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 Ã©s Hetes, Bence	2019. április 1.	

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

Ajánlás

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

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

Tartalomjegyzék

Ábrák jegyzéke

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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!

Megoldás forrása: https://github.com/RakeTheRape/BHAKSZ/blob/master/turing/

Tanulságok, tapasztalatok, magyarázat...

A program ciklusokat használva dolgoztatja meg a procit. Sokszor véletlenül készítünk végtelen ciklusokat. Az alábbi példa azt az esetet mutatja be ahol csak 1 magot használ 100%-on.

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

A következő példa már minden magot használ 100%-on.

```
#include <omp.h>

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

}
```

Ennek a programnak a futtatásához szükséges a "-fopenmp" használata.

Utolsó példánk mikor a program 0%-ban használja ki a procit. Ezt sleep parancssal érhetjük el a legegyszerűbben.

```
#include <unistd.h>

int main()
{
  for(;;) sleep(1);
}
```

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: https://github.com/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

Elsőnek is segédváltozóval oldjuk meg a problémát. Azaz létrehozunk még egy változót amiben tárolhatjuk az értéket.

```
#include <stdio.h>

int main()
{
  printf("segédváltozó csere\n\n");
int a=5;
int b=7;

  printf("eredeti: a=%d b=%d\n",a,b);
int c=a;
a=b;
b=c;

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

  return 0;
}
```

Ezután segédváltozó nélkül, szimpla összeadás kivonással oldjuk meg.

```
#include <stdio.h>

int main()
{
  printf("valtozocsere valtozo nelkul\n\n");
int a=5;
int b=7;
  printf("eredeti: a=%d b=%d\n",a,b);
a=a+b;
b=a-b;
a=a-b;
  printf("felcserelt: a=%d b=%d\n",a,b);
  return 0;
}
```

Végül exor-módszerrel is megoldjuk.

```
#include <stdio.h>

int main()
{
  printf("Csere exorral\n\n");
  int a=5;
  int b=7;
  printf("Eredeti értékek: a=%d b=%d\n",a,b);
  a=a^b;
  b=a^b;
  a=a^b;
```

```
    printf("Felcserélt értékek: a=%d b=%d\n", a,b);
    return 0;
}
```

2.4. Labdapattogás

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

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

Megoldás forrása:https://github.com/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

Labdapattogtatást egyszerű feltételvizsgálatokkal oldhatjuk meg, felvesszük a terminál méretét (initscr() function-nal), a labda kezdő helyét, gyorsaságát. Miután megadtuk a kezdő adatokat, "if" függvényekkel vizsgáljuk, hogy a labda az ablak széléhez ért-e, és irányt változtatunk. A labda gyorsaságának csökkentéséhez használhatjuk a usleep() függvényt, kép frissitéshez pedig a refresh()-t.

```
#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 );
  clear();

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

```
  if ( x>=mx-1 )
  {
      xnov = xnov * -1;
  }
  if ( x<=0 )
  {
      xnov = xnov * -1;
  }
  if ( y<=0 )
  {
      ynov = ynov * -1;
  }
  if ( y>=my-1 )
  {
      ynov = ynov * -1;
  }
    }

    return 0;
}
```

Ha nagyon kalandvágyóak vagyunk if nélkül is elérhetjük ezt.

```
#include <iostream>
#include <iomanip>
#include <unistd.h>

using namespace std;

int rajz(int x,int y, int h, int w)
{
    for(int i=1;i<=x;i++)
    {
  cout<<"X";
  for(int j=1;j<=w;j++)
  {
    cout<<" ";
  }
  cout<<" X"<<endl;
    }
    cout<<"X";
    for(int i=0;i<y;i++)
    {
  cout<<" ";
    }
    cout<<"o";
    for(int i=y;i<w;i++)
    {
  cout<<" ";
```

```
    }
    cout<<"X"<<endl;
    for(int i=x;i<h;i++)
    {
  cout<<"X";
  for(int j=1;j<=w;j++)
  {
    cout<<" ";
  }
  cout<<" X"<<endl;
    }

    return 0;
}

int main()
{
    int x=0,y=0;
    int width,height;

    cout<<"Add meg a terulet szelesseget es magassagat, amin szeretned, ↩
        hogy pattogjon a labda! \n";
    cin>>height>>width;
    while(true)
    {
  system("clear");
  cout<<" ";
  for(int i=0;i<width+1;i++)
  {
    cout<<"X";
  }
  cout<<endl;
  rajz(abs(height-(x++%(height*2))),abs(width-(y++%(width*2))),height, ↩
     width);
  cout<<" ";
  for(int i=0;i<width+1;i++)
  {
    cout<<"X";
  }
  cout<<endl;
  usleep(50000);
    }
    return 0;
}
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: https://github.com/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

Az 'int' méretét, és hogy hány bitet foglal, bitshifteléssel nézzük meg. A 'while' függvényben folyamatosan balra lépegetünk, és minden lépésnél növeljük a 'hossz' változót, amig a 'wat' változó nulláig nem ér.

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

Az 'int' 32 bites szó.

BogoMips

A BogoMips egy algoritmus ami a Linux kernelben méri fel a processzor sebességét az ú.n. 'busy-loop' konfigurálásához. A 'busy-loop' annyit jelent, hogy egy process folytonosan egy feltételt vizsgál, amíg az igazat ad vissza értékül, BogoMips azt mutatja hány mp-ig áll a proci, tehát nem csinál semmit.

```
#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;
}
```

Bitshifteléssel megyünk a while ciklusban végig a 2 hatványain. A ticks-ben tároljuk mennyi processzoridőt használt a CPU eddig, majd a delay() függvénynek átadjuk loops_per_sec változót (aminek a bitjei mindig odébb vannak eggyel tolva), ahol elszámolunk 0-tól a változó végéig. Ezután megint lekérjük a processzoridőt kivonva az előző ticks-ben tárolt CPU időt, így megkapjuk, mennyi ideig tartott a elszámolni a loops_per_sec változó végéig. Majd megnézzük if-el, hogy nagyobb vagy egyenlő a kapott ticks, mint a CLOCKS_PER_SEC aminek az értéke 1 millió és ha ez igaz, akkor kiszámoljuk, hogy milyen érték kell ahhoz,hogy a ciklusértékéket megkapjuk, ezzel meghatározva a CPU sebességét.

2.6. Helló, Google!

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

Megoldás forrása: https://github.com/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

A Page Rank az interneten található oldalakat rangsorolja. Kezdetben minden oldalnak van egy szavazati pontja és ha az egyik linkeli a másikat, akkor a linkelt oldal megkapja a linkelő pontját. Tehát egy oldal akkor lesz előkelőbb helyen egy google kereséskor, ha minél több másik oldal linkel rá, illetve ezen oldalakra is minél többen linkelnek, annál jobb minőségűnek fog számítani egy linkelése vagy szavazata.

```
#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;
    }
```

Az L nevű többdimenziós tömbben vannak rögzítve mátrix formájában az adatok a linkelésekről, melyik oldalra melyik oldal linkel és mennyit. A végtelen ciklusban nullázzuk PR összes elemét, majd rögtön hozzáadjuk az L mátrix és PRv vektor szorzatainak értékét. Ezután a távolság metódusunkban végigmegyünk a PR és PRv vektorokon és egy változóban eltároljuk ezek különbségének a négyzetét (hogy ne legyen negatív) és gyököt vonva visszaadjuk az értéket, amely ha kisebb mint 0.00000001, akkor kilépünk a végtelen ciklusból, ellenkező esetben pedig PRv tömböt feltöltjük PR elemeivel. Végül kiiratjuk az értékeket.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

A Brun tétel azt mondja, hogy az ikerprímszámok reciprokából képzett összege konvergál egy számhoz. Ezt határt Brun konstansnak nevezzük. Ezzel ellentétben a prímszámok a végtelen felé tartanak.

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

A program először a primeket számolja ki x-ig. Ezután az egymásmellett álló prímeket kivonja egymásból. Az 'idx' azt nézi, hogy ha a különbség kettő, akkor azok ikerprímek. 't1primes' változóba azokat a vizsgált számpároknak első elemét tároljuk melyeknek különbsége 3.

't2primes' változóba azt a vizsgált számpár második elemét mentjük, ahol a különbség 4. Ezután jön a matek része, a függvény reciprokokat hoz létre, azokat összeadja. Ezeket kiírjuk, és láthatjuk merre konvergál. A prímek felső határhoz konvergálnak.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

A Monty Hall probléma egy amerikai televíziós vetélekedőben jelent meg, ahol a műsor végén a játékosnak 3 ajtó közül kellett választania. A nyeremény csak az egyik ajtó mögött volt. A játékos választása után a műsorvezető kinyitott egy üres ajtót és feltette a kérdést, hogy fenntartja-e a választását a játékos vagy egy másik ajtót választ. A Monty Hall probléma kérdése, hogy számít-e, hogy a játékos megváltoztatja-e a választását. Józan ésszel gondolkodva nem számít, mivel a maradék két ajtó közül az egyik mögött van a nyeremény, így 50-50% az esélye annak, hogy nyerünk. A feladvány bizonyítása több matematikai professzoron is kifogott, köztük a világhírű Erdős Pálon is, akit csak a számítógépes szimuláció győzött meg, ami alapján számít, hogy másik ajtót választunk, ugyanis ekkor megduplázódik az esélyünk a nyerésre. Amikor először választunk ajtót, akkor 1/3 az esélye annak, hogy eltaláljuk a nyertes ajtót és 2/3, hogy nem. Ezután a játékvezető kinyit egy ajtót, amelyik üres és ha nem változtatunk a döntésünkön, továbbra is 1/3 lesz annak az esélye, hogy nyertünk. Viszont mivel már csak 2 ajtó van a játékban ezért ha változtatunk, akkor 2/3 lesz az esélyünk a nyerésre.

```
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 eltároljuk, hogy hány kisérletet végezzünk el, majd a kiserlet és jatekos vektorokban kisorsolunk 1

és 3 között véletlenszerűen számokat. A műsorvezető vektorát beállítjuk a kísérletek számával. Ezután egy for ciklussal végigmegyünk minden kísérleten és ha kiserlet i-edig értéke megyegyezik a a játékos i-edig találatával, az jelenti, hogy eltalálta a játékos a nyereményt és a mibol vektorba az a két érték kerül be amelyeket a játékos nem választott. (Ez a két érték az üres ajtókat jelenti ebben az esetben.)

Ha a játékos nem találta el elsőre a kiserlet vektorban található számot, akkor a mibol vektorba már csak 1 egy érték kerülhet, az amelyik nem a nyeremény és nem a játékos által kiválasztott érték. Ezután a musorvezeto vektorba berakjuk a mibol vektorban található számot, illetve ha két érték van benne akkor a kettőből az egyiket véletlenszerűen.

Ezután értkezik a kiértékelés. A nemvaltoztatesnyer vektorba kerülnek azok az esetek, amikor elsőre eltalálja a játékos a megfelelő ajtót. Megint végigmegyünk a kísérleteken és a holvalt vektorba azok vagy az az érték kerül az 1, 2 és 3 közül amely nem egyenlő a műsorvezető és a játékos által választottal vagyis ekkor ha váltana a játékos akkor nyerne. A változtat vektorba pedig a holvalt vektor elemei közül az egyiket rakjuk át.

A valtoztatesnyer vektorba pedig azok az értékek kerülnek, amelyek a kiserlet vektorba és a változtat vektorba találhatóak, vagyis ekkor az az ajtó a nyertes ,amelyiket másodjára választanánk. Eztuán pedig kiiratjuk az esetek számait:

```
[1] "Kiserletek szama: 1000000"
> length(nemvaltoztatesnyer)
[1] 333590
> length(valtoztatesnyer)
[1] 666410
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.5005777
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 1000000
```

3. fejezet

Helló, Chomsky!

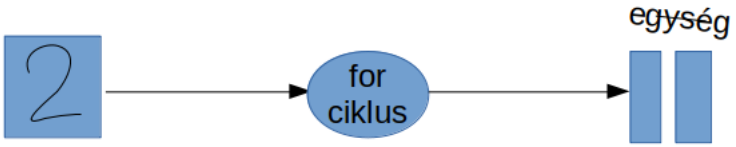
3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...



Az unáris számrendszer, azaz egyes számrendszer, lényege, hogy a számot egységekkel jelezzül, pl a 2-őt két db vonallal, és igy tovább. Az alábbi program bekér egy számot, majd a 'for' ciklussal annyiszor írjuk ki az egységet (jelen esetben 'I').

```
int main ()
{
  int x;
  printf ("Adjon meg egy szamot:");
  scanf ("%d", &x);
  for (int i = 0; i < x; i++)
    {
      printf ("I");
    }
  return 0;
}
```

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

```
        #include <stdio.h>
int main ()
{
  // Printing to screen.
  printf ("Hello World\n");
}
```

C99-et használva lefordul. :)

```
batf41@batf41-VirtualBox:~/Asztal$ gcc -o c89 -std=c99 c89.c
batf41@batf41-VirtualBox:~/Asztal$
```

C89-el már nem. :(

```
batf41@batf41-VirtualBox:~/Asztal$ gcc -o c89 -std=c89 c89.c
c89.c: In function ‘main’:
c89.c:4:7: error: C++ style comments are not allowed in ISO C90
      // Printing to screen.
       ^
c89.c:4:7: error: (this will be reported only once per input file)
batf41@batf41-VirtualBox:~/Asztal$
```

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Elsőnek is készítünk egy változót, amelyben majd taroluk a számok mennyiségét. Ezt majd növeljük minden szám beírása után. Majd a számjegyeket 0-9-ig definiáljuk 'digit' név alatt.

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

A következő részben megadjuk a programnak a mintát amit keresnie kell.

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

A '{digit}*' rész azt jelenti hogy bármennyi digit, tehát szám lehet.

```
{digit}*(\.{digit}+)?
```

A '\.' részen a pont önmagában annyit tesz hogy bármilyen karakterre rá lehet illeszteni. Viszont jelen esetben nekünk a '\' jellel le kell "védenünk". Így a program a valós számoknál lévő pontot fogja értelmezni.

A '\.' után jönnek a számjegyek, a '+' pedig annyit jelent, hogy legalább egy digitnek lennie kell a pont után.

```
(\.{digit}+)?
```

Ezután kiírjuk a számot amit a program felismert, 'atof'-al pedig a double tipusba konvertált változatát is.

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

Végül lehívjuk a lexert, és szépen elvégez nekünk minden maradékot.

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

A teljes kód így néz ki:

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

```
digit [0-9]
%%
{digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
 yylex ();
 printf("The number of real numbers is %d\n", realnumbers);
 return 0;
}
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

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

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

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

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

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

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

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

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

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

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

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

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A háromszögmátrix, nevétől eltérően, valójában kvadratikus mátrixok, tehát 4 sarka van, egyenlő oldalakkal. Háromszög nevét abból kapta, hogy a kvadratikus mátrixnak a főátlója adja a háromszög átfogó oldalát. Ha a főátló feletti számok mind 0-ák, az az alsóháromszög, ha az átló alattia számok nullák, akkor pedig felsőháromszög. Tehát amelyik résznél számokat látunk, arra áll a háromszög. (duh)

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

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

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

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

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

    for (int i = 0; i < nr; ++i)
    {
```

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

      }

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

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

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

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

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

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

      free (tm);

      return 0;
}
```

Elsőnek deklaráljuk a változóinkat. 'nr'-ben a háromszögmátrix sorainak a kívánt számát állítjuk le. '**tm' egy mutató, ami a tömbünk első elemére mutat, ami a double* mutatokra, tehát a tömb soraira mutat. Csak a nem nulla értékeket tároljuk, mivel a háromszöghöz felesleges ezeket az értékeket mutatni, így nem négyzet alakot kapunk. Egy 'i' sornak 'i+1' oszlopot foglalunk le.

Feltöltjük a struktúrát nem nulla értékekkel, majd kiiratás következik, és felszabadítjuk a helyet.

Az alábbi módon hivatkozunk 2D tömb elemeire:

'tm[3][0] = 42.0;'

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az EXOR titkosítás a logikai vagy-ra, másnéven a XOR müveletet használja, ami bitenként összehasonlít két 'operandus'-t, és a bittek értékétől függően ad vissza egy értéket. Ha a két bit megegyezik akkor 0-át, minden más esetben 1-et. Az operandusok amiket most használni fogunk, a titkosításhoz használt kulcs, és maga a titkosítandó szöveg/bemenet. Ahhoz hogy garantáltan (szinte) feltörhetetlen kódolást kapjunk, a kulcsnak és a bemenetnek megegyező méretűnek kell lennie; ugyanis ha a kulcs kisebb, az a kulcs ismétlését kényszeriti, ami már gyengít a titkosítás erősségén.

```
#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);
    }
}
```

Elsőnek konstansokat deklaráljuk. A kulcs és a méret maximum értékét állítjuk be. A konstans nem változtatható később. Szokás, hogy az előre meghatározott konstansokat nagybetűkkel írjuk, ebből is látszódjon a típusa.

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

A main-nek most argumentumokat adunk, a terminálon keresztül. Az 'argc' az argumentumok száma, a '**argv' pedig az argumentumokra mutató mutatók, amiket tömbbe tárolunk.

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

Deklarálunk két tömböt, a kulcsnak, és a buffer-nek. A bufferben a beolvasott karaktereket tároljuk. Mindkét tömb mérete korlátolt, lásd konstans.

```
    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

Készítünk két változót, amit számlálónak használunk majd. Ezek segítségével járjuk majd be a kulcs tömböt, megszámolva a beolvasott bájtokat. Használni fogjuk a 'strncpy()' függvényt, másnéven "string copy". Ez átmásol forrásból célba megadott mennyiségű karaktert. Itt a forrás a 'kulcs', a cél az 'argv[1]', és a hossza a konstans.

```
    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 'while' ciklus addig tart, amig a 'read' be nem olvassa a megadott mennyiségű bájtokat. Maga a 'read' három argumentumot kap: - az első a 'file descriptor', ami a forrás fájl lenne, de mi most standard inputrol szedjük az adatokat. - a második a 'buffer', ebben tároljuk a bájtokat amig végig nem megy a 'read', amit a harmadik argumentum határoz meg. - a harmadik a 'BUFFER_MERET', ez adja meg mennyi bájtot olvassunk be. Tehát honnan, hol tároljuk, és mennyit.

Ezután jön a titkosítás. Egyenként végig megyün a 'buffer'-ben eltárolt karaktereken, azokat össze 'EXOR-ozzuk' a 'kulcs' karaktereivel, amin a 'kulcs_index' megy végig, és a 'kulcs_index'-et növeljük 1-el, amíg el nem éri a 'kulcs_meret' értékét, ezután lenullázódik.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/

Tanulságok, tapasztalatok, magyarázat...

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

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: https://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia

Tanulságok, tapasztalatok, magyarázat...

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

```
#define _GNU_SOURCE

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

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

                      && strcasestr (titkos, "hogy") && strcasestr ( ↩
                         titkos, "nem")
                      && strcasestr (titkos, "az") && strcasestr (titkos, ↩
                          "ha");

              }

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

              int kulcs_index = 0;

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

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

              }

          }

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

                  exor (kulcs, kulcs_meret, titkos, titkos_meret);

                 return tiszta_lehet (titkos, titkos_meret);

              }
```

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

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


            for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
                    titkos[p - titkos + i] = '\0';


for (int ii = '0'; ii <= '9'; ++ii)
  for (int ji = '0'; ji <= '9'; ++ji)
    for (int ki = '0'; ki <= '9'; ++ki)
for (int li = '0'; li <= '9'; ++li)
  for (int mi = '0'; mi <= '9'; ++mi)
    for (int ni = '0'; ni <= '9'; ++ni)
      for (int oi = '0'; oi <= '9'; ++oi)
  for (int pi = '0'; pi <= '9'; ++pi)
    {
      kulcs[0] = ii;
      kulcs[1] = ji;
      kulcs[2] = ki;
      kulcs[3] = li;
      kulcs[4] = mi;
      kulcs[5] = ni;
      kulcs[6] = oi;
      kulcs[7] = pi;

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

      // ujra EXOR-ozunk, igy nem kell egy masodik buffer
      exor (kulcs, KULCS_MERET, titkos, p - titkos);
    }
```

Ezzel a programmal fel tudjuk törni az EXOR titkosításokat. Pontosabban mondva, ezzel 100%-osan csak az előzőt tudjuk feltörni. Ugyanis előre tudjuk hogy 8 karakteres a kulcs, így ehhez tudjuk igazítani a

feltörőnket.

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

Először, mint mindig, definiáljuk a konstansainkat. A kulcs méretet előre tudjuk, így azt 8-ra állítjuk.

```
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;
}
```

'atlagos_szohossz' függvénnyel a titkos szöveg szó átlagát nézzük meg. Az 'sz' változóban a szavak számát tároljuk, amit a 'for' ciklussal rakunk bele, úgy, hogy minden szóköznél adunk hozzá egyet. A függvény visszatérő értéke a titkosított szöveg mérete elosztva a szavak számával.

```
int
tiszta_lehet (const char *titkos, int titkos_meret)
{
        && strcasestr (titkos, "hogy") && strcasestr ( ↩
           titkos, "nem")
        && strcasestr (titkos, "az") && strcasestr (titkos, ↩
            "ha");
}
```

A 'tiszta_lehet' függvény azt nézi, hogy van-e már tiszta, azaz nem titkosított szöveg rész/szó a forrásban. Egyszerűen csak keresi a leggyakrabban használt magyar szavakat. Ezzel viszont csak magyar szövegre szűkítettük a lehetőségeinket a feltörésben.

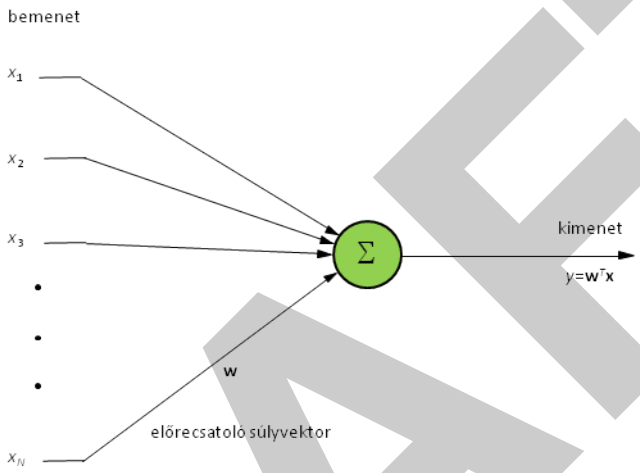
```
void
exor (const char kulcs[], int kulcs_meret, char titkos[], int  ↩
   titkos_meret)
{
    int kulcs_index = 0;

    for (int i = 0; i < titkos_meret; ++i)
    {
        titkos[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}
```

Itt lényegében megismételjük a titkosítást, mivel ha valamit kétszer EXOR-ozunk (a megfelelő kulcsal), megkapjuk az eredetit. Argumentumként kap egy kulcsot, annak méretét, a titkos szöveget, és annak méretét.

4.5. Neurális OR, AND és EXOR kapu

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

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



4.1. ábra. Neuron


```
#   Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#   This program is free software: you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program.  If not, see <http://www.gnu.org/licenses/>
#
# https://youtu.be/Koyw6IH5ScQ

library(neuralnet)
```

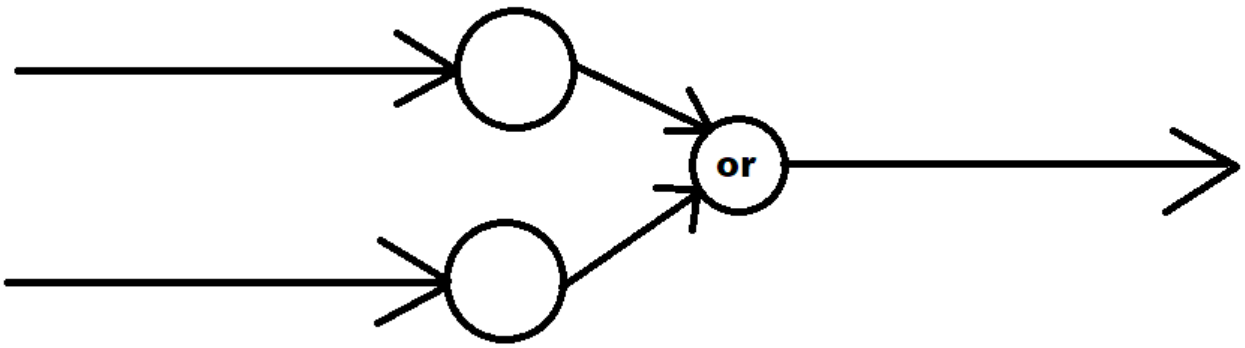
```
a1     <- c(0,1,0,1)
a2     <- c(0,0,1,1)
OR     <- c(0,1,1,1)

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])
```

Bontsuk fel részekre, a megértés érdekében.

```
library(neuralnet)
```

Itt lekérjük a neuralnet csomagot, ezt fogjuk használni a programunkban.




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

Implementáljuk a szabályokat. ’a1’ és ’a2’-ben megadjuk a bemeneti értéket, az ’OR’-ban pedig a kívánt eredményt. Itt a szabály az, hogy ahol a két vizsgált érték eltér egymástól, 1-est ad vissza, minden más esetben 0. Megkezdődik az öntanítás, amint futtatjuk a programot, befejeződik.

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

Átalakítjuk adattá a gépnek.

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

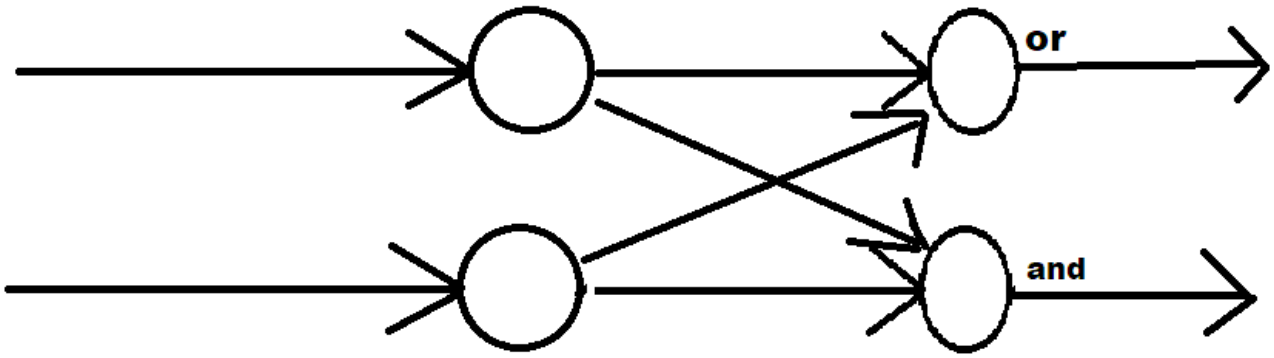
```
plot(nn.or)
```

Itt használt vesszük az elején meghívott library-nak. A neuralnet függvény neurális hálót készít az adatokból.

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

Compute függvény itt elvégzi a számolást, és vissza adja az eredményt.

A következő részben annyi változik, hogy most hozzá rakjuk az 'AND' kapcsolót.




```
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])
```

4.6. Hiba-visszaterjesztéses perceptron

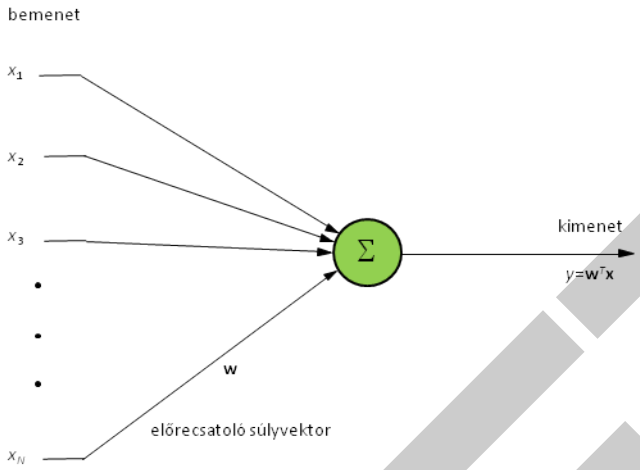
C++

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

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

Továbbra is a neuron hálózatokal foglalkozik ez a feladat. Most a perceptronokról lesz szó. A lényeg hasonló az előzőhöz, a Neuron kap egy bemenetet, és akkor aktiválódik, ha elér egy bizonyos pontot és ad egy kimenetet.

Ez az algoritmus betanítja a számítógépet a bináris osztályzásra. A forrás kód igen hosszú, így inkább egy linken adom meg, a "megoldás forrása" részen megtalálható. Készítette: Bántfai Norbert.



4.2. ábra. Neuron

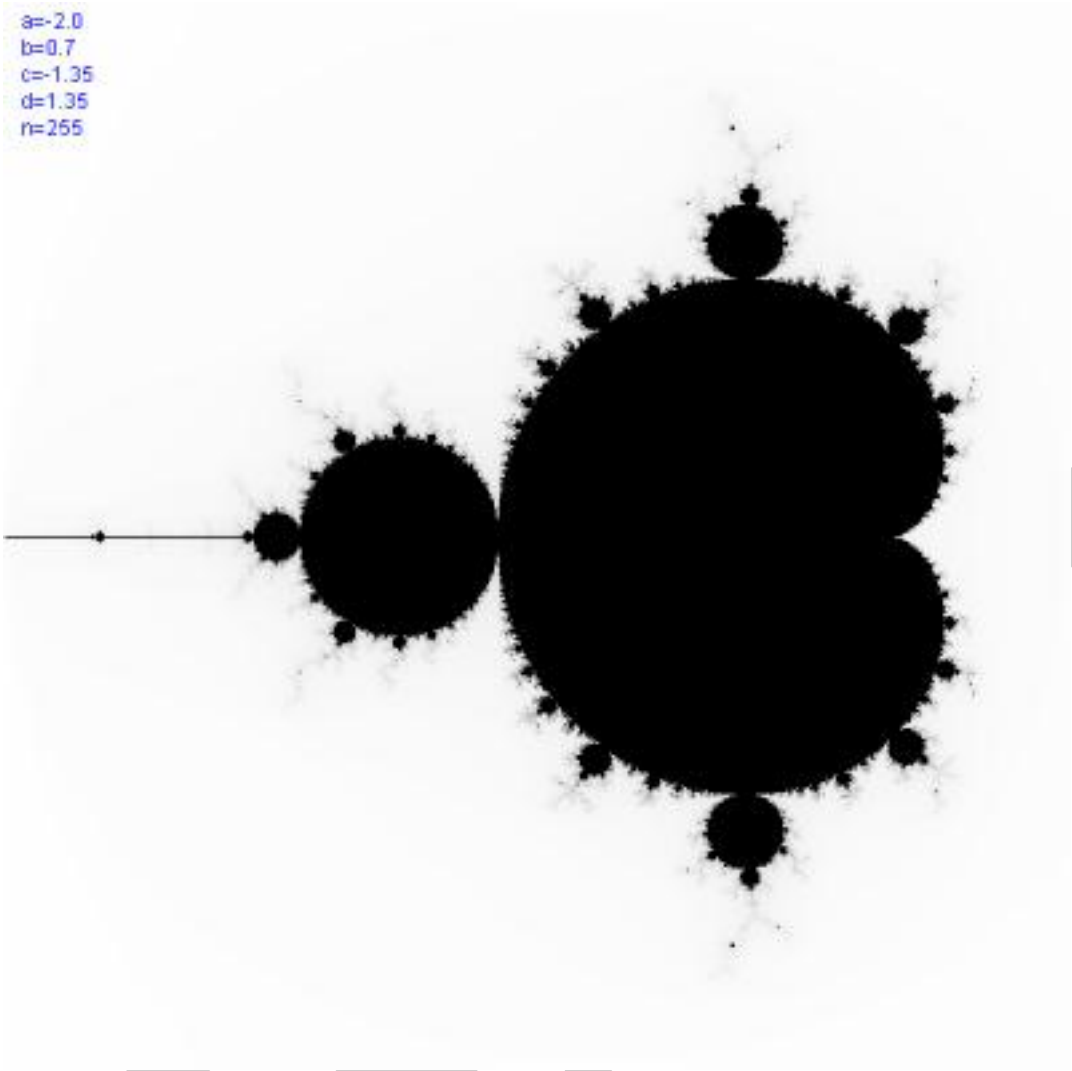
```
#include <iosteam>
#include "mlp.hpp"
#include <png++/png.hpp>

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

png::image <png::rgb_pixel> png_image (argv[1]);

int size = png_image.get_widith() *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}; i<png_image.get_width();++j)
    image[i*png_image.get_width()+j]=png_image[i][j].red;

double value =(*p)(image);

std::cout<< value<<std::endl;

delete p;
delete [] image;
}
```

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

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



5.1. ábra. Mandelbrot

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

int main (int argc, char *argv[])
{
  if (argc !=2)
  {
    std::cout << " Missing starter argument " << std::endl;
    std::cout << " use like this: ./file filename ";
    return -1;
  }

  double a = -2.0;
  double b = 0.7;
  double c = -1.35;
  double d = 1.35;
```

```
  int height = 1024;
  int width = 768;
  int iterationHatar = 255;

  png::image <png::rgb_pixel> image (width, height);

  double dx= (b-a) / width;
  double dy= (d-c) / height;

  double reC;
  double imC;
  double reZ;
  double imZ;
  double newreZ;
  double newimZ;

  int iteration = 0;

  for (int i=0; i<height; ++i)
  {
    for(int j=0; j<width; ++j)
    {
      reC = a+k*dx;
      imC = d-j*dy;
      reZ = 0;
      imZ = 0;
      iteracio = 0;

      while (reZ*reZ + imZ* imZ < 4 && iteration < iterationHatar)
      {
        newreZ = reZ*reZ - imZ*imZ + reC;
        newimZ = 2*reZ*imZ + imC;
        reZ = newreZ;
        imZ = newimZ;

        ++iteration;
      }

    image.set_pixel(i, j, png::rgb_pixel(255-iteration%256, 255-iteration ↩
       %256, 255-iteration%256));
    }
  }

  image.write (argv[1]);
  std::cout << argv[1] << " Done " << std::endl;

}
```

A Mandelbrot-halmaz egy síkbeli alakzat, amelyet egy alapvetően nagyon egyszerű algebrai összefüggés bonyolultabb (végtelennel kapcsolatos, analitikus fogalmakat, határértékszámítást igénylő) elemzése ad meg, rajzol ki. Ezeknek az összefüggéseknek a még legegyszerűbb (bár nem az egyetlen lehetséges) megközelítési módja a komplex számok felhasználásával történhet. A Mandelbrot-halmaz a komplex számsíkon ábrázolható alakzat, amely számsík geometriailag semmiben sem különbözik a jól ismert („euklideszi") síktól, csak a pontok számokkal való leírása más. (wikipedia)

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

Elsőnek is szükségünk lesz a libpng, és libpng++ könyvtárakra, ezeket telepítenünk kell, amihez root jogok kellenek majd. Ezt felhasználva fogjuk létrehozni a képet ami tartalmazza magát a mandelbrotot.

A 'main()'-ben legelső dolgunk hogy ellenőrizzük a helyes kezdő argumentumokat. Ha nem megfelelő, hiba üzenetet dobunk, leirjuk hogyan is kéne használni, és kilépünk.

Megadjuk az értékészletet, és az iterációs határt, azaz az értelmezési tartományt. Használva a png++/png.hpp header fájlt, létrehozzunk egy üres png-t, megadjuk a paramétereit. Két 'for' ciklussal végig pásztázzuk a rácsot, mármint a png "pixeleit". Ahogy végig járjuk az értelmezési tartományt, megadjuk a 'c' számot, ehhez kiszámoljuk a 'z' értékét.

Következik egy 'while' ciklus, ami addig fut, míg a 'z^2' kisebb mint 4, illetve amíg az értelmezési tartományon belül vagyunk. És mi van ha elérjük a határt? Azt jelenti, hogy konvergens az iteráció, tehát a Mandelbrot eleme lesz a 'c'.

Utolsó lépésként megadjuk a png színeit, itt feketére szinezzük a mandelbrot figurát, majd kiiratjuk az

```
  image.write([1]);
```

parancssal.

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

Az előző programot átgondolva, úgymond magasabb szintre helyezzük. A '<complex>' könyvtár segitségével most az előzővel ellentétben nem két változóban mentjük el a komplex számokat, szétválasztva képzetes és valós részre, hanem egyben.

```
#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;

}
```

Az új elemek a programban az 'atoi' és az 'atof'. Ezeknek feladata, hogy a felhasználó által megadott stringeket 'int' illetve 'double' típussá irják át. 'atof' 'double'-ra, 'atoi' 'int'-re.

Itt is létrehozzuk az üres png-t, amin ugyanúgy végig pásztázunk két egymásba ágyazott 'for' ciklussal a rácsot, beállítgatva a pixeleket. Most azonban teljes színkavalkádot fogunk kapni.

A 'for' ciklusban már használjuk is a 'complex' könyvtárat. Alatta lévő 'while' részben abszolut értékeket számolunk 'abs' használatával.

Ahhoz hogy futtasuk, meg kell adnunk a cél fájl nevét, a kép méreteit, az iterációs határt, és négy értéket. Mindet egy sorba.

./complex complex.png 1920 1080 1020

0.41276554182095892553405747094075195491 31

0.41276554182458180530801428176346234977 25

0.21353870517687464913869632709975121542 81

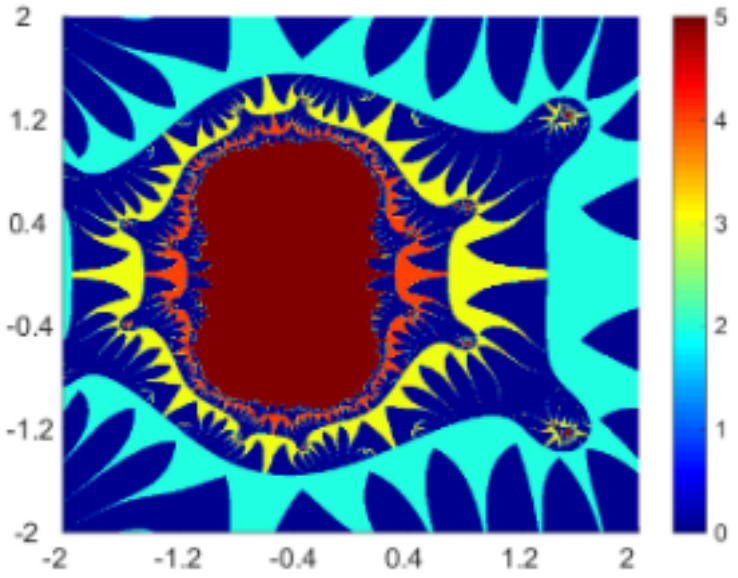
0.21353870518049752891265313792246161028 74

5.3. Biomorfok

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

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

Megoldás forrása: http://www.t-es-t.hu/minden/kaosz/mandel.htm



(b) $f(z) = z^2 + \sin z,\ c = 0.285$

5.2. ábra. Biomorf

Mindenek előtt beszélnünk kell a Júlia halmazokról. Mi is az, mi köze van a biomorfokhoz?

Az I. világháború idején Gaston Julia és Pierre Fatou francia matematikusok kutatásaik során furcsa halmazokat kaptak eredményül. Ezeket később Julia-halmazoknak nevezték el.

A Julia-halmazok formailag nagyon sokfélék. A számítógépek korában már könnyen megjeleníthetőek, így kiderült, hogy olyan bonyolult alakzatok ezek, amelyeket a végtelenségig nagyítva sem ismerhetünk meg teljes egészében. Vagyis szintén fraktálok.

Benoit Mandelbrot 1979-ben fedezett fel egy egyszerű szabályt, illetve ezen szabály által létrehozható képet, amely tartalmazza az összes Julia-halmazt.

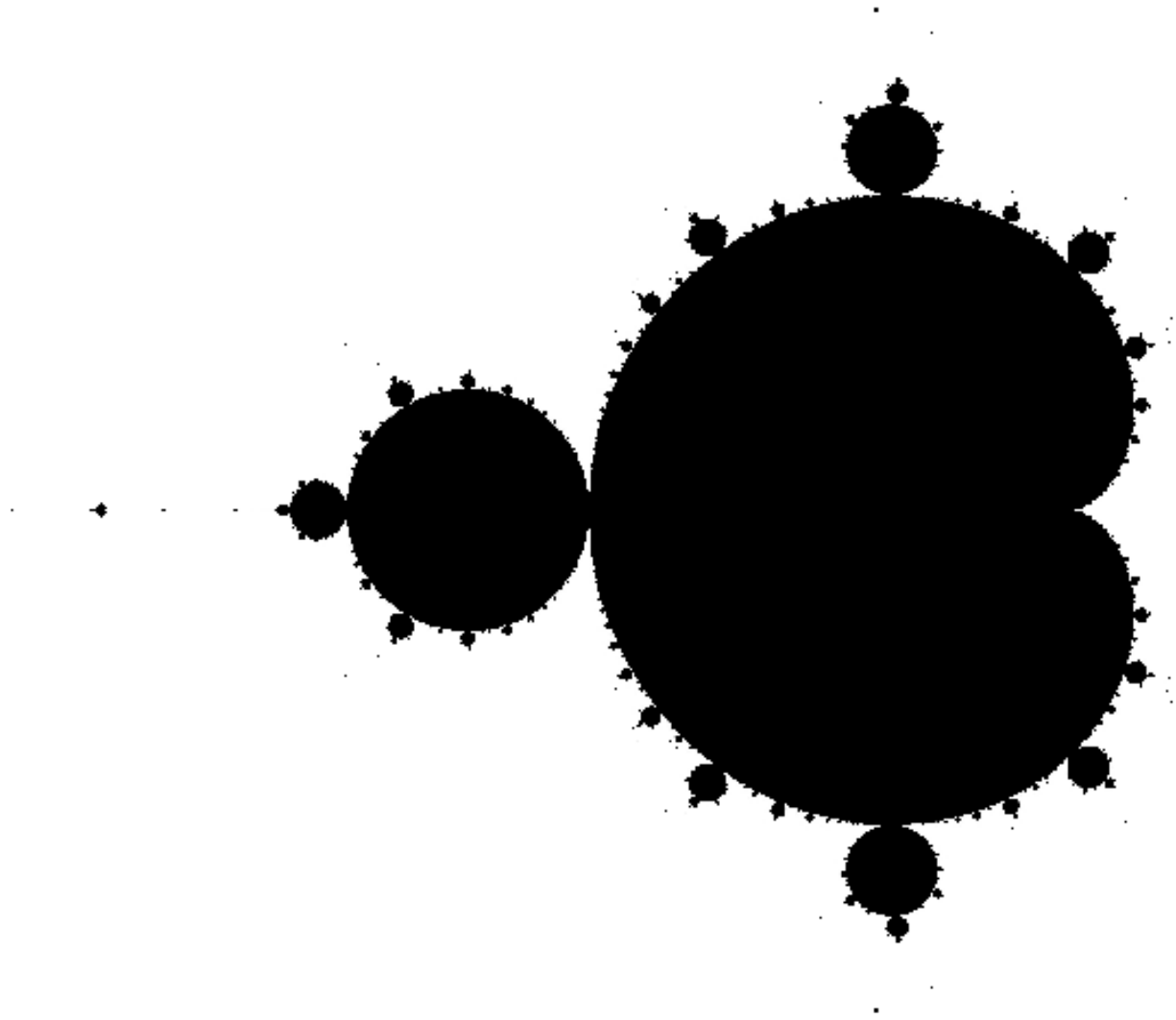
További érdekességekért keresd fel ezt az oldalt: http://www.t-es-t.hu/minden/kaosz/mandel.htm

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
```

```
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

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

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

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

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

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

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

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

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

            double reZ = xmin + x * dx;
            double imZ = ymax - y * dy;
            std::complex<double> z_n ( reZ, imZ );

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

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

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

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

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

}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása



5.3. ábra. Mandelbrot CUDA-val készítve

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_parhuzamos_prog_linux/adatok.html

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

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

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


#define MERET 600
#define ITER_HAT 32000

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

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

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

  // c = (reC, imC) a rács csomópontjainak
  // megfelelő komplex szám
  reC = a + k * dx;
  imC = d - j * dy;
  // z_0 = 0 = (reZ, imZ)
  reZ = 0.0;
  imZ = 0.0;
  iteracio = 0;
  // z_{n+1} = z_n * z_n + c iterációk
  // számítása, amíg |z_n| < 2 vagy még
  // nem értük el a 255 iterációt, ha
  // viszont elértük, akkor úgy vesszük,
  // hogy a kiinduláci c komplex számra
  // az iteráció konvergens, azaz a c a
  // Mandelbrot halmaz eleme
  while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
      // z_{n+1} = z_n * z_n + c
      ujreZ = reZ * reZ - imZ * imZ + reC;
      ujimZ = 2 * reZ * imZ + imC;
      reZ = ujreZ;
      imZ = ujimZ;

      ++iteracio;
```

```
    }
  return iteracio;
}


/*
__global__ void
mandelkernel (int *kepadat)
{

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

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

}
*/

__global__ void
mandelkernel (int *kepadat)
{

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

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

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

}

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

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

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

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

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

```
}

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

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

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

  int kepadat[MERET][MERET];

  cudamandel (kepadat);

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

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

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

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

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

```
}
```

Most a Mandelbrotunkat az Nvidia CUDA technológiájával hozzuk létre. Ez nagyságrendellek meggyorsítja a műveletet. Ehhez természetesen szükséges egy Nvidia videókártya, emellé fel kell telepítenünk a 'nvidia-cuda-toolkit'-et. A varázslat ott történik hogy létrehozunk egy 600x600 blokkból álló rácsot, és minden blokkot egy szálhoz kapcsolunk, így felgyorsítva a műveletet.

A forráskód elején konstansokat definiálunk: a méretet és az iterációs határt. Ezután az előzőekkel megegyezően a 'mandel' függvénnyel létrehozzuk a halmazunkat.

'nvcc'-vel fogjuk fordítani a programot, ahhoz hogy a CUDA-t használhassuk. Ez két részre osztja a programot: Host-ra és egy eszközhöz kötött részre, amit az Nvidia fordító hoz létre. Ehhez deklarációknál használnunk kell a '__global__' és '__device__' kifejezéseket. Ez jelzi a CUDA-nak hogy, haver, ezt kéne gépi kóddá átabrakadabrálni, kösz.

A 'threadIdx.x/y' azt a szálat mutatja, amelyiken éppen az x és y számhoz tartozó érték számolódik ki.

5.5. Mandelbrot nagyító és utazó C++ nyelven

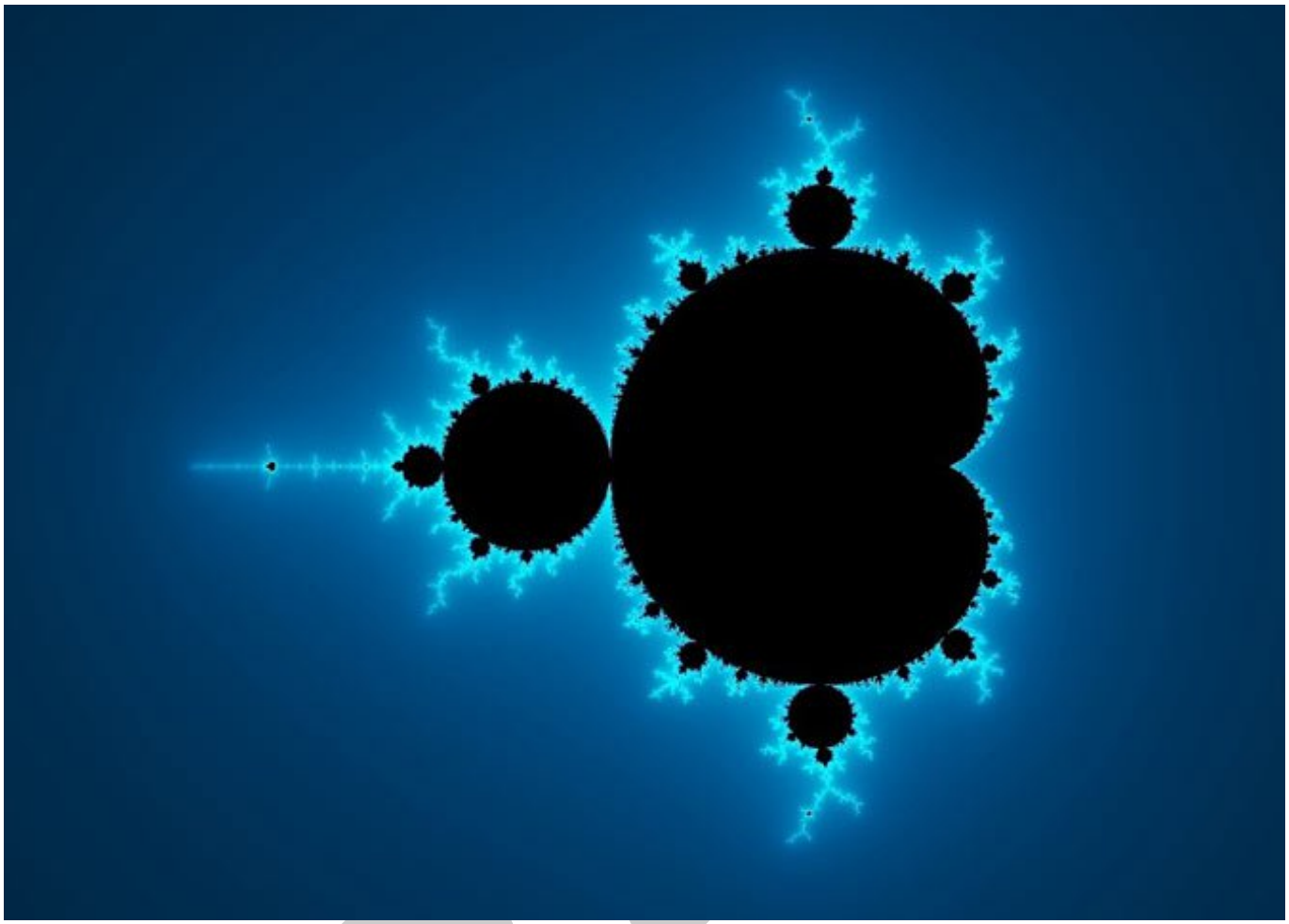
Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteréció bejárta z_n komplex számokat!

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

Itt is Mandelbrot, what a surprise. Azonban most képesek leszünk nagyítani is egyes részeket. Viszont ennek ára van, a futtatás és fordítás most igazi nehézség lesz, az előzőekhez képesz legalábbbis. 5 bemeneti fájlra lesz szükségünk, természtetesen egy mappában, hogy használni tudjuk.

Szükségünk lesz továbbá a libqt4-dev csomagra. Telepitéshet root jogok kellenek.

```
$ sudo apt-get install libqt5-dev
$ ls
frakablak.cpp frakablak.h frakszal.cpp frakszal.h main.cpp
$ qmake -project
$ ls
frakablak.cpp frakablak.h frakszal.cpp frakszal.h main.cpp Mandelnagy.pro
$ qmake Mandelnagy.pro
$ ls
frakablak.cpp frakszal.cpp main.cpp Mandelnagy.pro frakablak.h frakszal.h  ↩
   Makefile
$ make
```



5.4. ábra. Mandelbrot szinesen

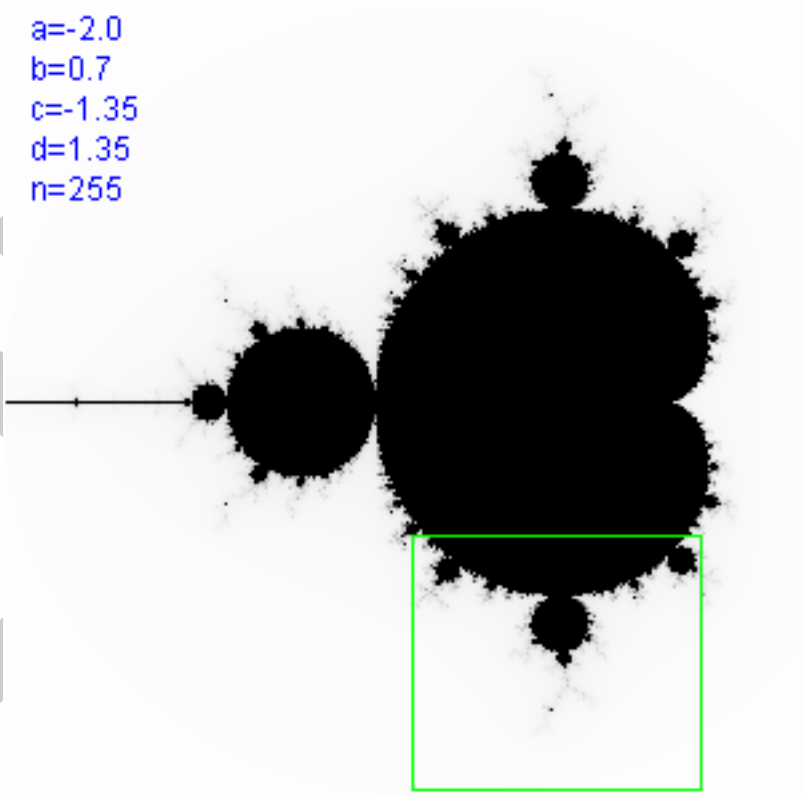
5.6. Mandelbrot nagyító és utazó Java nyelven

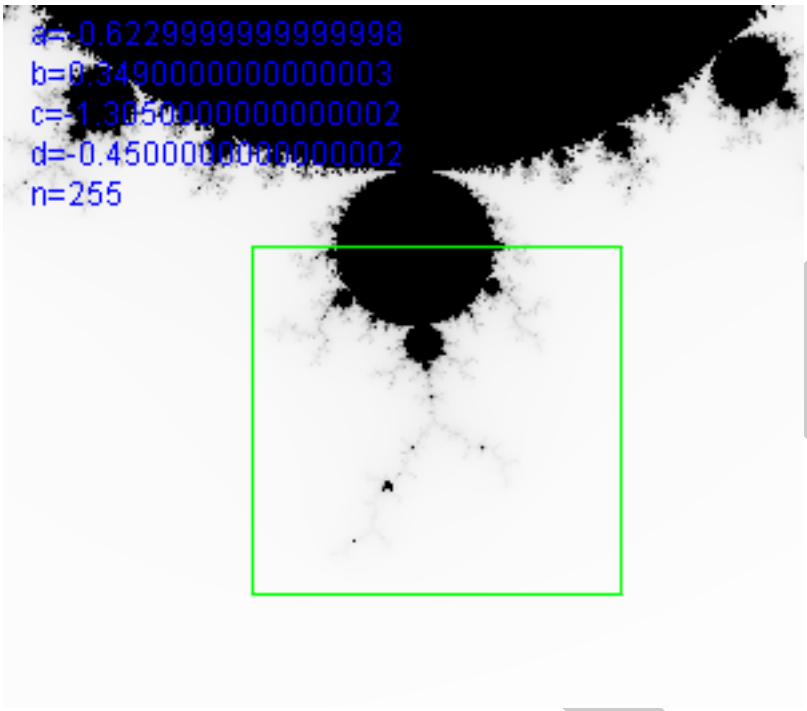
```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /** A nagyítandó kijelölt területet bal felső sarka. */
    private int x, y;
    /** A nagyítandó kijelölt terület szélessége és magassága. */
    private int mx, my;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nygítani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     *
     * @param      a              a [a,b]x[c,d] tartomány a koordinátája.
     * @param      b              a [a,b]x[c,d] tartomány b koordinátája.
     * @param      c              a [a,b]x[c,d] tartomány c koordinátája.
     * @param      d              a [a,b]x[c,d] tartomány d koordinátája.
     * @param      szélesség      a halmazt tartalmazó tömb szélessége.
     * @param      iterációsHatár a számítás pontossága.
```

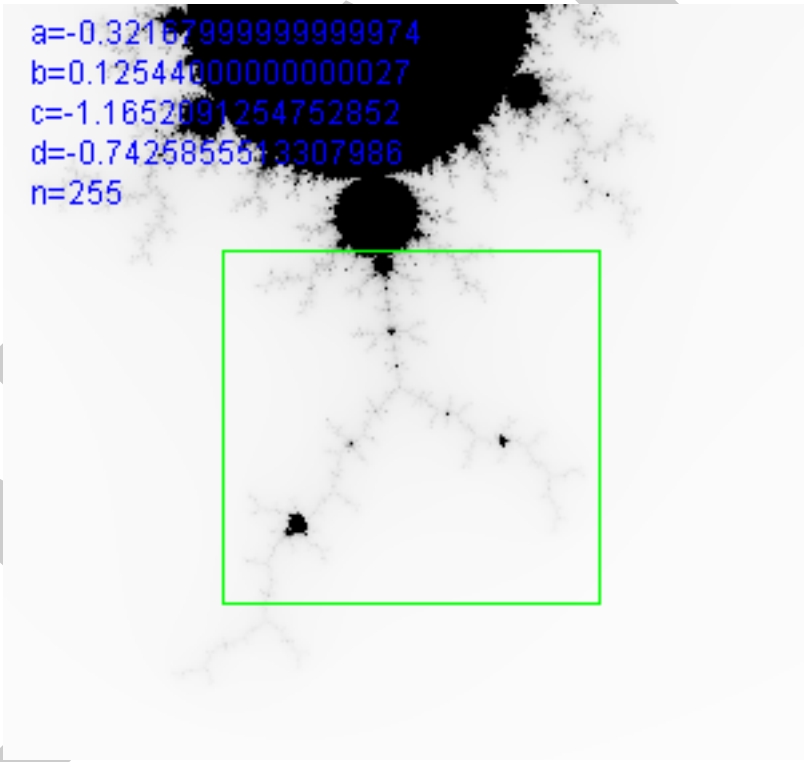
```
     */
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
            int szélesség, int iterációsHatár) {
        // Az ős osztály konstruktorának hívása
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        // Egér kattintó események feldolgozása:
        addMouseListener(new java.awt.event.MouseAdapter() {
            // Egér kattintással jelöljük ki a nagyítandó területet
            // bal felső sarkát:
            public void mousePressed(java.awt.event.MouseEvent m) {
                // A nagyítandó kijelölt területet bal felső sarka:
                x = m.getX();
                y = m.getY();
                mx = 0;
                my = 0;
                repaint();
            }
            // Vonszolva kijelölünk egy területet...
            // Ha felengedjük, akkor a kijelölt terület
            // újraszámítása indul:
            public void mouseReleased(java.awt.event.MouseEvent m) {
                double dx = (MandelbrotHalmazNagyító.this.b
                        - MandelbrotHalmazNagyító.this.a)
                        /MandelbrotHalmazNagyító.this.szélesség;
                double dy = (MandelbrotHalmazNagyító.this.d
                        - MandelbrotHalmazNagyító.this.c)
                        /MandelbrotHalmazNagyító.this.magasság;
                // Az új Mandelbrot nagyító objektum elkészítése:
                new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+ ↩
                   x*dx,
                        MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
                        MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
                        MandelbrotHalmazNagyító.this.d-y*dy,
                        600,
                        MandelbrotHalmazNagyító.this.iterációsHatár);
            }
        });
        // Egér mozgás események feldolgozása:
        addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
            // Vonszolással jelöljük ki a négyzetet:
            public void mouseDragged(java.awt.event.MouseEvent m) {
                // A nagyítandó kijelölt terület szélessége és magassága:
                mx = m.getX() - x;
                my = m.getY() - y;
                repaint();
            }
        });
    }
    /**
```

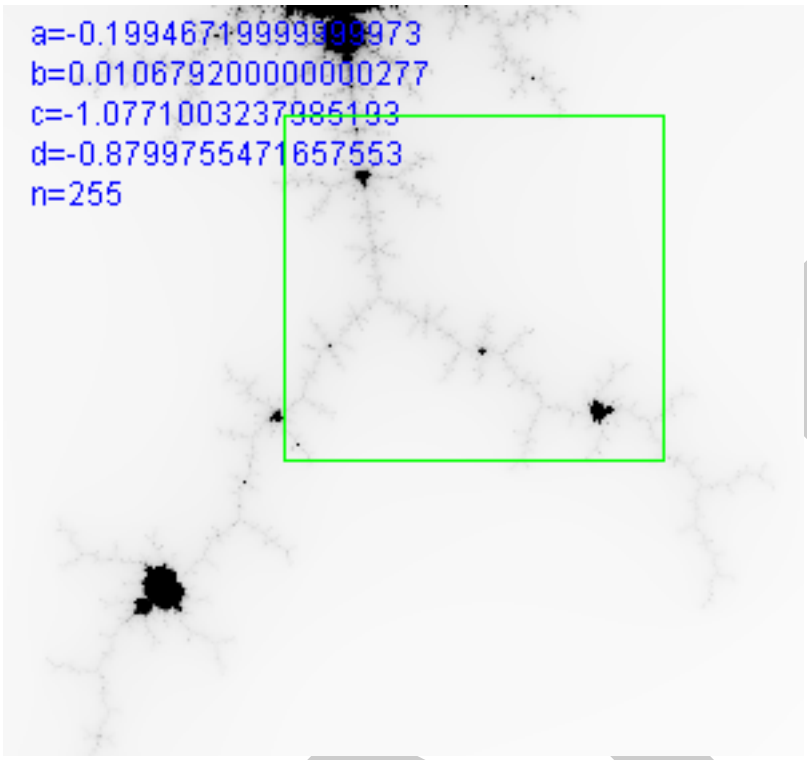
```
     * Pillanatfelvételek készítése.
     */
    public void pillanatfelvétel() {
        // Az elmentendő kép elkészítése:
        java.awt.image.BufferedImage mentKép =
                new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        java.awt.Graphics g = mentKép.getGraphics();
        g.drawImage(kép, 0, 0, this);
        g.setColor(java.awt.Color.BLUE);
        g.drawString("a=" + a, 10, 15);
        g.drawString("b=" + b, 10, 30);
        g.drawString("c=" + c, 10, 45);
        g.drawString("d=" + d, 10, 60);
        g.drawString("n=" + iterációsHatár, 10, 75);
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
        g.dispose();
        // A pillanatfelvétel képfájl nevének képzése:
        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("MandelbrotHalmazNagyitas_");
        sb.append(++pillanatfelvételSzámláló);
        sb.append("_");
        // A fájl nevébe belevesszük, hogy melyik tartományban
        // találtuk a halmazt:
        sb.append(a);
        sb.append("_");
        sb.append(b);
        sb.append("_");
        sb.append(c);
        sb.append("_");
        sb.append(d);
        sb.append(".png");
        // png formátumú képet mentünk
        try {
            javax.imageio.ImageIO.write(mentKép, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * A nagyítandó kijelölt területet jelző négyzet kirajzolása.
     */
    public void paint(java.awt.Graphics g) {
```

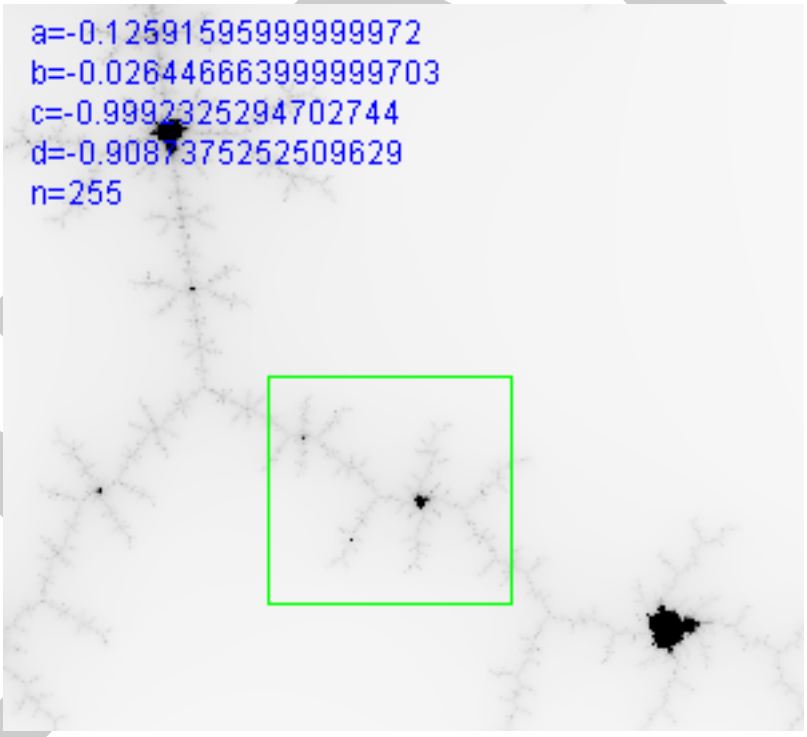
```
        // A Mandelbrot halmaz kirajzolása
         g.drawImage(kép, 0, 0, this);
        // Ha éppen fut a számítás, akkor egy vörös
        // vonallal jelöljük, hogy melyik sorban tart:
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
        // A jelző négyzet kirajzolása:
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
    }
    /**
     * Példányosít egy Mandelbrot halmazt nagyító obektumot.
     */
    public static void main(String[] args) {
        // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
        // tartományában keressük egy 600x600-as hálóval és az
        // aktuális nagyítási pontossággal:
        new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```




a=-0.6229999999999998
b=0.3490000000000003
c=-1.3050000000000002
d=-0.4500000000000002
n=255


a=-0.32167999999999974
b=0.12544000000000027
c=-1.1652091254752852
d=-0.7425855513307986
n=255


a=-0.19946719999999973
b=0.010679200000000277
c=-1.0771003237985193
d=-0.8799755471657553
n=255


a=-0.12591595999999972
b=-0.026446663999999703
c=-0.9992325294702744
d=-0.9087375252509629
n=255

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás forrása:

C++ program:

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

class Veletlen
{
  public:
    Veletlen();
    ~Veletlen(){};

    double get();

  private:
    bool vanElrakva;
    double ertek;
};

Veletlen::Veletlen()
{
  vanElrakva = false;
  std::srand (std::time(NULL));
};

double Veletlen::get()
```

```
{
  if (!vanElrakva)
  {
    double u1, u2, v1, v2, w;
      do
      {          //Algorithm starts
      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);
    ertek = r * v2;
    vanElrakva = !vanElrakva;

    return r * v1;           //Algorithm ends
  }
  else
  {
  vanElrakva = !vanElrakva;
  return ertek;
  }
};

int main ()
{
  Veletlen rnd;

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

Ehhez a programhoz lehívjuk a <cmath> és <ctime> könyvtárat, ugyanis szükségünk lesz az időre, a randomizáláshoz.

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

A 'Veletlen' class-unkban ketté szedjük a kódot: publikussá tesszük a konstruktort, destruktort, illetve egy függvényt, ami majd a random számokat fogja lekérni. Privátban pedig egy boolian változót, ami azt jelzi, hogy elvan e tárolva ugyan az a szám, mint amit épp generálunk, így elkerülve az ismétlést, illetve ennek a már meglévő számnak értékét is itt tároljuk.

```
    class Veletlen
```

```
{
  public:
    Veletlen();
    ~Veletlen(){};

    double get();

  private:
    bool vanElrakva;
    double ertek;
};
```

Értékeket rakunk a konstruktorunkba, ez fog lefutni minden indításkor elsőnek. A 'vanElrakva' boolian-t 'false' kezdő értékre állítjuk, és inicializáljuk a 'srand' függvényt, amelyet lehívtunk az elején.

```
Veletlen::Veletlen()
{
  vanElrakva = false;
  std::srand (std::time(NULL));
};
```

A lekérő függvényben történik a matek hókuszpókusz. Ha nincs eltárolva randomizált számunk, lefut az algoritmus, és vissza ad egy random számot két külön változóban. Az egyiket kiiratjuk, a másikat eltároljuk. Amennyiben viszont ha van eltárolva, egyszerüen csak kiiratjuk, és átváltjuk a 'vanElrakva' bool kapcsolónkat.

```
double Veletlen::get()
{
  if (!vanElrakva)
  {
    double u1, u2, v1, v2, w;
      do
      {          //Algorithm starts
      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);
    ertek = r * v2;
    vanElrakva = !vanElrakva;

    return r * v1;          //Algorithm ends
  }
  else
```

```
  {
  vanElrakva = !vanElrakva;
  return ertek;
  }
};
```

A 'main()'-ben csak meghívjuk az osztályunkat, és kiiratjuk a számokat for ciklussal.

```
int main ()
{
  Veletlen rnd;

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

```
rake@rake-desktop:~/BHAKSZ/Welch$ g++ polar.cpp -o p
rake@rake-desktop:~/BHAKSZ/Welch$ ./p
0.632766
-0.678855
-1.09296
-0.441097
-0.0241901
-1.40382
-0.0332053
0.113661
0.466147
-0.618171
```

Java:

```
public class polar
{
  boolean noSaved = true;
  double saved;

  public polar()
  {
  noSaved = true;
  }
  public double next()
  {
    if(noSaved)
    {
      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);
      saved = r*v2;
      noSaved = !noSaved;
      return r*v1;
    }
    else
    {
      noSaved = !noSaved;
      return saved;
    }
  }
  public static void main(String[] args)
  {
    polar g = new polar();
    for(int i=0; i<10; ++i)
    {
    System.out.println(g.next());
    }
  }
}
```

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!

```
rake@rake-desktop:~/BHAKSZ/Welch$ javac polar.java
rake@rake-desktop:~/BHAKSZ/Welch$ java polar
0.17893440014659887
-0.81947922846356
-0.19647809173484215
-1.1995248513359387
0.6569243187188295
-0.2912476938056324
0.5767102992766847
-0.2877210857376333
-1.0626943042449213
0.2258635782690551
```

Ha felkutatjuk a java development kit sajár Random.java fájlját, láthatjuk, hogy egy nálunk sokkal tapasztaltabb fejlesztő is hozzánk hasonlóan gondolkodva, szinte ugyanúgy írta le ezt az algoritmust.

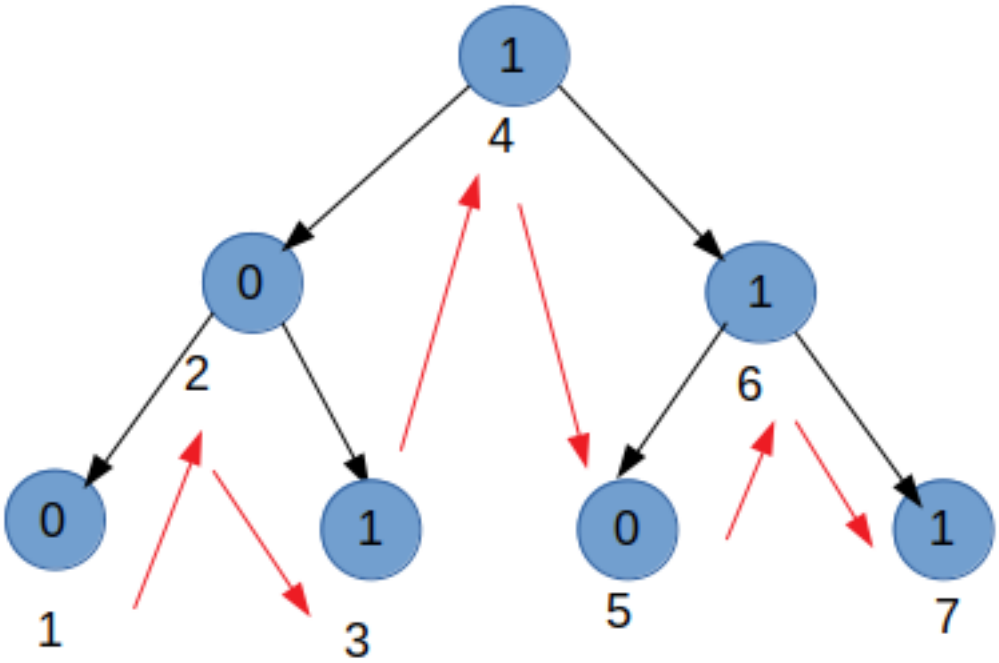
```
    synchronized public double nextGaussian() {
        // See Knuth, ACP, Section 3.4.1 Algorithm C.
        if (haveNextNextGaussian) {
            haveNextNextGaussian = false;
```

```
            return nextNextGaussian;
        } else {
            double v1, v2, s;
            do {
                v1 = 2 * nextDouble() - 1; // between -1 and 1
                v2 = 2 * nextDouble() - 1; // between -1 and 1
                s = v1 * v1 + v2 * v2;
            } while (s >= 1 || s == 0);
            double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
            nextNextGaussian = v2 * multiplier;
            haveNextNextGaussian = true;
            return v1 * multiplier;
        }
    }
```

6.2. LZW

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

Megoldás forrása: https://progpater.blog.hu/2011/02/19/gyonyor_a_tomor



6.1. ábra. Binfa inorder kiirási iránya

Először létrehozzuk magat a binfa strukturát, a magot. Létrehozunk egy binfa típust, ebben deklarálunk egy 'int' változót, amiben a beolvasott adatokat tárljuk. Majd létrehozunk két mutatót, amivel a fa gyerekeire mutatunk. Illetve a strukturán kívül még létrehozzuk a binfa típusra mutató mutatót.

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

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


BINFA, BINFA_PTR;
```

Ebbena függvényben memóriát foglalunk le a binfa mutatónknak. Mikor új elemet hozunk létre, memóriát szabadítunk fel, illetve ha hiba történik, kilépünk.

```
BINFA_PTR uj_elem()
{
  BINFA_PTR p;

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

Deklarálunk később felhasználandó függvényeket. Egyik kiirja a fát, másik memóriát szabadít majd fel.

```
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)
```

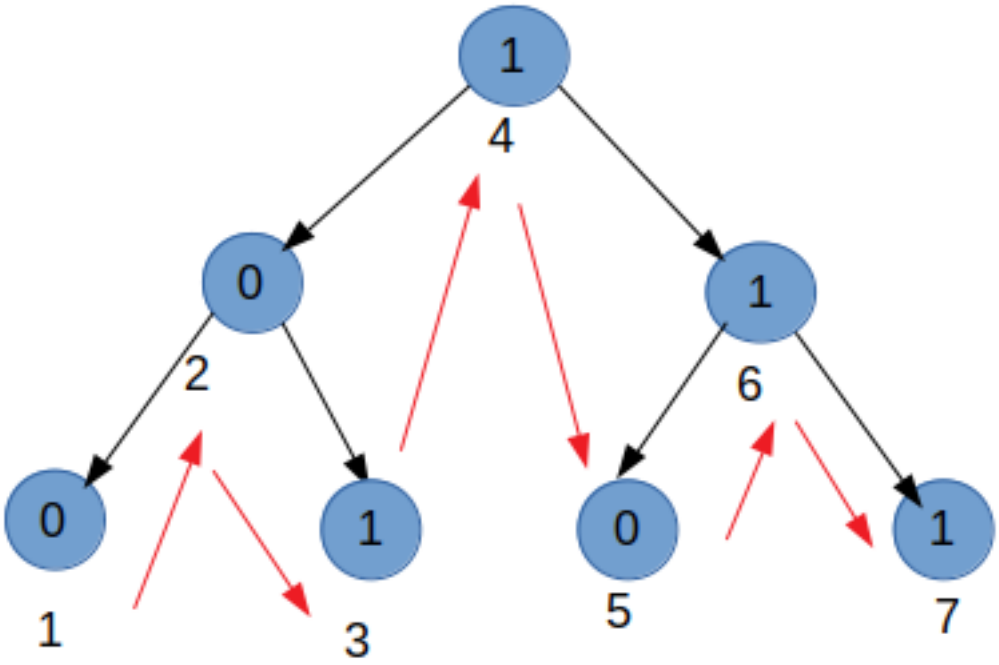
```
    {
      ++melyseg;
      if (melyseg > max_melyseg)
  max_melyseg = melyseg;
      kiir (elem->jobb_egy);
      for (int i = 0; i < melyseg; ++i)
  printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
          ,
        melyseg);
      kiir (elem->bal_nulla);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

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

```
rake@rake-desktop:~/BHAKSZ/Welch$ gcc binfa.c -o b
rake@rake-desktop:~/BHAKSZ/Welch$ ./b < test.txt
01010111110101010010101011011

---------1(3)
------1(2)
------------1(4)
---------0(3)
---/(1)
---------------1(5)
------------1(4)
---------1(3)
------------------1(6)
---------------1(5)
------------0(4)
------0(2)
melyseg=6
```

6.3. Fabejárás

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

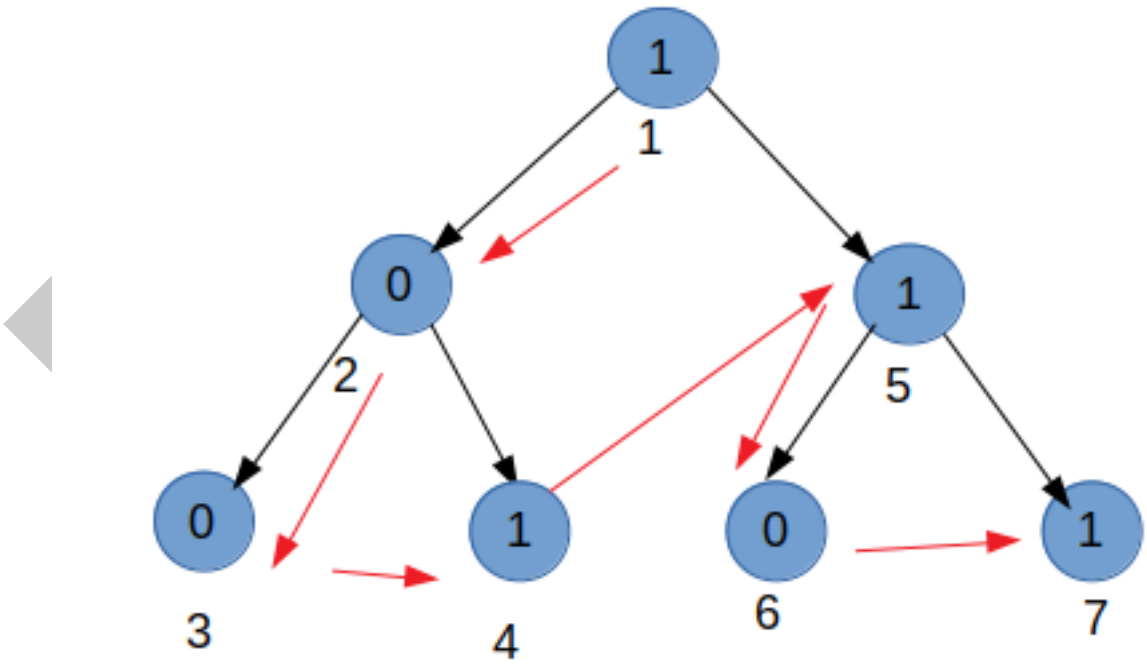


6.2. ábra. Binfa inorder kiírási iránya

```
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;
    }
}
```

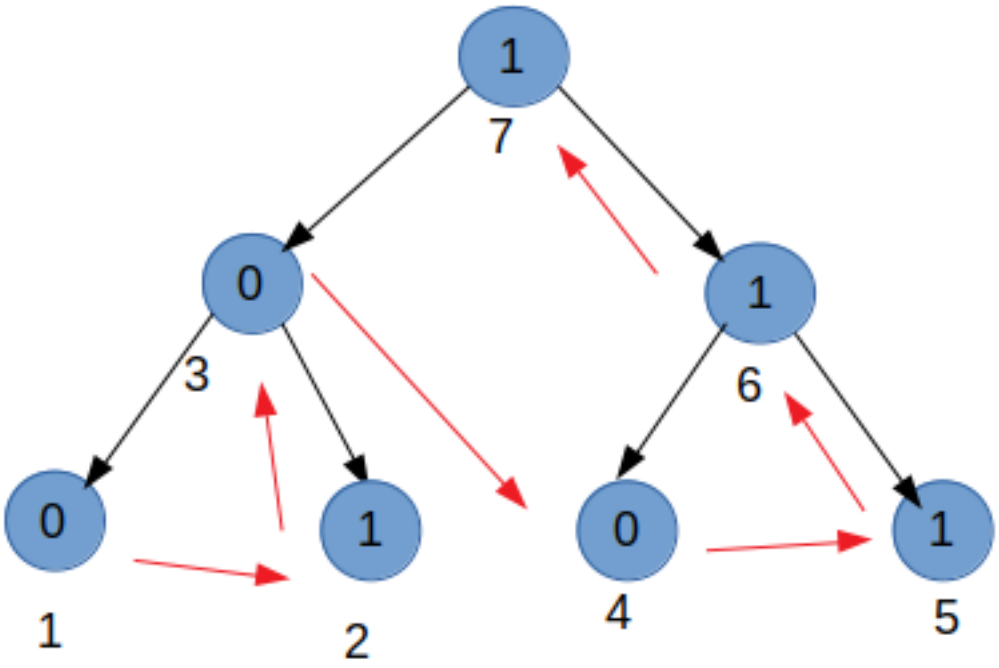
```
rake@rake-desktop:~/BHAKSZ/Welch$ gcc binfa.c -o b
rake@rake-desktop:~/BHAKSZ/Welch$ ./b < test.txt
01010111110101010010101011011

---------1(3)
------1(2)
------------1(4)
---------0(3)
---/(1)
---------------1(5)
------------1(4)
---------1(3)
-----------------1(6)
---------------1(5)
------------0(4)
------0(2)
melyseg=6
```



6.3. ábra. Binfa Preorder kiirási iránya

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

      --melyseg;
    }
}
```

```
rake@rake-desktop:~/BHAKSZ/Welch$ gcc binfapre.c -o bp
rake@rake-desktop:~/BHAKSZ/Welch$ ./bp < test.txt
01010111110101010010101011011

---------1(3)
------------1(4)
---------0(3)
------1(2)
---------------1(5)
------------1(4)
------------------1(6)
---------------1(5)
------------0(4)
---------1(3)
------0(2)
---/(1)
melyseg=6
```



6.4. ábra. Binfa Postorder kiirási iránya

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

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->jobb_egy);
      kiir (elem->bal_nulla);
      --melyseg;
    }
}
```

```
rake@rake-desktop:~/BHAKSZ/Welch$ gcc binfapost.c -o bpost
rake@rake-desktop:~/BHAKSZ/Welch$ ./bpost < test.txt
01010111110101010010101011011

---/(1)
------1(2)
---------1(3)
---------0(3)
------------1(4)
------0(2)
---------1(3)
------------1(4)
---------------1(5)
------------0(4)
---------------1(5)
-----------------1(6)
melyseg=6
```

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 forrása: https://progpater.blog.hu/2011/04/01/imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2 https://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg

Tehát ugyanezt kéne megoldanunk, egy Tree és egy Node osztály beágyazással. Egyelőre Node nem kap különös szerepet, csak beágyazzuk.

```
#include <iostream>
class LZWTree
{
  public:
    LZWTree (): fa(&gyoker){}
    ~LZWTree ()
    {
      szabadit (gyoker.egyesGyermek ());
      szabadit (gyoker.nullasGyermek ());
    }
  void operator<<(char b)
  {
    if (b == '0')
    {
      if (!fa->nullasGyermek ())
      {
        Node *uj = new Node ('0');
```

```
        fa->ujNullasGyermek (uj);
        fa = &gyoker;
      }
      else
      {
        fa = fa->nullasGyermek ();
      }
    }
    else
    {
    FT
      if (!fa->egyesGyermek ())
      {
        Node *uj = new Node ('1');
        fa->ujEgyesGyermek (uj);
        fa = &gyoker;
      }
      else
      {
      fa = fa->egyesGyermek ();
      }
    }
  }
  void kiir (void)
  {
    melyseg = 0;
    kiir (&gyoker);
  }
  void szabadit (void)
  {
    szabadit (gyoker.jobbEgy);
    szabadit (gyoker.balNulla);
  }
```

Ebben az osztályban a gyökér objektum, a fa mutató, kis cserebere történt. A fa mutató az éppen készülő fa azon csomópontjára/node-jára mutat ahol épp járunk.

```
private:
  class Node
  {
  public:
    Node (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
    ~Node () {};
    Node *nullasGyermek () {
    return balNulla;
  }
  Node *egyesGyermek ()
  {
    return jobbEgy;
  }
  void ujNullasGyermek (Node * gy)
```

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

  private:
    friend class LZWTree;
    char betu;
    Node *balNulla;
    Node *jobbEgy;
    Node (const Node &);
    Node & operator=(const Node &);
};
```

A Node osztály privátba került, védelmi szempontokból, így csak a fa osztályon belül érhető el. Konstruktorunknak nem adtunk paramétert egyenlőre, így a default müvelete, hogy '/' jellel hoz Node-ot létre.

```
Node gyoker;
Node *fa;
int melyseg;

LZWTree (const LZWTree &);
LZWTree & operator=(const LZWTree &);

void kiir (Node* elem)
{
  if (elem != NULL)
  {
    ++melyseg;
    kiir (elem->jobbEgy);
    for (int i = 0; i < melyseg; ++i)
      std::cout << "---";
    std::cout << elem->betu << "(" << melyseg - 1 << ")" << std:: ←-
      endl;
    kiir (elem->balNulla);
    --melyseg;
  }
}
void szabadit (Node * elem)
{
  if (elem != NULL)
  {
    szabadit (elem->jobbEgy);
    szabadit (elem->balNulla);
    delete elem;
  }
}
```

```
};
```

```
int main ()
{
  char b;
  LZWTree binFa;
  while (std::cin >> b)
{
  binFa << b;
}
binFa.kiir ();
binFa.szabadít ();

return 0;
}
```

6.5. Mutató a gyökér

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

Megoldás forrása:

Most annyiban változtatunk, hogy eddig a fa gyökerét objektumkét hoztuk létre, mostantól mutató lesz ez is, a fával együtt.

```
class LZWTree
{
public:
  LZWTree ()
  {
    gyoker = new Node();
    fa = gyoker;
  }

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

A konstruktorban a gyökeret is mutatóként írjuk fel, és mindenhol kitöröljük a gyökér elől a referencia jeleket. '&'

```
Node *gyoker;
Node *fa;
int melyseg;
```

6.6. Mozgató szemantika

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

Megoldás forrása: https://sourceforge.net/projects/udprog/

```
LZWTree& operator= (LZWTree& copy) //másoló értékadás
{
  szabadit(gyoker->egyesGyermek ()); //régi értéket törlöm
  szabadit(gyoker->nullasGyermek ());
  bejar(gyoker,copy.gyoker); //rekurzívan bejárom a fákat és átmásolom az ↩
     értékeket
  fa = gyoker; //mindkét fában visszaugrok a gyökérhez
  copy.fa = copy.gyoker;
}
  void bejar (Node * masolat, Node * eredeti) //rekurzív famásolás,
  másoló értékadáshoz
  {
    if (eredeti != NULL) //ha létezik a másolandó fa
    {
      if ( !eredeti->nullasGyermek() ) //ha nem létezik az eredeti ↩
         nullasgyermeke
      {
      masolat->ujNullasGyermek(NULL);
      }
      else //ha létezik az eredeti nullásgyermeke
      {
      //létrehozni a masolat nullasgyermeket és meghívni újra a bejart
        Node* uj = new Node ('0');
        masolat->ujNullasGyermek (uj);
        bejar(masolat->nullasGyermek(), eredeti->nullasGyermek());
      }
      if ( !eredeti->egyesGyermek() ) //ha nem létezik az eredeti ↩
         egyesgyermeke
      {
        masolat->ujEgyesGyermek(NULL);
      }
      else //ha létezik az eredeti egyesgyermeke
      {
      //létrehozni a masolat egyesgyermeket és meghívni újra a bejart
        Node *uj = new Node ('1');
        masolat->ujEgyesGyermek (uj);
        bejar(masolat->egyesGyermek(), eredeti->egyesGyermek());
      }
    }
    else //ha nem létezik a másolandó fa
    {
      masolat = NULL;
    }
```

```
}
```

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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. Robotpszichológia

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. Juhász István: Magas Szintű Programozási Nyelvek 1

A való világ túlságosan bonyolult ahhoz, hogy tökéletesen lemásoljuk a mechanikai működését, mégis a programozás valamelyest ezzel a céllal jött létre. Az emberi gondolkodást, modellezést próbálja utánozni, ezzel segítve az embereket a problémák megoldásában.

Először is pár alapfogalmat tisztáznunk. A számítógépek programozására kialakult nyelveket három szintre tagoljuk:

- -Gépi nyelv: Ez a legalsó nyelv, ez áll a legközelebb a géphez, közvetlenül ezt képes értelmezni a processzor, általában kettes számrendszerben íródok, de van pl. hexadecimális, azaz tizenhatos számrendszeren alapuló gépi kód is. Az ember és a gép között ez a nyelv a gép felé áll.
- -Assembly szintű nyelv: Az assembly egy program nyelv, amit csaknem minden programozási nyelv előállít végeredményként. Szimbolikus gépi kódnak is szokták nevezni.
- -Magas szintű nyelv: Ezzel foglalkozik a programozó, ez áll a legközelebb az emberi nyelvhez, a könnyű érthetőség érdekében. A magas szintű programozási nyelveket át kell alakítanunk valamilyen módon, pl. interpreteres, azaz értelmező móddal, vagy fordítóprogram segítségével. A fordítóprogram gépi kódot állít elő.

A Fordító Működése: Ahhoz hogy lefordítsa a program a forráskódot, a következő lépéseket hajtja végre a fordítóprogram:

- -Lexikális elemzés: A lexikális elemzés során a forrásszöveget feldarabolja lexikális egységekre.
- -Szintaktikai elemzés: A szintaktikai elemzés folyamán ellenőrzi, hogy teljesülnek-e az adott nyelv szintaktikai szabályai.
- -Szemantikai elemzés
- -Kódgenerálás

Minden nyelv rendelkezik saját szabályokkal, ami természetesen változhat. Ezek a szabályok affajta használati utasítás, korlátozás, mit tehetünk és hogyan. „C” esetében felhozhatjuk példaként a C89 és a C99 szabványt. Sokféle fordító program és forráskód szerkesztő létezik, úgynevezett IDE-k, úgymint Code::Blocks, Visual Studio, NetBeans.

Utasításokból áll az algoritmus, azokat lépésenként végrehajtja, illetve ezek alapján áll össze a tárgykód. Utasításokat két részre oszthatjuk:

- Deklarációs utasítások
- Végrehajtandó utasítások

10.2. Kerninghan & Richie

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

Minden programozási nyelvben közös, hogy változók és állandók alkotják a programban feldolgozott alapvető adatobjektumokat. Deklaráláskor szükséges megadni a típusukat, nevüket, és általában ezeknek már létrehozásukkor értéket utalnak. Operátorokat arra használjuk, hogy megadjuk a gépnek mit is csináljon az adattal. A típus megadása fontos lépés, ez határozza meg milyen értéket tartalmaz, ad vissza, illetve milyeneket kaphat.

Most a „C” nyelvvel fogunk foglalkozni.

Az ANSI szabvánnyal kibővült a lehetőség a programozók számára. Minősítők kerültek bele, melyek teret adtak a programozóknak hogy más hosszúságú egészekkel is dolgozhassanak. Ilyen minősítők pl. a „signed” és „unsigned” forma, amely az előjel meglétét vagy hiányát mutatja.

Bevezették a

```
long double
```

adattípust, ezzel együtt a „short” előjelet is. A „short” 16bites határral bír, a „long” viszont 32bites.

C-vel foglalkozva kevés alapvető adattípust használunk:

```
char
  //egyetlen bájt, a gépi karakterkészlet egy elemét tárolja


int
  //egész szám, mérete általában a befogadó számítógép egészek ↩
     ábrázolásához használt mérete
```

```
float
  //egyszeres pontosságú lebegőpontos szám



double
  //kétszeres pontosságú lebegőpontos szám
```

Ezekehez társulnak minősíthető specifikáció, pl 'short' és 'long'

```
short int valami;
long int nagyValami;
```

10.3. BME Szoftverfejlesztás C++ Nyelven

A „C++" a „C" nyelv család tagja, sok mindent örökölt tőle, legtöbbet említett örökölt tulajdonsága a „hardverközelség". A „C++" egy Objektumorientált nyelv, a mai világban egyik legelterjedtebb nyelv, alapja sok operációs és embedded rendszernek.

Alapjai a „C"-re épül, azonban vannak lényeges különbségek a két nyelv között, példaként véve egy „C++" függvényt.

```
Void f(){}
```

A függvénynek nincs paramétere, azaz nem lehet paraméterrel hívni. Azonban ha „C"-ben is ezt szeretnénk elérni, módosítanunk kell a függvényt, és paraméterként „..." kell írnunk.

```
Void f(...){}
```

Új típus is felüti a fejét, a 'bool'. Ennek értéke True vagy False lehet. Ez segíti az olvashatóságot, azonban átalakíthatjuk 'int'-re is, 0 vagy 1 értékké alakul.

'C++' teret adott egy új lehetőségnek: 'függvény túlterhelés' lehetségessé vált. 'C++'-ban beazonosítjuk a függvényeket a nevük és argumentum listájuk alapján, ezáltal megnyílik a lehetőség hogy ugyanazon névvel ellátott, viszont különböző argumentum listával rendelkező függvényeket képezzünk, csakis a visszatérési érték, ami nem változhat.

III. rész

Második felvonás

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

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

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

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