Univerzális programozás

Így neveld a programozód!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Ã©s Bátfai, Margaréta	2020. április 27.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai

Ajánlás

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

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

Tartalomjegyzék

Ábrák jegyzéke

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: 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:

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

  return 0;
}
```

Minden mag 100 százalékban:

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

A **gcc infty-f.c -o infty-f -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
%Cpu4 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu6 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu7 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem :16373532 total,11701240 free, 2254256 used, 2418036 buff/cache
KiB Swap:16724988 total,16724988 free,       0 used. 13751608 avail Mem

  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
```

```
#include <unistd.h>
int
main ()
{
for (;;)
sleep(1);
return 0;
}
```

mag0.c

```
#include <unistd.h>
#include <stdbool.h>
int main(){
while (true);
{
    sleep(1);
}

    return 0;
}
```

mag100.c

```
#include <omp.h>
#include <unistd.h>
#include <stdbool.h>

int
main ()
{
#pragma omp parallel

while(true);
{
    sleep(1);
```

```
}
return 0;
}
```

allmag100.c

Werkfilm

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

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: https://github.com/HernyakPisti/Prog1/blob/master/bhax/thematic_tutorials/bhax_textbook_IgTuring/valtcsere.c

Tanulságok, tapasztalatok, magyarázat...

```
#include <stdio.h>

int main()
{
    int a, b;
    printf("Add meg az a erteket: ");
    scanf("%d", &a);//a=10
    printf("Add meg a b erteket: ");
    scanf("%d", &b);//b=5
    a=a+b; //a=15 b=5
    b=a-b;//a=15 b=10
    a=a-b;//a=5 b=10
    printf("a: %d b: %d\n",a,b);
    a=a*b;//a=50 b=10
    b=a/b;//a=50 b=5
    a=a/b;//a=10 b=5
    printf("a: %d b: %d\n",a,b);
    return 0;
}
```

2.4. Labdapattogás

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

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

Megoldás forrása: https://github.com/HernyakPisti/Prog1/blob/master/bhax/thematic_tutorials/bhax_textbook_IgTuring/pattogifnelkull.c

Tanulságok, tapasztalatok, magyarázat...

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int sz=10,m=10,a=1,b=1;
#define SZG 50
#define MG 20
int falsz[SZG];
int falm[MG];
void palya(){
system("clear"); //letöröljük az ablakot hogy jól látszódjon a ↩
   pálya
for (int i=0;i<(SZG+2);i++) //megrajzoljuk az első sort
```

```
    printf("X");
    printf("\n");
for (int i=0;i<m;i++){ //megrajzoljuk a sorokat addig a sorig ↩
   amíg nem kell a labdát
    printf("X");
        for (int j=1;j<(SZG+1);j++)
        printf(" ");
    printf("X\n");
}   //megrajzoltuk a ladba előtti sort
printf("X");  //a labdát tartalmazó sor bal oszlopi X-je
for (int i=0;i<sz;i++) //a sort addig töltjük fel " "-kel amíg ↩
   nem kell magát a labdát megrajzolni
    printf(" ");
printf("X");  //labda kirajzolása
for (int i=(sz+1);i<SZG;i++) //a labda után feltöltjük a sort " ↩
    "-kel
    printf(" ");
printf("X\n"); //labdát tartlmazó sor vége
for (int i=m+1;i<MG;i++){ //labda utáni sorok rajzolásának ↩
   kezdete
    printf("X");
        for (int j=1;j<(SZG+1);j++)
        printf(" ");
    printf("X\n");
}   //utolsó előtti sorig való megrajzolás
for (int i=0;i<(SZG+2);i++) //megrajzoljuk az utolsó sort
    printf("X");
    printf("\n");
}
void mozdulj(){
sz=sz+a; //labdá helyzetetét növeljük
m=m+b; //labdá helyzetetét növeljük
a=a*falsz[sz]; //labda irányának megállapítása 1 vagy -1
b=b*falm[m]; //labda irányának megállapítása 1 vagy -1
}

int main(){

for (int i=1;i<SZG;i++) //feltöltjük a tömböt 1-sel hogy a ↩
   labda haladjon a jó irányba
    falsz[i]=1;
for (int i=1;i<MG;i++) //feltöltjük a tömböt 1-sel hogy a labda ↩
    haladjon a jó irányba
    falm[i]=1;
falsz[0]=-1; //falhoz ért a labda az irányt változtatjuk
falsz[SZG-1]=-1; //falhoz ért a labda az irányt változtatjuk
falm[0]=-1; //falhoz ért a labda az irányt változtatjuk
falm[MG-1]=-1; //falhoz ért a labda az irányt változtatjuk

while(free){
```

```
            palya();
            mozdulj();
            usleep(50000); //látszódjon a labda rendesen ne okozzon  ↩
               Parkinson kórt
            }
            }
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU, https://youtu.be/KRZlt1ZJ3qk, .

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bogomips.c

Tanulságok, tapasztalatok, magyarázat...

```
            #include <stdio.h>

            int main(){

                int szo;
                int darab=0;

                printf("Adj meg egy integer típusú számot: ");
                scanf("%d", &szo);

                while(a << 1){

                    darabb++;
                }

                printf("%d\n",darabb);


                return 0;
            }
```

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: https://github.com/HernyakPisti/Prog1/blob/master/bhax/thematic_tutorials/bhax_textbook_IgTuring/pagerank.c

Tanulságok, tapasztalatok, magyarázat...

```
#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;
    }
```

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

2.8. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

A 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){

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

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

Soronként értelemezzük ezt a programot:

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

Kiszámolja a megadott számig a prímeket.

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

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

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

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

```
idx = which(diff==2)
```

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

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

```
t1primes = primes[idx]
```

Kivette a primes-ból a párok első tagját.

```
t2primes = primes[idx]+2
```

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

Az 1/t1primes a t1primes 3,5,11 értékéből az alábbi reciprokokat képzi:

```
> 1/t1primes
[1] 0.33333333 0.20000000 0.09090909
```

Az 1/t2primes a t2primes 5,7,13 értékéből az alábbi reciprokokat képzi:

```
> 1/t2primes
[1] 0.20000000 0.14285714 0.07692308
```

Az 1/t1primes + 1/t2primes pedig ezeket a törteket rendre összeadja.

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

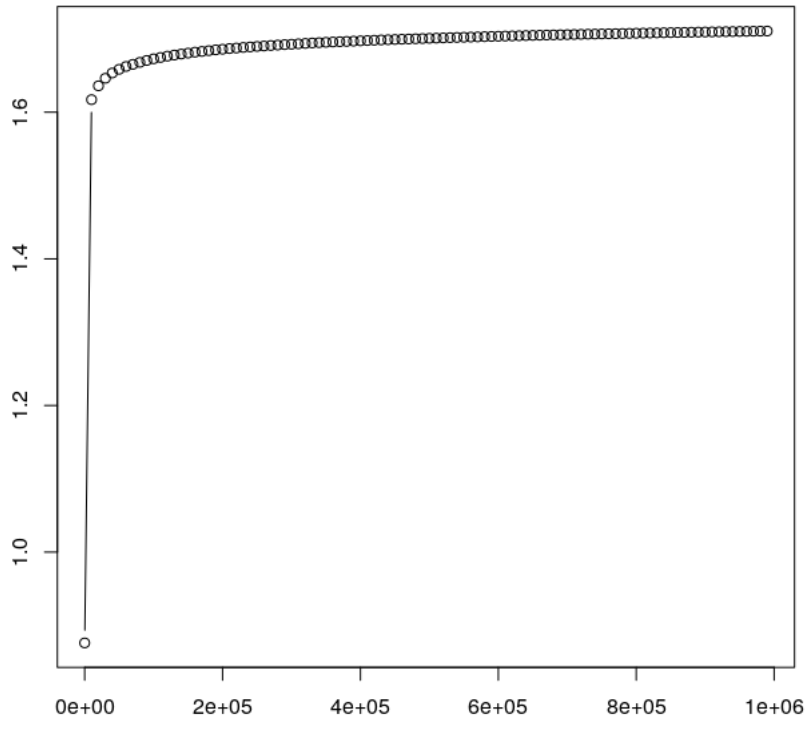
Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvénnyel.

```
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")
```



2.1. ábra. A B_2 konstans közelítése

Werkfilm

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

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Backus-Naur-forma környezetfüggetlen nyelvtanok leírására használható metaszintaxis: végeredményben formális nyelvek is leírhatók vele. A legtöbb programozási nyelv elméleti leírása és/vagy szemantikai dokumentumai általában BNF-ban vannak leírva. A BNF széles körben használatos a számítógépek programozási nyelveinek nyelvtanának leírására, ideértve az utasítás készleteket és a kommunikációs protokollokat is.

```
<utasítás> ::= <címkézett_utasítás> | <kifejezés_utasítás> | < ↩
   összesített_utasítás> | <kiválasztó_utasítás> | <literációs_utasítás> |  ↩
   <vezérlésátadó_utasítás>
<címékezett_utasítás> ::= <azonosító> ";" <utasítás>  | "case" < ↩
   állandó_kifejezés> ":" <utasítás> | "default :" <utasítás>
<kiválasztó_utasítás> ::= "if" "(" <kifejezés> ")" <utasítás> | "if" "(" < ↩
   kifejzés> ")" <utasítás> "else" <utasítás> | "switch" "("<kifejezés>")"  ↩
   <utasítás>
<kifejezés_utasítás> ::= [<kifejezés>] ";"
<vezérlésátadó_utasítás> ::= "goto" <azonosító> ";" | "continue;" | "break ↩
   ;" | "return" [<kifejezés>] ";"
<összesített_utasítás> ::= "{" [<deklarációs_lista>] [<utasítási_lista>]  ↩
   "}"
<deklarációs_lista> ::= <deklaráció> | <deklarációs_lista> <deklaráció>
<literációs_utasítás> ::= "while" "(" <kifejezés> ")" <utasítás> | "do"  < ↩
   utasítás> "while" "(" <kifejezés> ")" <utasítás> | "for" "(" [ < ↩
   kifejezés> ";" <kifejezés> ";"  <kifejezés> ] ")" <utasítás>
<utasítási_lista> ::= <utasítás> | <utasítási_lista> <utasítás>
```

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
#include <stdio.h>

int main() {
for (int i=1;i<=10;i++)
printf("Szamolunk %d\n",i);
retutn 0;
}
```

A c89-es szabványban hiba hogyha a ciklus magban deklarálunk egy változót (lásd int i=0), ezt csak c99 vagy c11 szabványban fordíthatjuk le hiba mentesen.

3.4. Saját lexikális elemző

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

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

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

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

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

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

  {'a', {"4", "4", "@", "/-\\"}},
  {'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... Ez is egy .l kiterjesztésű fájl vagyis ez egy lexer forráskód ami egy .c kiterjesztésű kódot készít. Ez is 3 részből áll. Az első részben incluidoljuk a kötelező dolgokat és létrehozunk egy cipher nevű struckuret. Ami egy charból és egy char típusú mutatútömbböl áll, aminek 4 eleme van. A szabályok részen belül minden bemeneti karakterre igaz feltétel áll mivel a feltétel "." ami akármiből egyet jelent. A bemeneti beolvasása után a kód talált egy megegyező karaktert akkor generál egy véletlen számot és ezzel a számmal választ egy elemet a leet tömbből. A leet tömb elemei között más valószínűségi esélyek vannak. Ezek rendre 90% 4% 3% 3%. És mivel egyezésünk volt a found változót "igazra váltjuk". Ha a found "hamis" marad akkor a kód végén lévő if teljesül ami általl visszakapjuk a vizsgált karaktert. A kód legévégén srand-dal készítünk véletlenszámot.

3.6. A források olvasása

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

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

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

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

Ha a SIGINT jel figyelmen kívül volt hagyva akkor maradjon úgy, azonban ha nem volt figyelmen kívül hagyva akkor a jelkezelő fügvény kezelje.

A ciklusban deklarált i változó értékehez először hozzáadunk egyet majd visszadja a megnövelt értékét.

A ciklusban deklarált i változó értékét először visszadja aztán megnöveli eggyel.

A programnak lekellene futnia de mivel az i változó egyszerre változik és van értékként megadva ezért hiba történik.

Rosszul van a for ciklus ciklusmagja felírva, mivel a (*d++ = *s++) a ciklusmagban van felírva és itt feltételként kéne szerepelni == -jel.

A kód 2 számot fog kiírni az f függvénytől függően. Az első esetben "a" eggyel kisebb mint ++a, a második esetben viszont ++a és "a" ugyan az a szám lesz.

A kód 2 számot fog kiírni. Az első szám amit f függvény ad vissza, a második szám pedig az a szám amit f-nek adtuk az első esetben.

A kód 2 számot fog kiírni. Az első esetben azt a memória címet írja ki amit az f függvény ad meg, a második esetben pedig azt a változó jelenik meg aminek kiírta a kód a memória címét.

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

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

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

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

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

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

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

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

```
#include <stdio.h>

  int* eg(); //egészre mutató mutatót visszaadó függvény

  int main () {
```

```
int a; //egész
int* b; //egészre mutató mutató
int* c = &a; //egész referenciája
int d[100]; //egészek tömbbje
int* e = &d[10] //egészek tömbbjének referenciája (nem az első elemé)
int* pp[100]; //egészre mutató mutatók tömbbje
int* (*eg_pointer) ();//egészre mutató mutatót visszaadó függvényre ↩
   mutató mutató
int (*egesz (int c3)) (int c1, int c2);//egészet visszaadó és két egészet ↩
    kapó függvényre mutató mutatót visszaadó, egészet kapó függvény
int (*(*egeszre) (int f3)) (int f1, int f2); //függvénymutató egy egészet ↩
    visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, ↩
   egészet kapó függvényre
return 0;
}
```

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása:

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

```
#include <stdio.h>

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

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

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

}

int
main ()
{

    int (*f) (int, int);

    f = sum;

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

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

    g = sumormul;

    f = *g (42);

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

    return 0;
}
```

```
#include <stdio.h>

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

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

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

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

int
main ()
{

    F f = sum;

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

    G g = sumormul;

    f = *g (42);

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

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

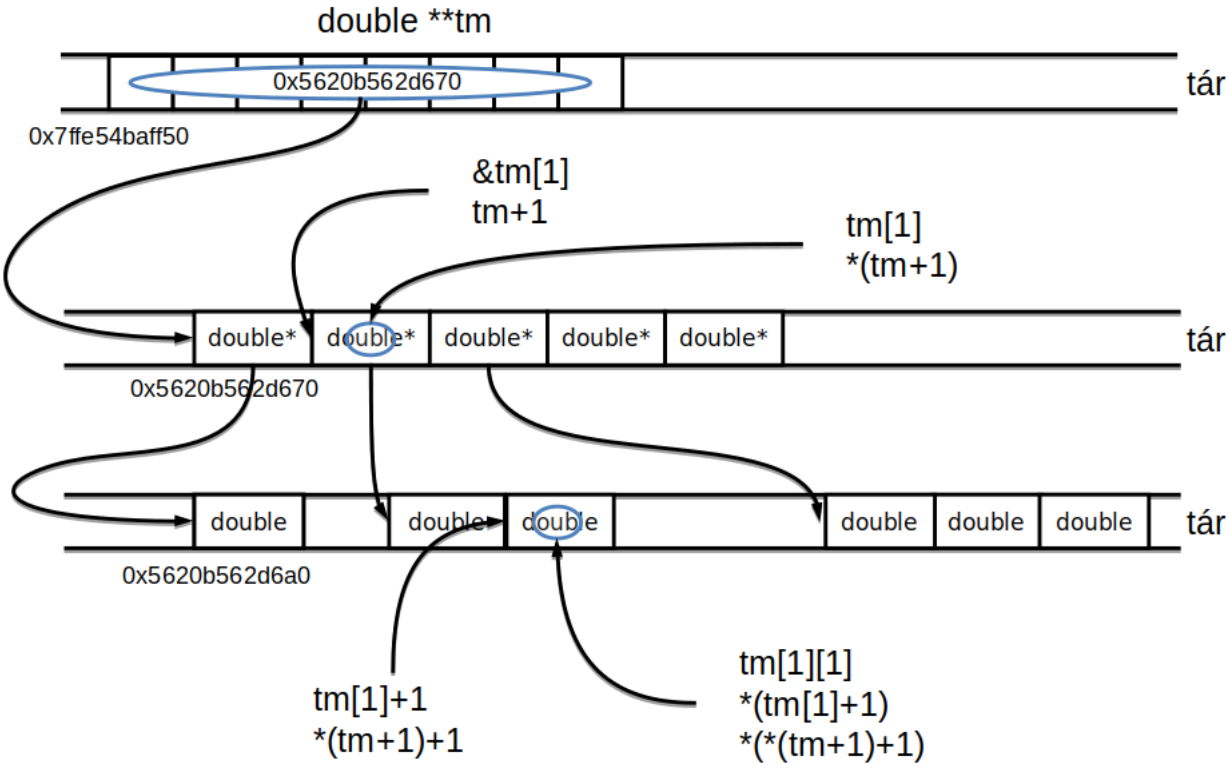
Írj egy olyan `malloc` és `free` párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

Megoldás videó: https://youtu.be/1MRTuKwRsB0, 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; //nr nevű int változó deklarálása
    double **tm; //tm nevű mutatú deklarálása

    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL) //HA
       tudunk, akkor lefoglalunk a memoriaban nr* double*-ot, azaz jelen
       esetben 5*8at, azaz 40et a malloc nevu utasitassal, ami visszater
       void*-al, de mi megmondjuk neki, hogy double ** tipusu legyen.
    {
        return -1; //ha nem tud lefoglalni memóriát, egyszerűen kilépünk
    }//ha sikeresen lefoglaltuk, akkor letrejon a tm a memoriaban, ami
       jelen esetben 5 darab double* nak az elejere fog mutatni

    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL
           )//ismet, ha tudunk, akkor lefoglalunk i+1 * double-t, azaz
           jelen esetben eloszor 1*8at ... majd a vegen 5*8at
            //ezt elmentjuk a tm[i]- be, ami tulajdonkeppen a *(tm), *(tm
               +1), vagy *(tm+2) ...
        {
```

```
            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; //alsó háromszög feltöltése

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

    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső (),semmi
    *(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]);//felszabaditjuk a tm[i]-ket, amikben az 1*8, 2*8, ... ↩
            lefoglalt double-k memoriacime van

    free (tm);//felszabadítjuk a tm-t

    return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

A double ** háromszögmátrix egy olyan háromszögmátrix, pontosabban alsó háromszögmátrix, ami a memóriában van felépitve, pointerek által.

Első sorban a **double **tm** jön képbe. Ez egy úgynevezett "dupla" pointer ami double*-ra mutat. Az nr és a double* méretének szorzatával egyenlő méretű memóriát foglalunk le, aminek a **malloc** a memóriacimét visszaadja, és a tm-ben lesz eltárolva.

Következő lépésben az előzőleg lefoglalt tm[i]-k(*(tm)) fognak értéket kapni. Ezekbe is memóriacimeket fogunk tárolni. Mindegyikbe egy i+1 * double méretű memóriaterület memóriacimét, amit szintén a malloc-al valósitunk meg tehát a malloc-al lefoglalunk i+1 * double méretű memóriaterületet, és mindegyik ilyen memóriaterület memóriacimét a tm[i]-ben tárolunk. Ha nem lehetséges, akkor ugyanúgy mint az előző lépésben, visszatér a program -1es értékkel és véget ér.

Ezután már csak fel kell tölteni a háromszögmátrixot, amit a dupla forral teszünk meg. Tulajdonképpen a tm[i][j] az felfogható *(*(tm+i)+j) ként is. Ezt követően ugyanúgy duplaforral ki is iratjuk a háromszög mátrixunkat.

Néhány alternativ módon értéket adunk a mátrix 3. sor 0,1,2,3 oszlopainak, és ezután újból kiiratjuk a mátrixunkat a duplaforral.

Legvégső, nagyon fontos lépés a memória felszabaditás, amit a **free** utasitással teszünk meg. Először is felszabaditjuk a tm[i]-t, amiben ugye a double mezők memóriacime van, majd ezt követően felszabaditjuk a tm-et is, ami mutatott a tm[i]-re. Ezt követően véget is ér a programunk.

Véleményem szerint eléggé érdekes volt egy ilyen háromszög mátrixot értelmezni, feldolgozni, és el is magyarázni.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: Megoldás forrása: https://github.com/HernyakPisti/Prog1/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/titkosit.c

Tanulságok, tapasztalatok, magyarázat...

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

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

int kulcs_index = 0;
int olvasott_bajtok = 0;

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

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

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

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

     }

     write (1, buffer, olvasott_bajtok);

    }
}
```

A titkositás úgy történik, hogy a bufferbe beolvasott byteokat bitenként XOR - kizáró vagy - művelettel "összefésül" a kulcs index-el, amiből egy új karaktert kapunk az ASCII táblázat kereteiből, és bele is irja

azt a titkos fájlba, tulajdon képpen ezen alapszik ez a titkosító algoritmus, ami azért ránézésre elég ijesztő dolgokat tud kiadni magából.

4.3. 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...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: https://github.com/HernyakPisti/Prog1/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/toro.c

Tanulságok, tapasztalatok, magyarázat...

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 3 // 3 betus a kulcs
#define _GNU_SOURCE

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

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

return (double) titkos_meret / sz;
}

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

```
 // potenciális töréseket

 double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

int kulcs_index = 0;

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

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

 }

}

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

exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);

 }

int
main (void)
{

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

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

                   // maradek hely nullazasa a titkos bufferben
                 for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
                     titkos[p - titkos + i] = '\0';
                   char str[3]= {'e', 'k', 'y'};//ide irod a karaktereket  ↩
                      amibol a kulcsod all
                // osszes kulcs eloallitasa
                    for (int ii = 0; ii <= 2; ++ii) //csak 3 egybeágyazott ↩
                        ciklus kell
                       for (int li = 0; li <= 2; ++li)
                           for (int ki = 0; ki <= 2; ++ki)
                           {
                               kulcs[0] = str[ii]; //csak 3 karakter  ↩
                                  hosszú a kulcs
                               kulcs[1] = str[li];
                                kulcs[2] = str[ki];

                                if (exor_tores (kulcs, KULCS_MERET, titkos ↩
                                   , p - titkos))
                                    printf
                                    ("Kulcs: [%c%c%c]\nTiszta szoveg: [%s ↩
                                       ]\n", //csak 3 karaktert irjon ki a ↩
                                        kulcsnál
                                     kulcs[ii], kulcs[li], kulcs[ki],  ↩
                                        titkos);

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

                    return 0;
            }
```

A program bonyolultsága egy fokkal nagyobb, mint az előző titkosító programé, de ezt se lehetetlen felfogni. A célunk az az, hogy a titkositott szöveghez találjuk meg a kulcsot, amivel újfent elvégezzük az XOR műveletet a titkos szöveg bitjeivel, és reméljük, hogy visszakapjuk az eredeti tiszta szövegünket.

Az átlagos szóhossz fügvénnyel megnézzük, hogy milyen az átlagos szóhossz, és ezzel jelentősen tudjuk csökkenteni a lehetséges töréseket

A tiszta_lehet függvény a gyakran előforduló magyar szavakat veszi figyelembe, azokat rögziti, hogy könnyebben be lehessen azonositani ezeket, mivel ezek a szavak szinte minden terjedelmesebb szövegbe legalább egyszer előfordúlnak.

A lényeges lépéseket az exor illetve az exor_tores alkalmazza. Az első az konkrétan alkalmazza az XOR-t a a bufferbe beolvasott titkos szövegen és a kulcs indexen, amivel visszakapható a tiszta szöveg, és az exor_tores pedig meghivja azt, és visszaadja a tiszta_lehet függvény titkos, titkos_meret paramétereinek meghivásának eredményét.

A main függvényben a titkos fájl bufferbe való beolvasása történik meg, a maradék hely nullázása ezen belül, majd legeneráljuk az egybeágyazott forciklusokkal az összes lehetséges kulcsot. Az ittlévő példában csak maximum 3 hosszúságú kulcsot tudunk legenerálni, ez csupán a forciklusok számától, a KULCS_MERET-től függ. Ezzel a programmal akár 8 karakterhosszúságú kulcsot is le tudunk generálni a titkos szöveg tisztává alakitásának érdekében. Ez a forráskód ami fentebb látható, ez egy átalakitott, felgyorsitott változata az eredetinek. A módositás a **char str[3]={'e','k','y'}** bevezetését jelenti, amiben megadom hogy a kulcs milyen hosszúságú, és milyen karakterekből állhat. Ezt azért alkalmaztam, hogy ne kelljen sokat várni a törés lefutására, de ezt egyszerűen ki lehet venni a programból, ha nem ismerjük a kulcs elemeit és hosszúságát. Ekkor egyszerűen el kell tünteni az előbbi command-ot, valamint a forciklusokon belül a **kulcs[0]=str[ii] -t kulcs[0]=ii** -re kell cserélni.

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

4.6. 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...

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

int main( int argc, char **argv)
{
     png::image <png::rgb_pixel> png_image ( argv[1] );
    int size = png_image.get_width() *png_image.get_height();

    Perceptron* p = new Perceptron ( 3, size, 256, 1);
```

```
        double* image=new double[size];
        for( int i{0}; i<png_image.get_width(); ++i)
            for( int j{0}; j<png_image.get_height(); ++j )
                image[i*png_image.get_width() +j] = png_image[i][j ↩
                   ].red;

        double value = (*p) (image);

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

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

Videóból kilesett kód a példa.

A Perceptron fogalmát a mesterséges neurális hálók és mesterséges intelligencia témakörében találhatjuk meg. A lényege, hogy leellenőrzi a bemenetet és feltétel alapján dönti el, hogy mi legyen a kimenet. Feltételként vehetjük azt a példát, melyben három bemeneti adathoz várunk pozitiv egész számokat. Feltétel legyen az,hogy ha a három bemeneti adat közül kettő negativ akkor mondjuk a kimeneti adat -1 lesz. Viszont ha legalább kettő pozitiv akkor a kimeneti adat legyen 1.

Ez azt jelenti, hogy 1 a hibahatár, mivel egyet hibázhatunk legfeljebb, második hibánál már -1et dob ki a kimenetre, miközben pozitiv egész eredményt várunk. Ez a hibahatár sokkal nagyobb mennyiségű bemeneti adathoz érdekes, az előbb emlitett példában kevésbé, mivel nem lehet sokat csiszolni rajta. Nagy hibahatárt szoktak állitani a nagy mennyiségű bemeneti adathoz, és ezt csökkentik mindaddig, amig meg tudnak tartani egy bizonyos elfogadható hiba mennyiséget.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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

A Mandelbrot halmaz a komplex síkon

5.1. ábra. A Mandelbrot halmaz a komplex síkon

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

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

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

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

A feladat hasonló mint a C++-os verzióhoz csak itt abban különbözik hogy használjuk a complex headert ami nagy segítsév mert egy struktúrát spórol nekünk.

```
#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: ./complex 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";
```

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

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

}
```

A fő függvényben dekralálunk 2 változót, ha argumentumként jól adjuk meg ezeket, akkor ezeket átadja a változóknak, ha nem jól adjuk meg, akkor kiírjuk, hogy kell helyesen használni. Ezek után megadjuk a szélességet és a magasságot,ami ebbe az esetbe FullHD és az iterácós határt. Továbbá létrehozunk változókat, amik a kép elkészítéséhez kellenek majd. Az if függvény vizsgálja meg, az argumentum érvényességét, illetve itt adja át az előbb említett értékeket. Az else ág a rossz esetén, a segítséget írja ki. Ezek után lefoglaljuk a helyet a képnek. A dx, dy-hez hozzá rendeljük a megfelelő változókat. A forciklusban végig megyünk minden elemen és megadjuk a c változó értékét. Ekkor használjuk a complex-et, while ciklusban végezzük a számításokat, utánna rgb kóddal a pixeleket kiszinezzük.

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

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

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

Megoldás forrása:

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

```
// 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  ↩
   -0.019473810573093663922605855987058021128 18  ↩
   -0.019473810572541341845642648422654019668 7  ↩
   0.798505756933826860155534177465597167611 1  ↩
   0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913 1  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327099751215428 1  ↩
   0.213538705180497528912653137922461610287 4
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// 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;

}
```

A forrás elején konstansok definiálásával kezdjük. Ezekkel szabjuk meg a kép felbontását. Az első függvénnyel generáljuk le a képet, a png csomag segítségével. Létrehoztunk egy üres png-t. Forcikluson belül az rgb kóddal határottuk meg a pixeleket színét, majd az image.write-tal pedig a képet a megadott névvel kiküldjük. Ezután egy struktúrában 2 double változót vezetünk be, ezek lesznek a komplex számok struktúrája. A main-ben létrehozunk a konstanstok segítsévégel egy XxY-os tömböt. 3 integer és 2 double változó bevezetésével (dx, dy) amikkel a pixelekt fogjuk meghatározni. Ezután a c,z,zuj változóknak lefoglaljuk helyet, kiszámoljuk a dolgokat, azoknak az eredményét belerakjuk a tömbbe azt meghívjuk a függvényt amivel legeráljuk.

5.3. Biomorfok

A biomorf program a mandelbrot programkódját vesszi alapul. A mandelbrot halmaz tartarlmazza az összes ilyen halmazt. A program ugyanúgy bekéri a megfelelő bemeneteket, ha nem jó akkor kiírja. Ha jó, akkor a megfelelő változók megkapják a megfelelő értékeket. Ezután történik a kép létrehozása. Ugyan úgy megkapja a dx és dy az értéket. Aztán pedig a komplex számokat hozzuk létre. Megint végig megy a program minden ponton és ahol kell használjuk az rgb kódos színezést. A legvégén pedig kiküldjük a képet a kimenetre.

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

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

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

```
int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;
    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag =  atoi ( argv[3] );
        iteraciosHatar =  atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );
    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↩
            d reC imC R" << std::endl;
        return -1;
    }
    png::image < png::rgb_pixel > kep ( szelesseg, magassag );
    double dx = ( xmax - xmin ) / szelesseg;
    double dy = ( ymax - ymin ) / magassag;
    std::complex<double> cc ( reC, imC );
    std::cout << "Szamitas\n";
    for ( int y = 0; y < magassag; ++y )
    {

        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;
                if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                {
```

```
                    iteracio = i;
                    break;
                }
            }
            kep.set_pixel ( x, y,
                            png::rgb_pixel ( (iteracio*20)%255, (iteracio ↩
                               *40)%255, (iteracio*60)%255 ));
        }
        int szazalek = ( double ) y / ( double ) magassag * 100.0;
        std::cout << "\r" << szazalek << "%" << std::flush;
    }
    kep.write ( argv[1] );
    std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

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" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
//
// 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_Biomorphs_via_modified_iterations.pdf
//

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása:

A program azt fogja csinálni, hogy létrejön nekünk egy mandelbrot halmaz és az egérrel képesek vagyunk belenagyítani akár a végtelenségig a halmazba.

```
#include<QApplication>
#include "frakablak.h"
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  Frakablak w1,
  w2(-.08292191725019529, -.082921917244591272,
     -.9662079988595939, -.9662079988551172, 1200, 3000),
  w3(-.08292191724880625, -.0829219172470933,
     -.9662079988581493, -.9662079988563615, 1200, 4000),
  w4(.14388310361318304, .14388310362702217,
     .6523089200729396, .6523089200854384, 1200, 38655);
  w1.show();
  w2.show();
  w3.show();
  w4.show();
  return a.exec();
}
```

Ez a program nem elég önmagában, több forrásra van szükséunk. ilyen például a frakablak.h header. Egy mappába össze kell szednünk az összes forrást és telepítenünk kell ezt: "sudo apt-get install libqt4-dev". A qmake -project parancsal létrehozunk egy .pro fájlt. ebbe meg kell adnunk a QT+=Widgets parancsot a megfelelő helyre. Ez létrehoz egy fájlokat .o kiterjesztéssel és egy makefilet, ezek után make parancsal létrehozzuk a nagyítót. Ezek után kész is a programunk. A frakszal.cpp-ben készül el az ábránk amit majd nagyítani fogunk. Az rgb pixel színezést azonban már a frakablak végzi.

5.6. Mandelbrot nagyító és utazó Java nyelven

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

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

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

```
#include <iostream>
#include <math.h>
#include <pthread.h>
#include <ctime>


class PolarGen
{
public:
  PolarGen()
  {
    nincsTarolt = true;
    std::srand (std::time(NULL));
  }
  ~PolarGen()
  {
  }
  double kovetkezo();
private:
  bool nincsTarolt;
  double tarolt;
};
double PolarGen::kovetkezo ()
{
  if (nincsTarolt)
```

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

Ez a program véletlenszerűen fog számokat generálni nekünk. Azért csak véletlenszerűen, mert a véletlent nem lehet generálni, már ha egyáltalán úgy gondoljuk, hogy létezik véletlen. Ezt egy osztályon belül fogjuk kivitelezni. Az osztály neve a PolarGen-t kapta. Két részre tudjuk bontani. Van egy nyilvános és egy privát. A nyilvános részhez hozzá tudunk férni, viszont a privát részt, csak az osztályon belül tudjuk meghívni. Az osztály elején egyből ott van a koonstruktor ezt onnan tudjuk felismerni, hogy ugyan úgy hívjuk ahogyan az osztályt is. Ebben kezdő értékeket tudunk adni és egy objektum létrehozásával egyből lefut. Esetünkben most a "nincsTarolt" privát változó értékét fogja "True"-ra állítani és az srand is itt lesz, ami a véletlenszerű szám generálásához kell. Utánna van a destruktor ami ugyan úgy néz ki mint a konstruktor csak előtte van '~' jel. Ez a program végén fog lefutni. Ebben felszabadítjuk a memóriát. A privát részben létrehozunk egy logikai és egy double tipusú változót. A kovetkezo() függvény az, amiben a random számokat fogjuk létrehozni. Azt, hogy ezt hogyan végezzük matematikailag, most figyelmenkívül hagyjuk. A main függvényben meghívunk egy osztálytípusú változót. Ez fogja beindítani a konstruktort. Utánna pedig egy forciklusban tíz véletlen száot íratunk ki.

6.2. LZW

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

Megoldás videó:

A program a bemeneti adatokból egy bináris fát épít. Bármely típusú fa ábrázolható bináris fa segítségével. A bináris fa legfőbb jellemzője az, hogy bármelyik csomópontnak csak legfeljebb két utóda lehet. A bináris fák utódjait megkülönböztetjük aszerint, hogy bal illetve jobb részfák. A fa 0 és 1 számokból épül fel. A kitüntetett elem a gyökér. Innen minden elemet el tudunk érni. A következőben megnézzük, hogy is működik ez. Az eltérés, hog itt nem fő függvény van, hanem minden egy osztály része.

Megoldás forrása:

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

```
          fa = gyoker;
        }
      else
        {
          fa = fa->bal_nulla;
        }
    }
      else
    {
      if (fa->jobb_egy == NULL)
        {
          fa->jobb_egy = uj_elem ();
          fa->jobb_egy->ertek = 1;
          fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
          fa = gyoker;
        }
      else
        {
          fa = fa->jobb_egy;
        }
    }
    }
  printf ("\n");
  kiir (gyoker);

  extern int max_melyseg, atlagosszeg, melyseg, atlagdb;
  extern double szorasosszeg, atlag;
  printf ("melyseg=%d\n", max_melyseg-1);

  atlagosszeg = 0;
  melyseg = 0;
  atlagdb = 0;
  ratlag (gyoker);
  atlag = ((double)atlagosszeg) / atlagdb;
  atlagosszeg = 0;
  melyseg = 0;
  atlagdb = 0;
  szorasosszeg = 0.0;
 rszoras (gyoker);
  double szoras = 0.0;
  if (atlagdb - 1 > 0)
    szoras = sqrt( szorasosszeg / (atlagdb - 1));
  else
    szoras = sqrt (szorasosszeg);
  printf ("altag=%f\nszoras=%f\n", atlag, szoras);*/
  szabadit (gyoker);
}

int atlagosszeg = 0, melyseg = 0, atlagdb = 0;
void
```

```
ratlag (BINFA_PTR fa)
{
  if (fa != NULL)
    {
      ++melyseg;
      ratlag (fa->jobb_egy);
      ratlag (fa->bal_nulla);
      --melyseg;
      if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
    {
      ++atlagdb;
      atlagosszeg += melyseg;
    }
    }
}
double szorasosszeg = 0.0, atlag = 0.0;
void
rszoras (BINFA_PTR fa)
{
  if (fa != NULL)
    {
      ++melyseg;
      rszoras (fa->jobb_egy);
      rszoras (fa->bal_nulla);
      --melyseg;
      if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
    {
      ++atlagdb;
      szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
    }
    }
}
int max_melyseg = 0;
void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
    max_melyseg = melyseg;
      kiir (elem->jobb_egy);
      for (int i = 0; i < melyseg; ++i)
    printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
          ,
           melyseg-1);
      kiir (elem->bal_nulla);
      --melyseg;
    }
```

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

Magyarázat: Elsőnek is a szükséges headereket deklaráljuk. Ezek után pedig a Binfánk struktúráját. A struktúra egy egészet tartalmaz aminek a neve "ertek" és 2 mutatóval fog rendelkezni, amiben bal és jobb gyermekeket tároljuk majd.Az 1 érték jobbra fog kerülni, a 0 balra. A "typdef"-vel adunk neki egy nevet, amivel a programon belül fogjuk hívni. Ezután az uj_elem függvény következik. Ez fogja nekünk lefoglalni a tárhelyet a memóriában, ami "NULL" kezdőértékkel fog rendelkezni. Ha nincs memória, akkor hibát dob ki. A végén vissza adja a lefoglalt mutatót. Utánna függvény prototipusokat kapunk, ezek közül a feladatnak megfelelően csak a kiir() és a szabadit() függvényeket fogjuk megvizsgálni. Ugorjunk a main fő függvényre. Az első egy char tipusú változó, ebben fogjuk tárolni ideiglenesen a beolvasott karaktert. Aztán létrehozzuk a gyökérelemet és értékül adunk neki egy karaktert, jelen esetben '/'. A while ciklusban fog zajlani a faépítés. Először is megvizsgálja a beolvasott karaktert. Mindig a gyökér elemtől indul. Ha a beolvasott karakter értéke 0, akkor először megvizsgálja, hogy a gyökérnek vagy az adott csomópontnak van e bal_nullas gyermeke, ha van, akkor rálép a csomópontra, ha viszont nincs, akkor a gyökérnek vagy az adott csomópontnak létrehoz egy bal_nullas gyermeket. Ha a beolvasott karakter értéke 1, akkor a program ugyan ezen az elven mint a 0-ás értéknél végig vizsgálja, csak a jobb_egyes gyermekkel. Most következik a kiír és a szabadit függvény. A szabadit() függvény egy rekurzív függvény. Törli a memóriából az eltárolt elemeket. A kiir() függvény is rekúrzív függvény. Bejárja a fa elemeit.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

Inorder fabejárás: Az inorder fabejárásnál először a bal oldalt vizsgáljuk meg, utánna jön a gyökér és legvégül pedig a jobb oldalt nézzük.

```
//inorder fabejárás:
void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
  kiir (elem->bal_nulla);
  printf ("%c", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek);
  kiir (elem->jobb_egy);
```

```
    }
}
```

A postorder fabejárás: Itt először a fa bal oldalát fogja átvizsgálni, aztán a jobb oldalt, legutoljára pedig a gyökeret vizsgáljuk.

```
//postorder fabejárás:
void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
  kiir (elem->bal_nulla);
  kiir (elem->jobb_egy);
  printf ("%c", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek);
    }
}
```

Preorder: Itt mindig a gyökérrel kezdi a program a vizsgálatot, aztán a bal oldalt legvégezetül, pedig a jobb oldalt fogja bejárni.

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

Az alábbi program a fenti C változatnak lesz úgymond a C++ változata. Az első lépés, hogy ami C-ben struktúra volt, azt átírjuk C++-ban egy osztályba, mivel a C++-ban megtehetjük. Ez az alábbi módon fog kinézni:

```
        class LZWBinFa
        {
        public:
        LZWBinFa (char b = '/'):betu (b), balNulla (NULL), jobbEgy (NULL)  ↩
           {};
        ~LZWBinFa () {};
            void operator<<(char b)
    {
        if (b == '0')
        {
            // van '0'-s gyermeke az aktuális csomópontnak?
            if (!fa->nullasGyermek ()) // ha nincs, csinálunk
            {
                Csomopont *uj = new Csomopont ('0');
                fa->ujNullasGyermek (uj);
                fa = &gyoker;
```

```
            }
            else // ha van, arra lépünk
            {
                fa = fa->nullasGyermek ();
            }
        }
        else
        {
            if (!fa->egyesGyermek ())
            {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }
```

Ezen belül fogjuk a gyökeret létrehozni és értékül adni neki a '/'-t. A mutatóinak értékét 0-ra állítjuk. Ezek után jön a faépítés a már fentiekben elmagyarázott módon. A program megnézi, hogy 1-est vagy 0 érkezik a bemenetről. Itt azt látjuk, hogy a betétel a << operátorral történik, ez annyiban különbözik a C-ben írt programtól, hogy ez egyből beleteszi a fába a beérkezett karaktert. Egy új csomópontot a "new" szóval tudunk létrehozni, ha szükséges. Ez azért lehetséges mert van egy Csomópont osztályunk (Lásd a lenti programban). A Class csomóponton belül az egyesGyermek() és a nullasGyermek() függvények a gyermekükre mutató pointereket fogják tartalmazni. Az ujNullasGyermek és au ujEgyesGyermek-nek pedig adunk egy gyermeket ás arra fogja állítani a mutatót. A private részben fogjuk ezeket deklarálni, ez azt jelenti, hogy csak az osztályon belül használhatóak ezek a változók. A legvégén jön a main főfüggvény. Itt deklaráljuk a char tipusú változót amibe beolvasunk és innen kerül az osztályokhoz. Végül meghívjuk a kiir és a szabadit függvényeket amire példát az előző programokban találunk. Ugye a kiir()-al kiíratjuk az eredmény és a szabadit()-al pedig felszabadítjuk a lefoglalt meróriát.

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

```
        }
        void ujEgyesGyermek (Csomopont * gy)
        {
            jobbEgy = gy;
        }
    private:
        friend class LZWBinFa;
        char betu;
        Csomopont *balNulla;
        Csomopont *jobbEgy;
        Csomopont (const Csomopont &);
        Csomopont & operator=(const Csomopont &);
    };

  int main ()
  {
        char b;
        LZWBinFa binFa;
        while (std::cin >> b)
        {
        binFa << b;
        }
    binFa.kiir ();
    binFa.szabadit ();
    return 0;
    }
```

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

Vegyük alapul a C++ ban megírt LZWBinfa-t. Első dolgunk, hogy a fában a gyökér elemet átalakítjuk egy mutatóvá. Azt az alábbi módon fogjuk megcsinálni: A gyökér elem a protected részén van az osztálynak. Itt az eredeti "Csomopont gyoker;" az alábbi módon átírunk:

```
protected:
Csomopont *gyoker;
int maxMelyseg;
double atlag, szoras;
```

Ugye C++ ban a mutatót egy '*'-al jelüljuk. Ha most futtatnánk a programot, akkor számtalan hibába ütköznénk. Ezeket ki kell javítanunk. A programban így nem a gyökér memóriacímét kell átadnunk (Töröljük az összes referenciajelet a gyokerek előtt) és mivel mutató lett a gyökér így nem '.'-al hiavatkozunk hanem '->'-al. Itt láthatunk példát arra, hogy hogyan:

```
//előtte:
fa=&gyoker;
//utánna:
fa=gyoker;

//előtte:
szabadit (gyoker.egyesGyermek ());
szabadit (gyoker.nullasGyermek ());
//utánna:
szabadit (gyoker->egyesGyermek ());
szabadit (gyoker->nullasGyermek ());
    }
```

Ha mindezek után lefuttatjuk a programunkat az lefordul, azonban futtatáskor szegmentális hibába ütközünk. Ez azért van, ugyanis a gyökér memóriacíme nincs lefoglalva. Ennek a megoldását a konstruktorban és a destruktorban fogjuk megalkotni. A konstruktorban foglaljuk le és a destruktorba fogjuk törölni a lefoglalt memóriát. Lásd:

```
LZWBinFa ()
    {
    gyoker= new Csomopont('/');
    fa = gyoker;
    }
~LZWBinFa ()
    {
    szabadit (gyoker->egyesGyermek ());
    szabadit (gyoker->nullasGyermek ());
    delete(gyoker);
    }
```

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

```
       LZWBinFa ( const LZWBinFa & regi ) {

          gyoker.ujEgyesGyermek ( masol ( regi.gyoker.egyesGyermek (), regi ↩
             .fa ) );
          gyoker.ujNullasGyermek ( masol ( regi.gyoker.nullasGyermek (),  ↩
             regi.fa ) );

          if ( regi.fa == & ( regi.gyoker ) )
               fa = &gyoker;
```

```
    }

    LZWBinFa ( LZWBinFa && regi ) {

         gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
         gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

         regi.gyoker.ujEgyesGyermek ( nullptr );
         regi.gyoker.ujNullasGyermek ( nullptr );

    }
```

```
      Csomopont *  masol ( Csomopont * elem, Csomopont * regifa ) {

         Csomopont * ujelem = NULL;

         if ( elem != NULL ) {
              ujelem = new Csomopont ( elem->getBetu() );

              ujelem->ujEgyesGyermek ( masol ( elem->egyesGyermek (),  ↩
                 regifa ) );
              ujelem->ujNullasGyermek ( masol ( elem->nullasGyermek (),  ↩
                 regifa ) );

              if ( regifa == elem )
                   fa = ujelem;

         }

         return ujelem;
    }
```

A forráskódban a binfa működése gyakorlatilag nem változik semmit, ezért arról nem is írnék részletesebben. A mozgató szemantika titka viszont annyiban merül ki, hogy a először is szükségünk van egy másoló konstruktorra, ami egy olyan függvény lesz, ami megkapja a binfa elemeit és ezeket új memóriacímen új jobb és bal elemekként lementi. Ezzel építünk gyakorlatilag egy új fát, aminek minden eleme megegyezik az eredeti fánknak az elemeivel. Amikor ez megtörténik, régi fánknak a gyökerének a pointereit átállítjuk null pointerekre, ezzel pedig töröltük azoknak az elemeit és már csak az új fánk létezik.

A fánk mozgatását a main-ben a std::move függvénnyel fogjuk elérni, ami magától nem fogja mozgatni a binfánkat, csakis akkor, ha ehhez meg van írva már a mozgató konstruktorunk.

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

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

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat...

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Bármilyen progromzási nyelvben vannak feldolgozandó objektumok amiket változók és konstansok alkotnak. Ezeket a változókat/konstansokat deklarálással adjuk meg amikor meg kell adni a nevüket és típusukat. Hogy a gép tudja mit tegyen az adatokkal arra különböző operátorokat használunk. (+,-,*,/,%). Deklaláráskor a legfontosabb lépés a helyes típus kiválasztása mivel ez alapján tudjuk hogy a változó milyet értékeket kaphat, tartalmazhat illetve milyeneket adhat vissza. Például egy

```
char
```

típusú változó csak 1 karaktert tartalmazhat, vagyis ha nagyobb szöveggel szeretnénk dolgozni más típust kell választanunk. Például hosszabb szövegek tárolására a string típust használják amit C nyelven

```
char string[]
```

így deklarálunk. Egy string nevű változóba (nincs definiálva hogy hány karaktert) így hosszabb szöveget is tárolhatunk.

A könyv segítségével a C programozási nyelvet tudjuk jobban megérteni.

1983-ban az ANSI általl létrehozott bizottság azt kapta feladatul, hogy definiálják a modern C nyelvet átfogóan. A bizottság 1988-ban be is fejezte a definíció elkészítését amit röviden ANSI C-nek hívnak.

Ezek után a nyelvbe különféle minősítők kerültek amik a programozók munkáját voltak hivatottak megkönnyíteni. Ilyen minősítők voltak a "signed" és ennek párja az "unsigned" amivel egy változó előjelét lehetett meghatározni.

Bevezettek új adattípusokat is. Például

```
        short double
```

adattípust. Ennek párja a "long". A különbség a két típust között az, hogy a "short" csak 16 biten míg a "long" 32 biten tudja tárolni az adatot.

A C nyelvben vannak alapvető adat típusok. Ilyen adat típusok a char (egy karaktert tárol) az int (egész számot tárol) a float (lebegőpontos szám, egyszeres pontosságú) és a double (lebegőpontos szám, kétszeres pontosságú).

Az int adat típusnak van 2 fajta minősítője amivel megadhatjuk, hogy az int-nek milyen értékei lehetnek. Ez a két minősítő a "short" és a "long", short int értékei -32,767től akár +32,767-ig lehet, míg a long int értékei -2,147,483,647-től akár +2,147,483,647-ig lehet. Azért ilyen nagy a különbség a "short" és a "long" között mivel a "short" 16 míg a "long" 32 biten képes tárolni adatot.

10.3. Programozás

[BMECPP]

A C++ egy objektum orientált programozási nyelv ami a C nyelv családba tartozik. Nagyon elterjedt nyelv, több program alapja a mai napig.

Igaz hogy a C++ alapnak a C-t veszi de sok köztük a különbség is. Ilyen különbség mondjuk a bool.h szüksége a C-ben míg C++-ben nem.

```
                #include <stdbool.h>
                int main()
                    {
                    bool igaz=true;
bool hamis=false;
                    return 0;
                    }
            ]
```

```
                int main()
                    {
                    bool igaz=true;
bool hamis=false;
                    return 0;
                    }
```

Egy C++ újjításnak köszönhetően két függvény azonos névvel képes futni ha a paraméterezésük különböző.

```
            float f(int a)
        ]
```

```
            float f(double a)
        ]
```

A fenti program tehát gond nélkül letud futni mivel a két függvénynek igaz hogy ugyan olyan nevük van viszont mivel más a paraméter listájuk.

III. rész

Második felvonás

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

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

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

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