Helló, Arroway!

Balla Csaba

2020-02-22

> „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

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.

> **Tip**
>
> Ez esetben forgasd a https://tdg.docbook.org/tdg/5.1/ könyvet, a végén találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

Vízió

Mi a programozás?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk? Számítógép-programozás (vagy egyszerűen programozás) egy vagy több absztrakt algoritmus megvalósítását jelenti egy bizonyos programozási nyelven. A programozásban megtaláljuk a művészet, a tudomány, a matematika és a mérnöki tudomány elemeit. A programok maga az az utasítás vagy utasítás sorozat ami vezérel egy számítógépet. Valamint segíti a felhasználót(Szoftverek) a számítógép netaláltán más gépek használatában, vagy elszörakoztatja(Játékok) azt különböző szórakoztató programokkal. Ennek a skálája végtelennek tekinthető, hogy miket lehet létrehozni ebben a szakmában. Az írásuknak van egy úgymond részei:

- 1. A megoldandó probléma meghatározása, felmérése a majdani felhasználók igényei alapján, specifikáció készítése
- 2. Valamely programtervezési módszerrel a programszerkezet megalkotása és a használandó eszközök kiválasztása. (Hardver platform, nyelvek, adatok stb. . .)
- 3. Forrásprogram elkészítése (kódolás)
- 4. A kész program tesztelése
- 5. Dokumentáció készítése, mely tartalmazza a szoftvertervezés fázisaiban keletkezett adatokat (felhasználói leírás, igényfelmérés, programtervek, algoritmusok, forráskód, tesztelési jegyzőkönyvek stb.), fő célja a szoftver későbbi fejlesztésének elősegítése.

Milyen doksikat olvassak el?

- Kezd ezzel: http://esr.fsf.hu/hacker-howto.html!
- Olvasgasd aztán a kézikönyv lapjait, kezd a `man man` parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a `man 3 sleep` lapot
- C kapcsán a KERNIGHANRITCHIE könyv adott részei.
- C++ kapcsán a BMECPP könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a The GNU C Reference Manual, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf

- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a BMECPP könyv - 20 oldalas gyorstalpaló részét.

Milyen filmeket nézzek meg?

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

  Tip

  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.

Helló, Turing!

A kódok megtekínthetőek githubon. Link: https://github.com/Csabatron99/BHAX-konyv-b-csaba/tree/master/Source%20Codes

Végtelen ciklus

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

Megoldás videó(nbatfai): https://youtu.be/lvmi6tyz-nI

Megoldás forrása: `bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c`, `bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-w.c`.

Számos módon hozhatunk és hozunk létre végtelen ciklusokat. Vannak esetek, amikor ez a célunk, például egy szerverfolyamat fusson folyamatosan és van amikor egy bug, mert ott lesz végtelen ciklus, ahol nem akartunk. Saját példáinkban ilyen amikor a PageRank algoritmus rázza az 1 liter vizet az internetben, de az iteráció csak nem akar konvergálni...

Egy mag 100 százalékban:

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

  return 0;
}
```

vagy az olvashatóbb, de a programozók és fordítók (szabványok) között kevésbé hordozható

```
int
#include <stdbool.h>
main ()
{
  while(true);

  return 0;
}
```

Azért érdemes a `for(;;)` hagyományos formát használni, mert ez minden C szabvánnyal lefordul, másrészt a többi programozó azonnal látja, hogy az a végtelen ciklus szándékunk szerint végtelen és nem szoftverhiba. Mert ugye, ha a `while`-al trükközünk egy nem triviális `1` vagy `true` feltétellel, akkor ott egy másik, a forrást olvasó programozó nem látja azonnal a szándékunkat.

Egyébként a fordító a `for`-os és `while`-os ciklusból ugyanazt az assembly kódot fordítja:

```
$ gcc -S -o infty-f.S infty-f.c
$ gcc -S -o infty-w.S infty-w.c
$ diff infty-w.S infty-f.S
1c1
<   .file   "infty-w.c"
---
>   .file   "infty-f.c"
```

Egy mag 0 százalékban:

```c
#include <unistd.h>
int
main ()
{
  for (;;)
    sleep(1);

  return 0;
}
```

Minden mag 100 százalékban:

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

A `gcc [filename].c -o [filename] -fopenmp` parancssorral készítve a futtathatót, majd futtatva, közben egy másik terminálban a `top` parancsot kiadva tanulmányozzuk, mennyi CPU-t használunk:

```
top - 20:09:06 up  3:35,  1 user,  load average: 5,68, 2,91, 1,38
Tasks: 329 total,   2 running, 256 sleeping,   0 stopped,   1 zombie
%Cpu0 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 5850 batfai    20   0   68360    932    836 R 798,3  0,0   8:14.23 infty-f
```

Tip

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

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.

Ez a feladat a megállási probléma bemutatására szolgál. A megálláso problémea abból áll, hogy el lehet-e döntenti egy programról adott bemenet esetén, hogy végtelen ciklusba kerül-e. Alan Turin 1963-ban bizonyította be, hogy nem lehetséges olyan álltalános algoritmust írni, amely minden program-bemenet párról megmondja, hogy végtelen ciklusba kerül-e.

Változók értékének felcserélése

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

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

Megoldás forrása:

Változócserére több lehetőség is van, 4 féle alap módszer van. Ezek lehetnek:

- Segédváltozóval

- Kivonással / Összeadással
- XOR logikai módszer

Az első példára a segédváltozónak van jó és rossz oldala is. A jó oldala, hogy ez a legkönyebb mindközül logikailag kitalálni és programba illetve pszeudokódba írni. A rossz oldala pedig a segédváltozó miatt van. A plusz változó miatt több helyet kell lefoglalni. C beli leírása a következő:

```
#include <stdio.h>
int main()
{
    int tmp = 0,a =1,b = 2;
    tmp = a;
    a = b;
    b = tmp;
}
```

A második példához a matematikailag ismert osszeadási és kivonási szabályokból ismeretes elemeket használjuk fel. Az egyik a következő: a = a + b b = a - b a = a - b Lássuk végrehajtva. Az első lépésben a = 20 + 10 = 30 lesz. Utána b = 30 - 10 = 20, vagyis b-be máris bekerült a eredeti értéke, a harmadik lépésben pedig a = 30 - 20 = 10, puff, helyet cserélt a kettő! A másik ilyen módszer a következő: a = a - b b = a + b a = b - a Ezt is lássuk végrehajtva. Az első lépésben a = 20 - 10 = 10 lesz. Utána b = 10 + 10 = 20, vagyis b-be máris bekerült a eredeti értéke, a harmadik lépésben pedig a = 20 - 10 = 10, és így helyet cserélt a kettő! C beli leírása a következő:

```
#include <stdio.h>
int main()
{
    int a =1,b = 2;
    a = a + b;
  b = a - b;
  a = a - b;
}
```

```
#include <stdio.h>
int main()
{
    int a =1,b = 2;
    a = a - bé
  b = a + bé
  a = b - a;
```

```
}
```

Az utoló módszerről pedig kicsit többet kell beszéljünk mivel az már egy kicsivel bonyolultabb, helytelen működésű és lassab. Először nézzük, hogy mi is az az XOR csere: Annak idején, amikor még mindenki Assembly nyelvű programokat írt, rengeteg apró optimalizálást kézzel végeztek el a programozók. Például egy regiszter nullázása helyett: movl $0, eax (C-ben: eax = 0), inkább saját magával XOR kapcsolatba hozták azt: xorl eax, eax (eax ^= eax). Mert ez kevesebb bájtot foglalt, nem kellett a 0 számot eltárolni hozzá a memóriában. Akkoriban jöttek rá arra is, hogy két regiszter értékét szintén meg lehet cserélni az xor gépi utasítással. A módszer előnye, hogy nincsen hozzá szükség segédregiszterre. Eddig minden rendben is lenne. A baj csak az, hogy a '70-es években kitalált, az akkori szemléletet tükröző módszer annyira beszivárgott a köztudatba, hogy egyesek manapság is használják, magas szintű programozási nyelveken. A XXI. században, amikor a fordítóprogramok legtöbbje gyorsabb kódot tud generálni, mint amilyen Assembly kódot az emberek írnának. Magával a módszerrel az a baj, hogy sok programozó nem értené mit akarunk ezzel a programrésszel valamint sokkal nehezebb megérteni, ezért nem terjedt el és nem is szokták használni viszont úgy éreztem jólenne bemutatni mivel ha valamilyen oknál fogva talalkoznál vele akkor tudd, hogy mi is ez pontosan és egy érdekességnek is elkönyvelhető. C beli leírása pedig a következő:

```c
#include <stdio.h>
int main()
{
    int a =1,b = 2;
  a ^= b;
    b ^= a;
    a ^= b;
}
```

A tanulságnak levonhatjuk azt, hogy a segédváltozó segítségével a legkönyebb viszont a második módszerrel nem kell külön lefoglalni helyet a segédváltozónak. Az XOR módszer pedig nem ajánlott használni a fenti okokból kifolyólag. Így a második módszert ajánlom használatra változócsere esetén.

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 (nbatfai): https://bhaxor.blog.hu/2018/08/28/labdapattogas

Megoldás forrása:

Az if es megoldás C kódja:

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

int main(void)
{
    WINDOW *window;
    window = initscr();
    int x=0, y=0;
    int xnow=1, ynow=1;
    int mx,my;
    for(;;)
    {
        getmaxyx(window,my,mx);
        mvprintw(y,x,"O");
        refresh();
        usleep(100000);
        clear();
        x=x+xnow;
        y=y+ynow;
        if (x>=mx-1) //Elerte-e a jobb oldalt?
        {
            xnow = xnow*-1;
        }
        if (x<=0) //Elerte-e a bal oldalt?
        {
            xnow = xnow*-1;
        }
        if (y<=0) //Elerte-e a tetejet?
        {
            ynow=ynow*-1;
        }
        if (y>=my-1) //Elerte-e az aljat?
        {
            ynow = ynow*-1;
        }
    }
    return 0;
}
```

Az if nélküli megoldás C kódja:

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

int main(void)
{
    WINDOW *window;
    window = initscr();
    int xj=0, xk=0, yj=0, yk=0;
    int mx,my;
    nodelay(window, true);
    getmaxyx(window,my,mx);
    my=my*2;mx=mx*2;
    for(;;)
    {
        xj = (xj-1)%mx;
        xk = (xk+1)%mx;
        yj = (yj-1)%my;
        yk = (yk+1)%my;
        clear();
        mvprintw(abs((yj+(my-yk)) / 2),abs((xj+(mx-xk)) / 2), "O");
        refresh();
        usleep(100000);
    }
    return 0;
}
```

A logikai változós megoldás könyen megoldható viszont enélkül eléggé megnehezítette a dolgomat. Végül sikerült megoldanom néhány érték atfésüléssel ahogy az a fenti példában is látható. Részletesebb magyarázatért megtekintheted a videót.

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(nbatfai): https://youtu.be/9KnMqrkj_kU, https://youtu.be/KRZlt1ZJ3qk, .

Megoldás forrása(nbatfai): `bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bog`

A megoldás C++ kódja:

```
#include <iostream>
int main()
{
    int num = -1;
    int count = 0;
    unsigned int num_copy = (unsigned int)num;
    while(num_copy >>=1)
    {
        count++;
    }
    printf("%d",(count+1));
    return 0;
}
```

Először elég sokáig gondolkodtam, hogy a feladatban pontosan mit is kell csinálni mi is az amit kér. Egy barátom segítségével aki felsőbbéves rájöttem, innestől már elég könyen ment a feladat megoldása és a meglévő algoritmus átírása.

Helló, Google!

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

Megoldás videó:

Megoldás forrása:

A megoldás C kódja:

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

void kiir (double tomb[], int db)
{
    int i;
    for (i=0; i<db; i++)
    printf("%lf\n", tomb[i]);
}
double tavolsag(double pagerank[],double pagerank_temp[],int db)
{
    double tav = 0.0;
    int i;
    for(i=0;i<db;i++)
        tav+=(pagerank[i] - pagerank_temp[i])*(pagerank[i] - pagerank_temp[i]);
```

```
    sqrt(tav);
    return tav;
}
int main(void)
{
    double L[4][4] = {
        {0.0, 0.0, 1.0 / 3.0, 0.0},
        {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
        {0.0, 1.0 / 2.0, 0.0, 0.0},
        {0.0, 0.0, 1.0 / 3.0, 0.0}
    };
    double PR[4] = {0.0, 0.0, 0.0, 0.0};
    double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};
    for (;;)
    {
        for(int i=0;i<4;i++)
        {
            PR[i]=0;
            for (int j=0;j<4;j++)
            {
                 PR[i] = PR[i]+(PRv[j]*L[i][j]);
            }
        }
        if ( tavolsag(PR,PRv, 4) < 0.00000001)
            break;
        for(int i=0;i<4;i++)
            PRv[i]=PR[i];
    }
    kiir (PR,4);
    return 0;
}
```

Mivel pagerank ot már sikerült megírjam egyszer C++-ban így az átírás könyedén megoldható volt.

A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

```
kis_szam=10000000
kis=sample(1:3,kis_szam,replace=T)
jatekos=sample(1:3,kis_szam,replace=T)
vezeto=vector(length=kis_szam)
for(i in 1:kis_szam)
{
    if(kis[i]==jatekos[i])
    {
        mibol=setdiff(c(1,2,3),kis[i])
    }
    else
    {
        mibol=setdiff(c(1,2,3),c(kis[i],jatekos[i]))
    }
    vezeto[i]=mibol[sample(1:length(mibol),1)]
}
nemvaltnyer=which(kis==jatekos)
valt=vector(length=kis_szam)
for(i in 1:kis_szam)
{
    holvaltoztat=setdiff(c(1,2,3),c(vezeto[i],jatekos[i]))
    valt[i]=holvaltoztat[sample(1:length(holvaltoztat),1)]
}
valtnyer=which(kis==valt)
sprintf("Kiserletek szama: %i",kis_szam)
length(nemvaltnyer)
length(valtnyer)
length(nemvaltnyer)/length(valtnyer)
length(nemvaltnyer)+length(valtnyer)
```

Őszintén megvallva életemben nem hallodtam a Monty Hall problémáról eddig. Viszont informálódva Bátfai anyagaiból hamar megtudtam érteni a lényegét és összerakni a szimulációt. Igaz magáról az R nyelvről sem hallodtam de ahogy az alapjait kezdtem nézni be kell valjam nem nehéz nyelv valamitn nem valami elterjedt a használata.

100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó(nbatfai): https://youtu.be/xbYhp9G6VqQ

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

A természetes számok építőelemei a prímszámok. Abban az értelemben, hogy minden természetes szám előállítható prímszámok szorzataként. Például 12=2*2*3, vagy például 33=3*11.

Prímszám az a természetes szám, amely csak önmagával és eggyel osztható. Eukleidész görög matematikus már Krisztus előtt tudta, hogy végtelen sok prímszám van, de ma sem tudja senki, hogy végtelen sok ikerprím van-e. Két prím ikerprím, ha különbségük 2.

Két egymást követő páratlan prím között a legkisebb távolság a 2, a legnagyobb távolság viszont bármilyen nagy lehet! Ez utóbbit könnyű bebizonyítani. Legyen n egy tetszőlegesen nagy szám. Akkor szorozzuk össze n+1-ig a számokat, azaz számoljuk ki az 1*2*3*... *(n-1)*n*(n+1) szorzatot, aminek a neve (n+1) faktoriális, jele (n+1)!.

Majd vizsgáljuk meg az a sorozatot:

(n+1)!+2, (n+1)!+3,... , (n+1)!+n, (n+1)!+ (n+1) ez n db egymást követő azám, ezekre (a jól ismert bizonyítás szerint) rendre igaz, hogy

- (n+1)!+2=1*2*3*... *(n-1)*n*(n+1)+2, azaz 2*valamennyi+2, 2 többszöröse, így ami osztható kettővel
- (n+1)!+3=1*2*3*... *(n-1)*n*(n+1)+3, azaz 3*valamennyi+3, ami osztható hárommal
- ...
- (n+1)!+(n-1)=1*2*3*... *(n-1)*n*(n+1)+(n-1), azaz (n-1)*valamennyi+(n-1), ami osztható (n-1)-el
- (n+1)!+n=1*2*3*... *(n-1)*n*(n+1)+n, azaz n*valamennyi+n-, ami osztható n-el
- (n+1)!+(n+1)=1*2*3*... *(n-1)*n*(n+1)+(n+1), azaz (n+1)*valamennyi+(n+1), ami osztható (n+1)-el

tehát ebben a sorozatban egy prim nincs, akkor a (n+1)!+2-nél kisebb első prim és a (n+1)!+ (n+1)-nél nagyobb első prim között a távolság legalább n.

Az ikerprímszám sejtés azzal foglalkozik, amikor a prímek közötti távolság 2. Azt mondja, hogy az egymástól 2 távolságra lévő prímek végtelen sokan vannak.

A Brun tétel azt mondja, hogy az ikerprímszámok reciprokaiból képzett sor összege, azaz a (1/3+1/5)+ (1/5+1/7)+ (1/11+1/13)+... véges vagy végtelen sor konvergens, ami azt jelenti, hogy ezek a törtek összeadva egy határt adnak ki pontosan vagy azt át nem lépve növekednek, ami határ számot B_2 Brun konstansnak neveznek. Tehát ez nem dönti el a több ezer éve nyitott kérdést, hogy az ikerprímszámok halmaza végtelen-e? Hiszen ha véges sok van és ezek reciprokait összeadjuk, akkor ugyanúgy nem lépjük át a B_2 Brun konstans értékét, mintha végtelen sok lenne, de ezek már csak olyan csökkenő mértékben

járulnának hozzá a végtelen sor összegéhez, hogy így sem lépnék át a Brun konstans értékét.

Ebben a példában egy olyan programot készítettünk, amely közelíteni próbálja a Brun konstans értékét. A repó `bhax/attention_raising/Primek_R/stp.r` mevű állománya kiszámolja az ikerprímeket, összegzi a reciprokaikat és vizualizálja a kapott részeredményt.

```
#   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/>

library(matlab)

stp <- function(x){

    primek = primes(x)
    differencia = primek[2:length(primes)]-primek[1:length(primes)-1]
    index = which(differencia==2)
    v1primek = primek[idx]
    v2primek = primek[idx]+2
    v1pv2 = 1/v1primek+1/v2primek
    return(sum(v1pv2))
}

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

Soronként értelemezzük ezt a programot:

```
 primek = primes(13)
```

Kiszámolja a megadott számig a prímeket.

```
> primek=primes(13)
> primek
[1]  2  3  5  7 11 13
```

```
 diff = primek[2:length(primes)]-primek[1:length(primek)-1]
```

```
> diff = primek[2:length(primes)]-primek[1:length(primek)-1]
> differencia
[1] 1 2 2 4 2
```

Az egymást követő prímek különbségét képzi, tehát 3-2, 5-3, 7-5, 11-7, 13-11.

```
index = which(diff==2)
```

```
> index = which(diff==2)
> index
[1] 2 3 5
```

Megnézi a `differencia`-ban, hogy melyiknél lett kettő az eredmény, mert azok az ikerprím párok, ahol ez igaz. Ez a `differencia`-ban lévő 3-2, 5-3, 7-5, 11-7, 13-11 különbségek közül ez a 2., 3. és 5. indexűre teljesül.

```
v1primek = primek[index]
```

Kivette a primes-ból a párok első tagját.

```
v2primek = primek[index]+2
```

A párok második tagját az első tagok kettő hozzáadásával képezzük.

```
rt1plust2 = 1/t1primes+1/t2primes
```

Az 1/v1primek a v1primek 3,5,11 értékéből az alábbi reciprokokat képzi:

```
> 1/v1primek
[1] 0.33333333 0.20000000 0.09090909
```

Az 1/v2primek a v2primek 5,7,13 értékéből az alábbi reciprokokat képzi:

```
> 1/v2primek
```

```
[1] 0.20000000 0.14285714 0.07692308
```

Az 1/v1primek + 1/v2primek pedig ezeket a törteket rendre összeadja.

```
> 1/t1primes+1/t2primes
[1] 0.5333333 0.3428571 0.1678322
```

Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvénnyel.

```
sum(rt1plust2)
```

```
>   sum(rt1plust2)
[1] 1.044023
```

A következő ábra azt mutatja, hogy a szumma értéke, hogyan nő, egy határértékhez tart, a B_2 Brun konstanshoz. Ezt ezzel a csipettel rajzoltuk ki, ahol először a fenti számítást 13-ig végezzük, majd 10013, majd 20013-ig, egészen 990013-ig, azaz közel 1 millióig. Vegyük észre, hogy az ábra első köre, a 13 értékhez tartozó 1.044023.

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

Tip

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

Helló, Chomsky!

Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

A megoldáshoz tartozik egy kód, amit konyen lehet vizualizálni és futtatni a kobvetkező linken: https://turingmachine.io/ A kód pedig a követekező:

```
            %{
#https://turingmachine.io/
input: '12'
```

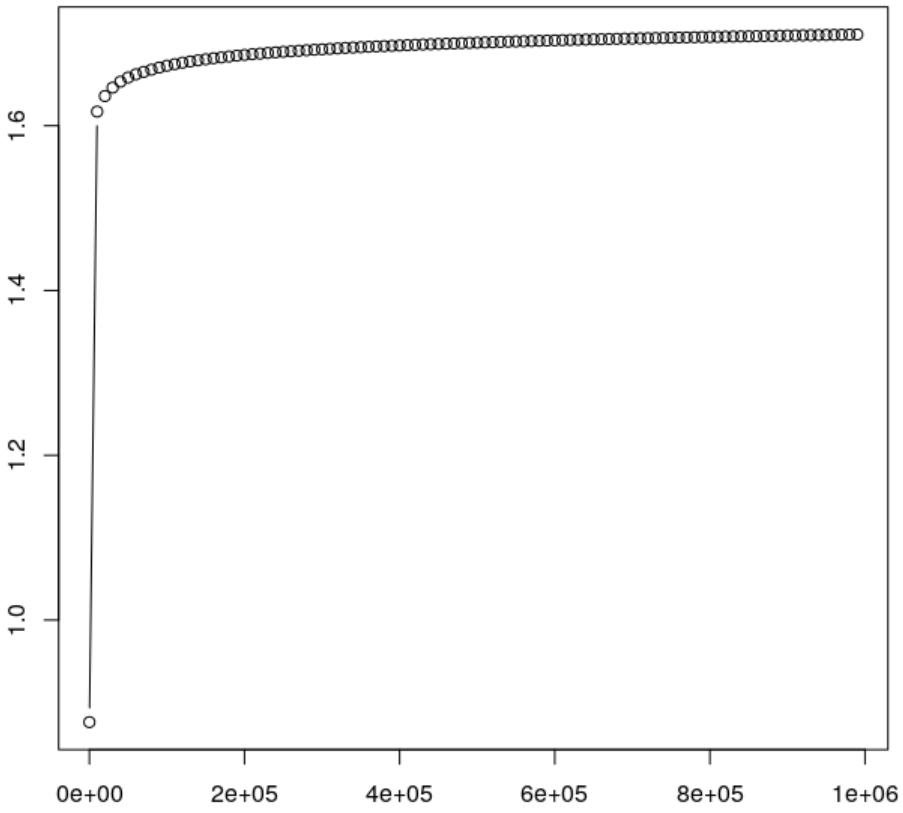

Figure 1: A B_2 konstans közelítése

```
blank: ' '
start state: GoRight
table:
  # scan to the rightmost digit
  GoRight:
    [0,1,2,3,4,5,6,7,8,9]: R
    ' '  : {L: Decrease}

  Decrease:
    ' ': {R: done}
    0: {L: 0ism}
    1: {write: 0, R: ToUniStartP}
    2: {write: 1, R: ToUniStartP}
    3: {write: 2, R: ToUniStartP}
    4: {write: 3, R: ToUniStartP}
    5: {write: 4, R: ToUniStartP}
    6: {write: 5, R: ToUniStartP}
    7: {write: 6, R: ToUniStartP}
    8: {write: 7, R: ToUniStartP}
    9: {write: 8, R: ToUniStartP}

  ToUniStartP:
    [0,1,2,3,4,5,6,7,8,9]: R
    ' '  : {R: ToUniEndP}

  ToUniEndP:
    1: R
    ' ': {write: 1, L: ToDecimalStartP}

  ToDecimalStartP:
    1: L
    ' ': {L: Decrease}

  0ism:
    0: {L: 0ism}
    ' ': {R: done}
    1: {write: 0, R: TENmONE}
    2: {write: 1, R: TENmONE}
    3: {write: 2, R: TENmONE}
    4: {write: 3, R: TENmONE}
    5: {write: 4, R: TENmONE}
    6: {write: 5, R: TENmONE}
    7: {write: 6, R: TENmONE}
    8: {write: 7, R: TENmONE}
    9: {write: 8, R: TENmONE}
```

```
  TENmONE:
    0: {write: 9, R: TENmONE}
    ' ': {L: ToUniStartP}
  done:
}
```

A turing gép vizualizációja: Decimal to Uniral turing gép

Megoldás videó: (Hamarosan)

Megoldás forrása: (hamarosan)

Ilyen tárgyunk ahol ezzel részletesebben foglalkoztunk volna még nem volt, de a Turing-gép elég egyszerűnek bizonyult ahhoz hogy előzetes tudás nélkül egy online eszköz használatával megírjam a programot.

Megéri belekezdeni egy olyan feladatba is amiről előzetesen semmit sem tudunk, mert lehet könnyen megérthető és egyszerűbb, mint gondolnánk. Előre eléggé szkeptikus voltam de kicsi áskálás után már világossá és könyuvé vállt. A tanulást sok online eszköz is elősegíti. Én ezt az állapotátmenet-gráfot a turingmachine.io segítségével írtam, ami egy egyszerű basic-szerű scriptnyelvből tud turing-gépeket generálni és szép grafikákat is csinálni hozzá.

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:

1.		S	→	a	B	C	
2.		S	→	a	S	B	C
3.	C	B	→	C	Z		
4.	C	Z	→	W	Z		
5.	W	Z	→	W	C		
6.	W	C	→	B	C		
7.	a	B	→	a	b		
8.	b	B	→	b	b		
9.	b	C	→	b	c		
10.	c	C	→	c	c		

```
1.        S   →   a   B   C
2.        S   →   a   S   B   C
3.    C   B   →   C   Z
4.    C   Z   →   W   Z
5.    W   Z   →   W   C
6.    W   C   →   B   C
7.    a   B   →   a   b
8.    b   B   →   b   b
9.    b   C   →   b   c
10.   c   C   →   c   c
11.       S   →   S
```

Tanulságok, tapasztalatok, magyarázat...

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:

Tanulságok, tapasztalatok, magyarázat...

```
<utasítás>  ::= <címkézett> | <kifejezés> | <összetett> | <kiválasztó> | <iterációs> | <vezé

<címkézett> ::= azonosító : <utasítás>
                        case <állandó_kif> : <utasítás>
                        default : <utasítás>

<kifejezés> ::= <kifejezés_kif> ;

<összetett> ::= {deklarációs_lista utasítás_lista}

    <deklarációs_lista> ::= deklaráció
                                        deklarációs_lista deklaráció
    <utasítás_lista>    ::= utasítás
                                      utasítás_lista utasítás

<kiválasztó>    ::= if ( <kifejezés_kif> ) <utasítás>
                           if ( <kifejezés_kif> ) <utasítás> else <utasítás>
```

```
                              switch ( kifeje
                              zés_kif ) <utasítás>

<iterációs> ::= while ( <kifejezés_kif> ) <utasítás>
                        do  <utasítás> while ( <kifejezés_kif> ) ;
                        for (<kifejezés_kif> ; <kifejezés_kif> ; <kifejezés_kif> ) <utasítás

<vezérlésátadó> ::= goto <azonosító> ;
                                continue ;
                                break ;
                                return <kifejezés_kif> ;
```

Valamint a C kód:

```c
#include "stdio.h"

int main()
{
    //c99+
    for (int i = 0; i<5; i++)
    {
        long long int a = 5;
    }

    char* a;

    //c11-
    gets(a);

    return 0;}
```

Saját lexikális elemző

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

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

Megoldás forrása: `bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumbe`

```
        Imports System.Console
```

```
Imports System.Text.RegularExpressions

Module MLexer
    Sub ChomskyLexer()
        Do
            Dim be As String = ReadLine()
            Dim pattern As String = "[0-9]*(\.[0-9]+)?"
            Dim realnums As Integer = 0

            be = Regex.Replace(be, pattern, Function(match As Match)
                                                If match.Length > 0 Then
                                                    realnums += 1
                                                    Return "<real num: " + Conve
                                                End If
                                                Return match.Value
                                            End Function)

            WriteLine(be)
            WriteLine(realnums)
        Loop
    End Sub
End Module
```

A regex alkalmazása különböző dolgok parseolására. A lexerek definíciója és szerepe a programozásban. A VB regex integrációja. Az emberi nyelv szintaktikájának nem-matematikaisága, nem szabályossága. Például: mi a valós szám? -Kell a tizedesvessző előtt szám vagy nem? -Tizedes vessző vagy pont? -Tagolás ezresével? -Ezrenkénti tagolás szóközzel vagy más jellel? -Exponenciális reprezentáció? Mivel működnek a különböző programnyelveket lintelő és parseoló programok, algoritmusok.

Leetspeak

Lexelj össze egy l33t ciphert!

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

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

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

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

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

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

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

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

```
  };

%}
%%
.   {

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

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

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

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

          found = 1;
          break;
        }

      }

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

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

Tanulságok, tapasztalatok, magyarázat...

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.)

> **Caution**
>
> 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. `for(i=0; i<5; ++i)`

   Először i-t 0-val tesszük egyenlővé. Ezután megnézzük, hogy i kisebb-e mint 5. Ha igen, akkor lefuttatjuk a body-t majd inkrementáljuk i-t. Ha nem, akkor tovább haladunk a következő utasításhoz

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

   Először i-t 0-val tesszük egyenlővé. Ezután megnézzük, hogy i kisebb-e mint 5. Ha igen, akkor lefuttatjuk a body-t majd inkrementáljuk i-t. Ha nem, akkor tovább haladunk a következő utasításhoz

   A post-vagy pre-inkrementálás különbsége itt teljesen mindegy, mivel nem használjuk a visszadott értéket és a tévhittel ellenben ugyanannyi instrukciót generál (ugyanolyan "gyors") mindkettő.

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

   Először i-t 0-val tesszük egyenlővé. Ezután megnézzük, hogy i kisebb-e mint 5. Ha igen, akkor lefuttatjuk a body-t majd a tomb[i]-t egyenlővé tesszük i-vel, ezután inkrementáljuk i-t. Ha nem, akkor tovább haladunk a következő utasításhoz

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

   s-ből d-be n elemet másol.

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

   Unspecified behaviour. (Mert a függvényargumentumok kiértékelési sorrendje nincs meghatározva.) Természetes nyelven: Mindhárom függvényhívás kiértékelődik, de nem tudjuk milyen sorrendben értékelődnek ki az argumentumok.

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

Kiírja az f(a)-t integerként és az a-t integerként. Itt azért nincs baj mert a-t másoljuk mikor meghívjuk f-et, tehát a eredeti értéke nem változik.

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

Ha a-t módosítjuk f-ben akkor mivel nincs meghatározva a kiértékelési sorrend, ezért az a lehet a módosítatlan a is vagy a módosított is.

Megoldás forrása:

Megoldás videó:

Egy kis gyakorlás C ből. Valamint, hogy legyen fogalmunk mi mit csinál és hogyan. Megtudtam milyen jó dolgunk van manapság a c++ auto-val, ami kitalálja helyettünk a típust, valamint a template-ekkel amik segítségével nem is kell foglalkoznunk a funkciók pontos definíciójával.

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... First things first: Ragd be LAETX-be

Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

  decltype(2) a

- egészre mutató mutató

  std::add_pointer_t<std::add_pointer_t<decltype(3)>> a

- egész referenciája

  std::add_lvalue_reference_t<decltype(4)> a

- egészek tömbje

  int a[5];

- egészek tömbjének referenciája (nem az első elemé)

  int (&a)[64];

- egészre mutató mutatók tömbje

  int *a[64];

- egészre mutató mutatót visszaadó függvény

  int* f();

- egészre mutató mutatót visszaadó függvényre mutató mutató

  int* (*fn)();

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

  int (*fn(int))(int, int)

- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

  int (*(*fn)(int))(int, int)

Mit vezetnek be a programba a következő nevek?

- `int a;`

  egy egész

- `int *b = &a;`

  egész pointer

- `int &r = a;`

  egész referencia

- `int c[5];`

  egy egész tömb

- `int (&tr)[5] = c;`

  egy egész tömb referencia

- `int *d[5];`

  egész pointerek tömbje

- `int *h ();`

  funkció ami egész pointert ad vissza

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

  funkcióra mutató pointer ami egész pointert ad vissza

- `int (*v (int c)) (int a, int b)`

  egy egészet kérő funkció ami egy pointert ad vissza egy funkcióra ami két egészet kér és egy egészet ad vissza

- `int (*(*z) (int)) (int, int);`

  egy egészet kérő funkcióra mutató pointer ami egy pointert ad vissza egy funkcióra ami két egészet kér és egy egészet ad vissza

Megoldás videó:

Megoldás forrása:

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

```
#include <stdio.h>

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

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

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

}

int
```

```
main ()
{

    int (*f) (int, int);

    f = sum;

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

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

    g = sumormul;

    f = *g (42);

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

    return 0;
}
```

```
#include <stdio.h>

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

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

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

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

int
```

```
main ()
{

    F f = sum;

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

    G g = sumormul;

    f = *g (42);

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

    return 0;
}
```

Nagyon nehéz a functio-pontier-functio.... esatobbi deklarálása amikor iylen sokszor kell alkalmazni. Az a jó, hogy sokszor viszont nem kell alkalmazni ezt a módszert mert vagy túlbonyolítaná a dolgokat, vagy mert teljesen felesleges. Olvasni meg megérteni pláne a legnehezebb ezért ez a módszer kerülendő.

Helló, Caesar!

`double **` háromszögmátrix

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

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

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

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

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

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

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

    }

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

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

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

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

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

    free (tm);

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

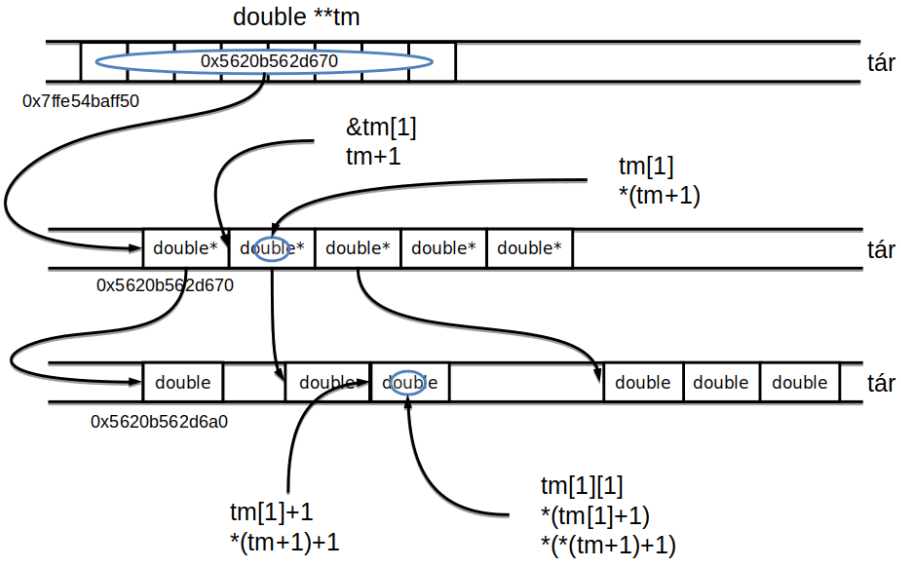

Figure 2: A `double **` háromszögmátrix a memóriában

C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat...

C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

Hiba-visszaterjesztéses perceptron

C++

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

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

Tanulságok, tapasztalatok, magyarázat...

Helló, Mandelbrot!

A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

A Mandelbrot halmaz a komplex síkon

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

```
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>
#define MERET 600
```

```
#define ITER_HAT 32000
void
mandel (int kepadat[MERET][MERET]) {
    //Mérünk időt (PP 64)
    clock_t delta = clock ();
    //Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);
    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;
    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;
                ++iteracio;
```

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

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1}=z_n{}^2+c$, $(0<=n)$ képlet alapján úgy, hogy a c az éppen vizsgált rácspont. A z0 az origó. Alkalmazva a képletet a

- $z_0 = 0$

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

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

A Mandelbrot halmaz a `std::complex` osztállyal

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

Megoldás forrása:

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

Ez a feladat olyan , mint az előző, néhány kissebb eltérés kivételével. Például a egy változóban fogjuk eltárolni a komplex számokat. De a lényeg itt is ugyan az. Megalkotunk egy üres png fájlt, és itt is két for ciklus segítségével beállítjuk a kívánt értékekre a kép pixeleit.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 -0.019473810573093663922605855987058021128l8 -0.019473
// ./3.1.2 mandel.png 1920 1080 1020 0.412765541820958925534057470940751954913l 0.41276554l8
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer="BATF41 HAXOR STR34M"
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  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 <https://www.gnu.org/licenses/>.
#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;
}
```

Biomorfok

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

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

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

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

```
{
  // c = (reC, imC) a halo racspontjainak
  // megfelelo komplex szam
  reC = a + k * dx;
  imC = d - j * dy;
  std::complex<double> c ( reC, imC );
  std::complex<double> z_n ( 0, 0 );
  iteracio = 0;
  while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
    {
      z_n = z_n * z_n + c;
      ++iteracio;
    }
```

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

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon
    for ( int k = 0; k < szelesseg; ++k )
    {
        double reZ = a + k * dx;
        double imZ = d - j * dy;
        std::complex<double> z_n ( reZ, imZ );
        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {
            z_n = std::pow(z_n, 3) + cc;
            if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
            {
                iteracio = i;
                break;
            }
        }
```

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

jelöléseivel:

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1  --left-footer="BATF41 HAXOR STR34M"
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
//  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 <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_
//
#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" <<
    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
```

```
        }
        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;
}
```

A Mandelbrot halmaz CUDA megvalósítása

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

Kezdetnek deviniáljuk a méretet(600) és az iterációs határt(32000), majd megalkotjuk a mandel halmazt:

```
<png++/image.hpp>
#include <png++/rgb_pixel.hpp>
#include <sys/times.h>
#include <iostream>
#define MERET 600
#define ITER_HAT 32000
__device__ int
mandel (int k, int j)
{
  float a = -2.0, b = .7, c = -1.35, d = 1.35;
  int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;
  float dx = (b - a) / szelesseg;
  float dy = (d - c) / magassag;
  float reC, imC, reZ, imZ, ujreZ, ujimZ;

  int iteracio = 0;
  reC = a + k * dx;
  imC = d - j * dy;

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

  while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
      ujreZ = reZ * reZ - imZ * imZ + reC;
      ujimZ = 2 * reZ * imZ + imC;
      reZ = ujreZ;
```

```
      imZ = ujimZ;
      ++iteracio;
    }
  return iteracio;
}
}
```

Az Nvidia fordító gépi kóddá változtatja azt a részt , ahol a "__global__" (vagy "__device__") szerepel. `threadIdx.x/y` abban lesz segítségünkre , hogy az x és y számokhoz melyik szálon fut az érték kiszámítása

```
__global__ void
mandelkernel (int *kepadat)
{
  int tj = threadIdx.x;
  int tk = threadIdx.y;
  int j = blockIdx.x * 10 + tj;
  int k = blockIdx.y * 10 + tk;
  kepadat[j + k * MERET] = mandel (j, k);
}
void
cudamandel (int kepadat[MERET][MERET])
{
  int *device_kepadat;
  cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

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

  cudaMemcpy (kepadat, device_kepadat,
          MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
  cudaFree (device_kepadat);
}
int
main (int argc, char *argv[])
{
  clock_t delta = clock ();
  struct tms tmsbuf1, tmsbuf2;
  times (&tmsbuf1);
  if (argc != 2)
    {
      std::cout << "Hasznalat: ./mandelpngc fajlnev";
      return -1;
    }
  int kepadat[MERET][MERET];
```

```
  cudamandel (kepadat);
  png::image < png::rgb_pixel > kep (MERET, MERET);
  for (int j = 0; j < MERET; ++j)
    {
      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;
}
```

Ezzel a programmal is egy Mandelbrot halmazt akarunk kirajzoltatni , viszont itt már Nvidia videókártya segítségével , ami jóval gyorsabban működne.

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!

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

Megoldás forrása: az ötödik előadás 26-33 fólia, illetve https://sourceforge.net/p/udprog/code/ci/master/tree/source/binom/Batfai-Barki/frak/.

frak.pro - Ennek a fájlnak a feladata lesz az összes kódot összeszedni, és ezt futtatva hozhatjuk létre a nagyítható Mandelbrot halmaz képét.

```
#############################################################################
# Automatically generated by qmake (3.0) Thu Apr 10 11:36:01 2014
#############################################################################
TEMPLATE = app
```

```
TARGET = frak
INCLUDEPATH += .
DEFINES += QT_DEPREACTED_WARNINGS
#include <QtWidgets>
QT += widgets
# Input
HEADERS += frakablak.h frakszal.h
SOURCES += frakablak.cpp frakszal.cpp main.cpp
```

Az egésznek van egy main fájlja is , ami a következő

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

    FrakAblak w1;
    w1.show();

    return a.exec();
}
```

frakablak.cpp

```
// frakablak.cpp
//
// Mandelbrot halmaz nagyt
// Programoz Pternoszter
//
// Copyright (C) 2011, Btfai Norbert, nbatfai@inf.unideb.hu, 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/>.
//
// Ez a program szabad szoftver; terjeszthet illetve mdosthat a
// Free Software Foundation ltal kiadott GNU General Public License
// dokumentumban lertak; akr a licenc 3-as, akr (tetszleges) ksbbi
```

```
// ultozata szerint.
//
// Ez a program abban a remmyben kerl kzreadsra, hogy hasznos lesz,
// de minden egyb GARANCIA NLKL, az ELADHATSGRA vagy VALAMELY CLRA
// VAL ALKALMAZHATSGRA val szrmaztatott garancit is belertve.
// Tovbbi rszleteket a GNU General Public License tartalmaz.
//
// A felhasznlnak a programmal egytt meg kell kapnia a GNU General
// Public License egy pldnyt; ha mgsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1    Br a Nokia Qt SDK ppen tartalmaz egy Mandelbrotos pldt, de
// ezt nem tartottam megfelelnek els Qt programknt ajnlani, mert elg
// bonyolult: hasznl klcsns kizrst stb. Ezrt "from scratch" megrtam
// egy sajtot a Javt tantokhoz rt dallamra:
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
//
#include "frakablak.h"
FrakAblak::FrakAblak(double a, double b, double c, double d,
                     int szelesseg, int iteraciosHatar, QWidget *parent)
                         : QMainWindow(parent)
{
    setWindowTitle("Mandelbrot halmaz");
    szamitasFut = true;
    x = y = mx = my = 0;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    magassag = (int)(szelesseg * ((d-c)/(b-a)));
    setFixedSize(QSize(szelesseg, magassag));
    fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, iteraciosHatar, this);
    mandelbrot->start();
}
FrakAblak::~FrakAblak()
{
    delete fraktal;
    delete mandelbrot;
}
void FrakAblak::paintEvent(QPaintEvent*) {
```

```
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    if(!szamitasFut) {
        qpainter.setPen(QPen(Qt::white, 1));
        qpainter.drawRect(x, y, mx, my);
    }
    qpainter.end();
}
void FrakAblak::mousePressEvent(QMouseEvent* event) {
    // A nagytand kijellt terletet bal fels sarka:
    x = event->x();
    y = event->y();
    mx = 0;
    my = 0;
    update();
}
void FrakAblak::mouseMoveEvent(QMouseEvent* event) {
    // A nagytand kijellt terlet szlessge s magassga:
    mx = event->x() - x;
    my = mx; // ngyzet alak
    update();
}
void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {
    if(szamitasFut)
        return;
    szamitasFut = true;
    double dx = (b-a)/szelesseg;
    double dy = (d-c)/magassag;
    double a = this->a+x*dx;
    double b = this->a+x*dx+mx*dx;
    double c = this->d-y*dy-my*dy;
    double d = this->d-y*dy;
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, iteraciosHatar, this);
    mandelbrot->start();
    update();
}
void FrakAblak::keyPressEvent(QKeyEvent *event)
{
    if(szamitasFut)
        return;
    if (event->key() == Qt::Key_N)
```

```
        iteraciosHatar *= 2;
    szamitasFut = true;
    delete mandelbrot;
    mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, iteraciosHatar, this);
    mandelbrot->start();
}
void FrakAblak::vissza(int magassag, int *sor, int meret)
{
    for(int i=0; i<meret; ++i) {
        QRgb szin = qRgb(0, 255-sor[i], 0);
        fraktal->setPixel(i, magassag, szin);
    }
    update();
}
void FrakAblak::vissza(void)
{
    szamitasFut = false;
    x = y = mx = my = 0;
}
```

frakszal.cpp

```
#include "frakszal.h"
FrakSzal::FrakSzal(double a, double b, double c, double d,
                   int szelesseg, int magassag, int iteraciosHatar, FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelesseg = szelesseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;
    egySor = new int[szelesseg];
}
FrakSzal::~FrakSzal()
{
    delete[] egySor;
}
// A szál kódját a Javát tanítokhoz írt Java kódomból vettem át
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
// mivel itt az algoritmust is leírtam/lerajzoltam, így meghagytam
// a kommenteket, hogy a hallgató könnyen hozzáolvashassa az "elméletet",
// ha érdekli.
void FrakSzal::run()
{
```

```
// A [a,b]x[c,d] tartományon milyen sűrű a
// megadott szélesség, magasság háló:
double dx = (b-a)/szelesseg;
double dy = (d-c)/magassag;
double reC, imC, reZ, imZ, ujreZ, ujimZ;
// Hány iterációt csináltunk?
int iteracio = 0;
// Végigzongorázzuk a szélesség x magasság hálót:
for(int j=0; j<magassag; ++j) {
    //sor = j;
    for(int k=0; k<szelesseg; ++k) {
        // c = (reC, imC) a háló rácspontjainak
        // megfelelő komplex szám
        reC = a+k*dx;
        imC = d-j*dy;
        // z_0 = 0 = (reZ, imZ)
    std::complex<double> c(reC, imC);

        reZ = 0;
        imZ = 0;
    std::complex<double> z_n(reZ, imZ);
        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+ std::atan(reZ*reZ - imZ*imZ) + std::sqrt(reC);
      ujimZ = 2*reZ*imZ+std::atan(2*reZ*imZ) + imC;
            reZ = ujreZ;
            imZ = ujimZ;
            ++iteracio;
        }
        */
    while( std::abs(z_n) < 4 && iteracio < iteraciosHatar) {
      z_n = z_n  * z_n + c;

      ++iteracio;
    }

        // ha a < 4 feltétel nem teljesült és a
```

```
            // iteráció < iterációsHatár sérülésével lépett ki, azaz
            // feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c
            // sorozat konvergens, azaz iteráció = iterációsHatár
            // ekkor az iteráció %= 256 egyenlő 255, mert az esetleges
            // nagyításoks során az iteráció = valahány * 256 + 255
            iteracio %= 256;
            //a színezést viszont már majd a FrakAblak osztályban lesz
            egySor[k] = iteracio;
        }
        // Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.
        frakAblak->vissza(j, egySor, szelesseg);
    }
    frakAblak->vissza();
}
```

frakablak.h

```
#ifndef FRAKABLAK_H
#define FRAKABLAK_H
#include <QMainWindow>
#include <QImage>
#include <QPainter>
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"
class FrakSzal;
class FrakAblak : public QMainWindow
{
    Q_OBJECT
public:
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
              double d = 1.35, int szelesseg = 600,
              int iteraciosHatar = 255, QWidget *parent = 0);
    ~FrakAblak();
    void vissza(int magassag , int * sor, int meret) ;
    void vissza(void) ;
    // A komplex sk vizsglt tartomnya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sk vizsglt tartomnyra fesztett
    // hl szlessge s magassga.
    int szelesseg, magassag;
    // Max. hny lpsig vizsgljuk a z_{n+1} = z_n * z_n + c itercit?
    // (tk. most a nagytsi pontossg)
    int iteraciosHatar;
protected:
    void paintEvent(QPaintEvent*);
    void mousePressEvent(QMouseEvent*);
```

```
    void mouseMoveEvent(QMouseEvent*);
    void mouseReleaseEvent(QMouseEvent*);
    void keyPressEvent(QKeyEvent*);
private:
    QImage* fraktal;
    FrakSzal* mandelbrot;
    bool szamitasFut;
    // A nagytand kijellt terletet bal fels sarka.
    int x, y;
    // A nagytand kijellt terlet szlessge s magassga.
    int mx, my;
};
#endif // FRAKABLAK_H
```

frakablak.h

```
#ifndef FRAKABLAK_H
#define FRAKABLAK_H
#include <QMainWindow>
#include <QImage>
#include <QPainter>
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakszal.h"
class FrakSzal;
class FrakAblak : public QMainWindow
{
    Q_OBJECT
public:
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
              double d = 1.35, int szelesseg = 600,
              int iteraciosHatar = 255, QWidget *parent = 0);
    ~FrakAblak();
    void vissza(int magassag , int * sor, int meret) ;
    void vissza(void) ;
    // A komplex sk vizsglt tartomnya [a,b]x[c,d].
    double a, b, c, d;
    // A komplex sk vizsglt tartomnyra fesztett
    // hl szlessge s magassga.
    int szelesseg, magassag;
    // Max. hny lpsig vizsgljuk a z_{n+1} = z_n * z_n + c itercit?
    // (tk. most a nagytsi pontossg)
    int iteraciosHatar;
protected:
    void paintEvent(QPaintEvent*);
    void mousePressEvent(QMouseEvent*);
    void mouseMoveEvent(QMouseEvent*);
```

```
    void mouseReleaseEvent(QMouseEvent*);
    void keyPressEvent(QKeyEvent*);
private:
    QImage* fraktal;
    FrakSzal* mandelbrot;
    bool szamitasFut;
    // A nagytand kijellt terletet bal fels sarka.
    int x, y;
    // A nagytand kijellt terlet szlessge s magassga.
    int mx, my;
};
#endif // FRAKABLAK_H
```

frakszal.h

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

Mandelbrot nagyító és utazó Java nyelven

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

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

```
/*
 * MandelbrotHalmazNagyító.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
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);
        }
    }
```

Tapasztalatok: Fontos , hogy ezt a MandelbrotHalmazNagyító.java állományt ugyan oda helyezzük , ahova a MandelbrotHalmaz.java állományunkat is tettük. A Mandelbrot halmaz nagyító programunk a Mandelbrot halmaz kiszámoló és kirajzoló programunknak a fejlesztett változata , mellyel a kirajzolt Mandelbrot halmaz kirajzolt képét nagyíthatjuk , aztán ismét és ismét ... Fordítása és futtatása: javac MandelbrotHalmazNagyító.java java MandelbrotHalmazNagyító

A bal egérgomb lenyomásával kijelöljük a nagyítandó területet. A MandelbrotHalmazNagyító osztályban ezt egy beágyazott, a MouseAdapter adapter osztályt kiterjesztő névtelen osztálybeli eseménykezelő objektummal oldjuk meg:

```
// 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();
```

```
        }
```

A kijelölő, zölddel kirajzolt, nagyítandó területet kijelölő téglalap bal felső sarkának koordinátái az osztály x és y tagjai. A kijelölő téglalap szélessége és magassága az mx és az my tagok.

```
    /** A nagyítandó kijelölt területet bal felső sarka. */
    private int x, y;
    /** A nagyítandó kijelölt terület szélessége és magassága. */
    private int mx, my;
```

Ezek állítása történik az eseménykezelő egérkattintást feldolgozó mousePressed() módszerében. A függvény aktuális paramétereként megkapott eseményobjektumtól megkérdezzük a kattintás helyét:

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

az mx és my szélességet, magasságot nullára állítjuk, ezeket majd az egérgomb nyomvatartása melletti vonszolás állítja be a vonszolás méretének megfelelő értékre. Az egérmutató vonszolásának megfigyelését ugyancsak egy beágyazott, de most a MouseMotionAdapter adapter osztályt kiterjesztő névtelen osztálybeli eseménykezelő objektummal oldjuk meg:

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

Az x és y példányok tárolják a négyzet bal felső sarkának oszlop és sor koordinátáját, ezeket az egérmutató aktuális helyéből kivonva kapjuk a kijelölt terület szélességét, illetve magasságát, például: mx = m.getX() - x;. A repaint() hívása biztosítja, hogy a megváltozott példánytagoknak megfelelő zöld tégla kirajzolásra kerüljön, azaz a paint() függvény meghívódjon, ahol

```
// A jelző négyzet kirajzolása:
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
```

Ha úgy érezzük, hogy a nagyítások során leromlott a pontosság, akkor az n gomb nyomásával növelni tudjuk a számítások iterációs határát. Ezt a billentyűzet eseményt egy beágyazott, a KeyAdapter adapter osztályt kiterjesztő névtelen osztálybeli eseménykezelő objektummal oldjuk meg:

```
// A billentyűzetről érkező események feldolgozása
        addKeyListener(new java.awt.event.KeyAdapter() {
            // Az 's', 'n' és 'm' gombok lenyomását figyeljük
            public void keyPressed(java.awt.event.KeyEvent e) {
                ...
                // Az 'n' gomb benyomásával pontosabb számítást végzünk.
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    if(számításFut == false) {
                        MandelbrotHalmaz.this.iterációsHatár += 256;
                        // A számítás újra indul:
                        számításFut = true;
                        new Thread(MandelbrotHalmaz.this).start();
                    }
```

Helló, Welch!

Első osztályom

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

Megoldás videó:

Megoldás forrása:

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!

LZW

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

Megoldás videó:

Megoldás forrása:

Fabejárás

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

Megoldás videó:

Megoldás forrása:

Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

Helló, Conway!

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:

Az osztálydiagram:

A megoldás forrása, illetve az UML osztálydiagram Bátfai Norbert tulajdona.

Ebben a feladatban hangyákat kell szimulálnunk. A megoldás azt a biológiából már ismert tényt használja, hogy a hangyák a való életben szagokkal (feromonokkal) kommunikálnak egymással. Ha például egy hangya valamilyen érdekes dolgot talált, akkor ott hagyja a nyomát, illetve megjelöli az útvonalat. Az éppen arra járó többi hangya ezt megérzi, a legfrissebb feromon nyomát követve ők is el fognak jutni a célba. Ezeket felhasználva készítették el ezt a hangyaszimulációt.

Az osztálydiagrammon belül négy egységet találhatunk, ezek a következők: Ant; Ants; AntWin; és AntThread.

Ezek a programunk osztályai, ezeken az egységeken belül vannak megadva az adott osztály változói és függvényei is. Ilyen például az AntWin egységen belül található *width* és *height* változók, amik a képernyő hosszúságát és szélességét adják meg. Vagy például a `closeEvent()` és a `keyPressEvent()` függvények, amik szintén az AntWin osztály részei. Ezek alapján meghatározhatjuk, hogy az AntWin osztály a szimuláción belül a világot kezeli.

Az AntThread osztály kezeli a hangyákat, mozgásukat, illetve a virtuális feromonok terjedéséért is ez az osztály felelős.

Java életjáték

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

Megoldás videó:

Megoldás forrása:

A Megoldás forrása Bátfai Norbert tulajdona.

Arról, hogy mi az az életjáték, illetve, hogy mik a szabályai, a következő feladat leírásában részletesebben kifejtem. Röviden az életjáték szabályai:

Ha egy négyzetnek pontosan három darab élő szomszédja van, akkor abban a négyzetben egy új sejt jön létre.

Ha egy már élő sejtnek pontosan kettő vagy három darab szomszédja van, akkor az a sejt továbbra is életben marad.

Ha viszont egy már meglévő élő sejtnek háromnál több élő szomszédja van (túlnépesedés), vagy kettőnél kevesebb, akkor az a sejt meghal. Ezt szimulálja az életjáték.

Ezek a szabályok az `időFejlődés()` függvényben vannak lefektetve.

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

Igaz ugyan, hogy az életjáték egy úgynevezett nullszemélyes játék, de ebben a példában a játékos mégis tudja irányitani kicsit a dolgokat. Ugyanis a program figyeli a billentyűzet bizonyos gombjait (k, n ,l ,g ,s), illetve az egér mozgását, és kattintásait is. Ezt három fügvénnyel teszi. Az `addKeyListener(new java.awt.event.KeyAdapter()` fügvénnyel figyeli a billentyűzetet. Ezen a függvényen belül egy if-else szerkezet állapitja meg, hogy éppen melyik gombot nyomta le a felhasználó a

```
if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K){}
```

feltétel a K betű lenyomását azonosítja, és csökkenti a sejtek méretét.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N){}
```

feltétel az N betű lenyomását azonosítja, és növeli a sejtek méretét.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S){}
```

feltétel az S betű lenyomását azonosítja, és készít egy képet a sejttérről.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
```

feltétel a G betű lenyomását azonosítja, gyorsítja a szimuláció sebességét.

```
else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L
```

feltétel az L betű lenyomását azonosítja, és lassítja a szimuláció sebességét.

Az egér mozgását, illetve kattintásait pedig a `addMouseListener(new java.awt.event.MouseAdapter(){}` illetve a `addMouseMotionListener(new java.awt.event.MouseMotionAdapter(){}` függvények figyelik. Az egér kattintásaival egy sejt állapotát tudjuk megváltoztatni. Az egér mozgatásával pedig az összes érintett sejt élő állapotba kerül.

Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

A megoldás forrása Bátfai Norbert tulajdona.

Az életjátékot John Conway találta ki, és nem teljesen hiteles rá a játék kifejezés, mert ez egy úgynevezett nullszemélyes játék. Magának a játékosnak annyi a dolga, hogy megadja a kezdőalakzatot, majd pedig megfigyelheti, hogy mi lesz az eredmény.

Alapja egy négyzetrácsos tér, amikben élhetnek sejtek, de minden egyes négyzetben csak egy darab sejt élhet. A játék szabályai a következők:

Ha egy sejtnek kettő vagy három élő szomszédos sejtje van, akkor a sejt meg fogja élni a következő generációt. Az összes többi esetben viszont kihal a sejt, akár azért mert túl sok, akár azért mert túl kevés szomszédja van.

Ahol azonban egy üres négyzetrácsnak pontosan három élő sejt a szomszédja, akkor ott új sejt jön létre.

Érdekes az, hogy milyen hatást váltott ki az emberekből. Voltak akik napi rutinjukká tették azt, hogy az életjátékkal "játszanak", egyfajta függők lettek, és voltak azok, akik nem értették hogy mi a jó benne.

Maga a program ugyanúgy működik, mint a java verzió. Mind a két program két darab mátrixxal dolgozik, viszont itt a teljes kód megirása helyett a Q-t is segítségül hívjuk.

A programot a következőképpen tudjuk fordítani és futtatni: `qmake Sejtauto.pro make ./Sejtauto`

BrainB Benchmark

Megoldás videó:

Megoldás forrása:

A megoldás forrása Bátfai Norbert tulajdona.

A programhoz szükségünk lesz az OpenCV-re, aminek a feltelepitéséhez a lépéseket ezen a linken elérhető weblapon találjuk.

Ez egy miniatűr játék, ami a felhasználó szem-kéz koordinációjáról, illetve a megfigyelőképességéről gyűjt össze információkat.

Amikor elindítjuk a játékot akkor egy ablak fogad minket, és a lényege az, hogy a `Samu enthropy` nevű négyzeten belül lévő fekete pöttyön belül tartsuk az egerünk mutatóját.

A játék a teljesitményünk alapján lesz könnyebb, vagy nehezebb. Minél jöbban játszunk, annál több Enthropy lesz a képernyőn, ezáltal megnehezítve a Samu enthropy követését. Viszont ha már nem tudjuk nyomon követni a Samu enthropy-t akkor folyamatosan eltünteti a hozzáadott enthropy-kat, ezáltal megkönnyítve a játékot.

A programot futtatni az előző feladathoz hasonlóan szintén a `qmake` és `make` parancsokkal lehet forditani.

Vörös Pipacs Pokol/19 RF

Megoldás videó: https://youtu.be/VP0kfvRYD1Y

Megoldás forrása:

A Red Flower Hell II. körében a célunk ugyanaz mint eddig volt: a lehető legtöbb pipcsaot gyűjteni, amíg a láva leér az aréna aljába. Azonban az első körhöz képest néhány változtatást hozott Tanár Úr a szabályokban: tilos az XML fájl adatait felhasználni, tilos abszolút mozgási formát használni, tilos játékmódot változtatni, tilos az XML fájlt változtatni és tilos átvenni az egér fölött az irányítást.

Helló, Schwarzenegger!

Szoftmax Py MNIST

Python

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

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

Tanulságok, tapasztalatok, magyarázat...

Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Helló, Chaitin!

Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

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...

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...

Helló, Gutenberg!

Programozási alapfogalmak

PICI

1.2 Alapfogalmak

A számítógépes programozás három nyelvét különböztethetjük meg: gépi nyelv, assembly szintű nyelv, vagy magas szintű nyelv. Egy magas szintű programnyelvet a szintaktikai és szemantikai szabályai határoznak meg, és jelen kell lennie mindkettőnek. A szintaktika a formai szabályok, míg a szemtanika a tartalmi, jelentésbeli szabályok együttese. Ahhoz, hogy egy magas szintű programozási nyelven megírt programot le tudjunk futtatni, azt előbb le kell fordítani a processzor saját gépi nyelvére. Erre kétféle lehetőségünk van, a fordítóprogramos és az interpreteres.

A fordítóprogram a magas szintű nyelvű forrásból egy gépi kódú tárgyprogramot kreál négy lépésben. Lexikális elemzés, szintaktikai elemzés, szemantikai elemzés, majd kódgenerálás. Amennyiben az adott nyelv szabályai szerint szintaktikailag helyes, az elkészült tárgyprogramból a kapcsolatszerkesztő állít elő egy már ténylegesen futtatható programot.

Az interpreteres megoldás esetén is megvan az első három lépés, de itt nem készül tárgyprogram. Helyette utasításonként sorra veszi a forrásprogramot, értelmezi, majd végre is hajtja azt.

A programnyelvek lehetnek vagy fordítóprogramosak, vagy interpreteresek is, de lehetnek egyszerre mindkettő is.

Programozás bevezetés

KERNIGHANRITCHIE

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

Alapismeretek

Az első fejezet, mint bevezetés, írat velünk egy egyszerű helloworld programot, ezzel megtanítva, egyszersmint begyakoroltatva velünk az alapvető szintaktikát, valamint a fordítás és futtatás miként jét. A továbbiakban új alapfogalmakat tanít, mint hogy a C-beli programok .c-re végződnek, vagy épp hogy mi az a változó és milyen típusai vannak. Ezután találkozhatunk még olyan fontos elemekkel, mint a ciklusok, feltételes elágazások, tömbök és függvények. Az olvasót végig arra próbálják sarkallni, hogy minél többet gyakoroljon.

Típusok, operátorok és kifejezések

A második fejezetben olvashatunk bővebben a változókról, például a nevükhöz tartozó konvenciókról, a típusokról és azok méreteiről. Bővebb szó esik az értékadásról, megjelennek a konstansok. Érdemben tárgyal az aritmetikai, relációs és logikai operátorokról. Típuskonverziók.

Programozás

BMECPP

Ebben a kis olvasmányban megismerhettük a C++ és elődje, a C közötti néhány különbséget. C-vel ellentétben C++-ban az üres paraméterlista `void` visszatérési értéket jelent. Szintén lényeges különbséget jelent, hogy már nem kötelező a return használata, azt a fordítóprogram automatikusan a végére teszi. C++-ban megjelent a `bool` típusú változó, ami logikai igaz/hamis értékekkel dolgozik. Ebből kifolyólag a nyelv kulcsszavai részévé váltak a bool, true és false kifejezések.

Mobilprogramozás

MOBIL

2.3. Python

A Python egy magas szintű, általános célú programozási nyelv melyet a szkriptnyelvek családjába szoktak sorolni. A Pythonban megírt programokat általában valamilyen fejlesztői környezetben tudjuk futtatni. Egyik nagy előnye, hogy rendelkezik egy rendkívül hasznos alapkönyvtárral, mely számos praktikus segítséget tartalmaz, ilyenek például a szabványos kifejezések. Másik nagy előnye a bővíthetőség, C-ben vagy C++-ban írt programrészekkel is kompatibilis tud lenni, így azokkal könnyen bővíteni lehet a kódot. Népszerűsége egyszerűségéből

fakad. Fontos hátulütője viszont, hogy a legtöbb mobil alapvetően nem rendelkezik Python-futtató-környezettel, így hiába egyszerűbb megírni egy programot, nehezebb terjeszteni azt. Ha viszont egy mobil rendelkezik Python-futtató-környezettel, akkor lehetővé teszi azt is, hogy mobilon programozzunk és teszteljünk is, bár előbbi nem ajánlott hosszabb programkódok esetén.

Tip

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.

Helló, Arroway!

A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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...

Irodalomjegyzék

Általános

MARX György Marx Typotex 2005

C

KERNIGHANRITCHIE Brian W. Kernighan Dennis M. Ritchie Bp., Műszaki 1993

C++

BMECPP Zoltán Benedek Tihamér Levendovszky Bp., Szak Kiadó 2013

Lisp

METAMATH Gregory Chaitin 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.