés során a jellemzőket és a metódusokat el tudjuk rejteni más objektumok elől, pl. egy private vagy protected állapotot létrehozva. Az öröklés során a szülőosztály jellemzői és viselkedése öröklődik az alosztályra és egy új speciális változat jön létre.

C++: Osztály: Az osztály egy egységbe záró adatstruktúra, típus leírás. Class kulcsszóval ez új osztályt hozhatunk létre. Definiálja az objektumok tulajdonságait, attribútumait, mezőit, természetét, a viselkedéseit, metódusokat, műveleteket és funkciókat tartalmaz. Az osztály tagjait háromféleképpen érhetjük el. Ezek az elérhetőségi szintek a public, private és protected. A public esetén mindenki számára nyilvános az osztály tartalma, mindenki hozzáférhet. A private esetén, kizárólag osztályon belül lehet elérni, ugyanakkor barát (friend) osztályokból és függvényekből lehet elérni. A protected esetén az adattagok elérhetősége védett, közvetlen elérés csak a származtatott osztályok számára lehetséges, a private tagok elérése kizárólag az ősosztály metódusaiból lehetséges.

Objektum (példány): Az objektumok az osztályok példányai, amelyek adatokat tárolnak és utasításra műveleteket hajtanak végre, állapottal rendelkeznek. A példány alatt a futásidőben létrejövő aktuális objektumot értjük. Primitíven megfogalmazva az osztály és objektum kapcsolatát, mondhatnánk, hogy az osztály a típus, az objektum pedig a változó.

Példányosítás: A példányosítás az a folyamat, amely során a memóriában lefoglalt helyre létrejön az osztály egy példánya (objektuma). Kétféleképpen hozhatunk létre objektumot: dinamikusan és statikusan. Pl.:

```
class PeldaOsztaly {...};
PeldaOsztaly statikus_objektum;
PeldaOsztaly * dinamikus_objektum = new PeldaOsztaly();
```

A két objektum elérésben különbözik. Ahhoz, hogy elérjük a tagokat, a statikusan létrehozott objektum esetén, használnunk kell a „ . ” operátort. A dinamikus objektum esetén viszont a „->” operátort. Elérésük:

```
statikus_objektum.valamilyen_tagf();
dinamikus_objektum->valamilyen_tagf();
```

Java: Osztály: Az osztály az egy sablon/tervrajz amiből egyedi objektumok hozhatók létre. A class kulcsszóval egy új osztályt hozhatunk létre. Pl.:

```
 class Auto
    {
public String marka;
public int kor;
}
```

Az adattagokat és metódusokat tetszőleges sorrendben sorolhatjuk fel. Minden egyes tagnak a láthatóságát külön lehet szabályozni. Most minden elemet public szintű láthatóságra állítottunk. Protected esetén csak a leszármazottak számára, private esetén senki számára se érhetőek el a tagok. Ha nem határozzuk meg a láthatóság szintjét, akkor a tag csak a forrásszöveg környezetében lesz látható, máshol nem. Létrehoztak egy új osztályt, ez a String típus, amelynek léte már nem csak a karakterláncból álló tömböt teszi ki. Mivel a metódusok folyamatos ellenőrzés alatt állnak, így nem léphet fel szöveg túllépéséből eredendő hiba.

Példányosítás: Egy osztályból új objektumokat a new kulcsszóval kreálhatunk.

```
Auto kedvenc = new Auto();
```

A new operátor feladata, hogy lefoglalja az objektumnak szükséges memória területet és inicializálja azt. A new operátor után az osztály előírja a konstruktor hívását. A konstruktor neve megadja. hogy melyik osztályból jöjjön létre a példány és a konstruktor incializálja az új obketumot. Ekkor létrejön az Auto class objektuma, és a kedvenc bár az objektumnév, de nem az objektum maga, hanem a referencia, azaz a memóriacím, ahol az Auto class új objektuma létrejött. Az újonnan létrehozott objektumnak a tagjai 0 vagy null kezdeti értéket kapnak. Hogy ezeket feltöltsük új new operátorokat kéne használni vagy meglévő objektumok átadását alkalmazni.

Objektum: Az osztály sablonja alapján felépített és létrehozott példány. Ha az objektum valamely elemére szeretnénk hivatkozni, akkor az „objektum.elemneve” alakú névvel lehet hivatkozni. Ha az elem is több részből áll, akkor hasonló módon hivatkozunk azokra is. Az egyszerű típusú változókhoz nem kell további memóriafelületet foglalni, így nincs szükség a new operátorra.

```
class Auto
    {
    public String marka;
    public int kor;
    public static int   evjarat = 1990;
}
```

Azok az elemek, amelyek a static kulcsszóval lettek deklarálva, nem az objektumokhoz, hanem magához az osztályhoz fognak tartozni. Így tehát, ha használjuk a new operátort, akkor nem lesz memóriaterület foglalva nekik az objektumokban. Mivel ez az adattag azelőtt is létezett, hogy az objektum adattagjait inicializálnánk, így nem kell nekik értéket adni. Osztalynev.adattag formában lehet hivatkozni ezekre az elemekre. Ezeket az adattagoknak az osztály létrejöttével egyidőben foglalódik memória és lehet iniciálni azt. Objektumot nem tudunk explicit módon megszüntetni. Ha nem hivatkozunk objektumra, megszűnik: kedvenc=nulll; A null referenciát bármely típusú változónak értékül adhatjuk, ezzel megszűntetjük a hivatkozást az objektumra. A futtató rendszer tudja, hogy hány hivatkozás van egy objektumra, így ha senki sem hivatkozik arra, akkor megszüntethető. A java.lang.System.gc() eljárás ráveheti a rendszert a takarításra.

Python: Osztály: Attribútumai között megjelenik az objektum és függvény. A függvényt mé metódusoknak vagy tagfüggvénynek is emlegetjük. Osztályok örökölhetnek egymástól.

Osztály definiálás:

```
class osztalynev ( ososztalyok) :
    osztalytorzs
```

Az ososztályok már korábban létrejött osztályok listái. Az osztályhoz és példányokhoz is adhatunk attribútumokat. Ha megváltoztatjuk egy osztály valamely attribútumát, akkor az összes olyan példány attribútumára hatással lesz, ahol példányszinten nem változtattuk meg az attribútumot. Az osztály törzséből vagy kívülről is lehet attribútumoka meghatározni. Az osztályok metódusainak első kötelező paramétere a self kifejezés, amely étékül azt az objektumpéldányt kapja, ahol a függvény meg lett hívva.

```
    class Viszlat:
        def ModdViszlat(self, ember):
            print 'Szep ', napot, '
```

A speciális függvények és változók nevei két-két alulvonás jel közé vannak zárva, mint pl. a konstruktor tulajdonsággal rendelkező _ _init_ _. Ugyanúgy az első paraméter a self, de több paramétert is kaphat a konstruktor.

```
class Kocka(object)
    def _ _init_ _(self, szelesseg, magassag,, melyseg):
        self.szelesseg = szelesseg
        sel.magassag = magassag
        self.melyseg = melyseg
```

Objektum: Osztály példányai.

12.2. Homokózó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutassunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani és minden máris működik (erre utal

a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen).

Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját! 1

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

12.3. „Gagyi”

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!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

12.4. Yoda

Írjunk olyan Java programot, ami java.lang.NullPointerEx-el leáll, ha nem követjük a Yoda conditions-t! https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp- alg.pdf és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását!

Ha megakadsz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok- javat/apbs02.htm (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés.

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai- Barki/madarak/)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.2. Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetőek!

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.3. Anti OO

A BBP algoritmussal 4 a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10 6, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! https://www.tankonyvtar.hu/hu/tarta tanitok-javat/apas03.html#id561066

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.4. deprecated - Hello, Android! (zöld)

Élesszük fel a https://github.com/nbatfai/SamuEntropy/tree/master/cs projektjeit és vessünk össze néhány egymásra következőt, hogy hogyan változtak a források!

13.5. Hello,Android!

Élesszük fel az SMNIST for Humans projektet! https://gitlab.com/nbatfai/smnist/tree/master/forHumans/-SMNISTforHumansExp3/app/src/main Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.6. Hello, SMNIST for Humans! (piros)

Fejleszd tovább az SMNIST for Humans projektet SMNIST for Anyone emberre szánt appá! Lásd az smnist2_kutatasi_jegyzokonyv.pdf-ben a részletesebb hátteret!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.7. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf (77-79 fóliát)!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14. fejezet

Helló, Mandelbrot 2.0!

14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nlERlEOs.

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_6.pdf (28-32 fólia)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.2. Forward engineering UML osztálydiagram

UML-ben tervezzünk osztályokat és generáljunk belőle forrást!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.3. Egy esettan

A BME-s C++ tankönyv 14. fejezetét (427-444 elmélet, 445-469 az esettan) dolgozzuk fel!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.4. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog
(34-47 fólia)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.5. BPEL Helló, Világ! - egy visszhang folyamat (zöld)

Egy visszhang folyamat megvalósítása az alábbi teljes „videó tutoriál” alapján: https://youtu.be/0OnlYWX2v_I

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.6. TeX UML

Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15. fejezet

Helló, Chomsky!

15.1. Encoding

Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket! https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.2. OOCWC lexer

Izzítsuk be az OOCWC-t és vázoljuk a https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src lexert és kapcsolását a programunk OO struktúrájába!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.3. l334d1c4[5] (zöld)

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: https://simple.wikipedia.org/wiki/Leet (Ha ez első részben nem tetted meg, akkor írasd ki és magyarázd meg a használt struktúratömb memóriafoglalását!)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.4. Full screen (zöld)

Készítsünk egy teljes képernyős Java programot! Tipp: https://www.tankonyvtar.hu/en/tartalom/tkt/javat-tanitok-javat/ch03.html#labirintus_jatek

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.5. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.6. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.7. Perceptron osztály

Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! Lásd https://youtu.be/XpBnR31BRJ

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16. fejezet

Helló, Stroustrup!

16.1. JDK osztályok

Írjunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.2. Másoló-mozgató szemantika

Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékadásra!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.3. Hibásan implementált RSA törése

Készítsünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: https://arato.inf.unideb.hu/batfai.n
(71-73 fólia) által készített titkos szövegen.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.4. Változó argumentumszámú ctor

Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 hét/Perceptron osztály feladatot is.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.5. Összefoglaló

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutató „”esszé szöveget!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17. fejezet

Helló, Gödel!

17.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világbajnokságban https://youtu.be/DL6iQwPx1Yw (8:05-től)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.2. C++11 Custom Allocator

https://prezi.com/jvvbytkwgsxj/high-level-programming-languages-2-c11-allocators/ a CustomAlloc-os példa, lásd C forrást az UDPROG repóban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.3. STL map érték szerinti rendezése

Például: https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.4. Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang Interface Comparable<T> szerepét!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.5. Prolog családfa

Ágyazd be a Prolog családfa programot C++ vagy Java programba! Lásd para_prog_guide.pdf!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.6. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témát (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18. fejezet

Helló, !

18.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! https://github.com/nbatfai/future/tree/master/cs/F6
Itt láthatjuk működésben az alapot: https://www.twitch.tv/videos/222879467

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! https://github.com/nbatfai/robocar- emulator/blob/master/-justine/rcemu/src/carlexer.ll

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.3. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: https://github.com/nbatfai/SamuC

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.4. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: https://github.com/nbatfai/esport- talent-search

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.5. OSM térképre rajzolása

Debrecen térképre dobjunk rá cuccokat, ennek mintájára, ahol én az országba helyeztem el a DEAC hekkereket: https://www.twitch.tv/videos/182262537 (de az OOCWC Java Swinges megjelenítőjéből: https://github.com emulator/tree/master/justine/rcwin is kiindulhatsz, mondjuk az komplexebb, mert ott időfejlődés is van...)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19. fejezet

Helló, Schwarzenegger!

19.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/ch01.html#id527287

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.2. AOP

Szőj bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.3. Android Játék (zöld)

Írjunk egy egyszerű Androidos „játékot”! Építkezzünk például a 2. hét „Helló, Android!” feladatára!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.4. Junit teszt

A https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat poszt kézzel számított mélységét és szórását dolgozd be egy Junit tesztbe (sztenderd védési feladat volt korábban).

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.5. OSCI

Készíts egyszerű C++/OpenGL-es megjelenítőt, amiben egy kocsit irányítasz az úton. A kocsi állapotát minden pillanatban mentsd le. Ezeket add át egy Prolog programnak, ami egyszerű reflex ágensként adjon vezérlést a kocsinak, hasonlítsd össze a kézi és a Prolog-os vezérlést. Módosítsd úgy a programodat, hogy ne csak kézzel lehessen vezérelni a kocsit, hanem a Prolog reflex ágens vezérelje!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20. fejezet

Helló, Calvin!

20.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13/hello_s
bol Háttérként ezt vetítsük le: https://prezi.com/0u8ncvvoabcr/no-programming-programming/

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.2. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a https://arato.inf.unideb.hu/batfai.norb
8. fóliáját!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.3. CIFAR-10 (zöld)

Az alap feladat megoldása, +saját fotót is ismerjen fel, https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-
10_tf_tutorial_peldabol

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.4. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.5. SMNIST for Machines

Készíts saját modellt, vagy használj meglévőt, lásd: https://arxiv.org/abs/1906.12213 (kék)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.6. Minecraft MALMO-s példa

A https://github.com/Microsoft/malmo felhasználásával egy ágens példa, lásd pl.: https://youtu.be/bAPSu3Rndi8, https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innentol_mi, https://bhaxor.blog.hu/2018/10/28/minecraft_ (kék)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

20.7. Általános

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

20.8. C

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

20.9. C++

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

20.10. 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.