

Univerzális programozás

Így neveld a programozód!

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

Copyright © 2019 Dr. Bátfai Norbert

Copyright (C) 2019, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

<http://gnu.hu/fdl.html>

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Bátfai, Margaréta, Ács Semendi, Ádám-István	2020. április 10.	

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

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	8
2.3. Változók értékének felcserélése	10
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	11
2.6. Helló, Google!	12
2.7. A Monty Hall probléma	14
2.8. 100 éves a Brun tétel	15
3. Helló, Chomsky!	20
3.1. Decimálisból unárisba átváltó Turing gép	20
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	20
3.3. Hivatkozási nyelv	20
3.4. Saját lexikális elemző	21
3.5. Leetspeak	22
3.6. A források olvasása	24
3.7. Logikus	25
3.8. Deklaráció	26

4. Helló, Caesar!	30
4.1. double ** háromszögmátrix	30
4.2. C EXOR titkosító	32
4.3. Java EXOR titkosító	34
4.4. C EXOR törő	35
4.5. Neurális OR, AND és EXOR kapu	38
4.6. Hiba-visszaterjesztéses perceptron	38
5. Helló, Mandelbrot!	40
5.1. A Mandelbrot halmaz	40
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	41
5.3. Biomorfok	44
5.4. A Mandelbrot halmaz CUDA megvalósítása	44
5.5. Mandelbrot nagyító és utazó C++ nyelven	44
5.6. Mandelbrot nagyító és utazó Java nyelven	45
6. Helló, Welch!	46
6.1. Első osztályom	46
6.2. LZW	48
6.3. Fabejárás	48
6.4. Tag a gyökér	48
6.5. Mutató a gyökér	48
6.6. Mozgató szemantika	49
7. Helló, Conway!	55
7.1. Hangyaszimulációk	55
7.2. Java életjáték	55
7.3. Qt C++ életjáték	55
7.4. BrainB Benchmark	56
8. Helló, Schwarzenegger!	57
8.1. Szoftmax Py MNIST	57
8.2. Mély MNIST	57
8.3. Minecraft-MALMÖ	57

9. Helló, Chaitin!	58
9.1. Iteratív és rekurzív faktoriális Lisp-ben	58
9.2. Gimp Scheme Script-fu: króm effekt	58
9.3. Gimp Scheme Script-fu: név mandala	58
10. Helló, Gutenberg!	59
10.1. Programozási alapfogalmak	59
10.2. Programozás bevezetés	59
10.3. Programozás	59
10.4. Programozás, Python	60
III. Második felvonás	61
11. Helló, Arroway!	63
11.1. A BPP algoritmus Java megvalósítása	63
11.2. Java osztályok a Pi-ben	63
IV. Irodalomjegyzék	64
11.3. Általános	65
11.4. C	65
11.5. C++	65
11.6. Lisp	65

Ábrák jegyzéke

2.1. A B_2 konstans közelítése	19
4.1. A double ** háromszögmátrix a memóriában	32
5.1. A Mandelbrot halmaz a komplex síkon	40

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 (hogyan 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 dlatex 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
dlatex 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 'dlatex' 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 találsz 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.
- Kódjátzsma, <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.

DRAFT

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.

DRAFT

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-nl>

Videó: <https://www.youtube.com/watch?v=bl-yjJ58zLc>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c, bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozo/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 példánkban 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 százalékbán:

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

    return 0;
}
```

Minden mag 100 százalékbán:

```
#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



Werkfilm

- <https://youtu.be/lvmi6tyz-nl>

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 forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a Lefagy függvényt, amely tetszőleges programról el tudja dönteni, hogy van-e benne végtelen 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 építő Lefagy2 már nem tartalmaz feltételezett, csak konkrét kódot:

```
Program T1000
{
    boolean Lefagy(Program P)
    {
        if(P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

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

    main(Input Q)
    {
        Lefagy2(Q)
    }
}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Alan Turing, angol matematikus, a Turing gép feltalálója is vizsgálta már ezt a problémát. Neki sikerült bebizonyítania, hogy nem létezik olyan algoritmus, aminek segítségével egy gép meg tudja állapítani egy másik gépről, hogy az le fog e fagyni, végtelen ciklusba fog e lépni.

2.3. Változók értékének felcserélése

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

Megoldás videó: <https://www.youtube.com/watch?v=E-A9iyUPRrA>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/csere.cpp

```
int a,b;  
cin>>a;  
cin>>b;
```

Ebben a kódrészletben deklarálom a két változót, a-t és b-t, majd bekérek a felhasználótól két értéket.

```
a=a*b;  
b=a/b;  
a=a/b;
```

Szorzást használva felcserélem az értékeket. Például ha $a=2$ és $b=3$, akkor:

1.sor: Az a értéke $2*3=6$ lesz.

2.sor: A b értéke $6/3=2$ lesz.

3.sor: Az a értéke $6/2=3$ lesz.

Tehát az értékek felcserélődtek.

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés használata nélkül írd 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 Ifekkel: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/pattog.cpp

```
int x = 0;  
int y = 0;
```

Megadom az x és y koordinátákat, innen fog indulni a labda

```
int iranyx = 1;  
int iranyy = 1;
```

Megadom az irányokat, ezek fogják befolyásolni a labda haladásának irányát és sebességét. Ha a szám pozitív, a labda koordinátái nőnek, ha negatív akkor pedig csökkennek. Minél nagyobb a szám, annál gyorsabban fogja elérni a terminál oldalát.

```
int sor;  
int oszlop;
```

Ezzel a két változóval fogom biztosítani, hogy a labda ne hadja el a terminál területét

```
while (true) {  
  
    getmaxyx ( ablak, oszlop , sor );  
  
    mvprintw ( y, x, "X" );  
  
    refresh ();  
    usleep ( 1000000 );  
  
    x = x + iranyx;  
    y = y + iranyy;  
  
    if ( x>=sor)  
        iranyx *=-1;  
  
    if ( !(x>0) )  
        iranyx *=-1;  
  
    if ( !(y>0) )  
        iranyy *=-1;  
  
    if ( y>=oszlop )  
        iranyy *=-1;  
  
}
```

A getmaxyx függvény segítségével meg tudom állapítani a terminál méretét, míg a mvprintw függvény a labda útvonalát fogja jelezni.

A refres() usleep() parancsokkal beállítom, milyen sűrűn szeretném frissíteni a képernyőt, ami befolyásolja a labda sebességét.

A következő két sorban a koordinátákat fogom megváltoztatni, úgy, hogy hozzáadom az irány változót.

Az if-ek segítségével követem nyomon, hogy a labda elérte e a terminál szélét, és ha igen, az irány változó értékét megszorozom (-1)el, hogy visszafelé haladjon a labda.

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>, [Futás közben](#).

Megoldás forrása Bogomips: bmax/thematic_tutorials/bmax_textbook_IgyNeveldaProgramozasTuring/bogomips.c

Megoldás forrása Szóhossz: bmax/thematic_tutorials/bmax_textbook_IgyNeveldaProgramozasEltolas.cpp

```
int szo=1,db=1;
```

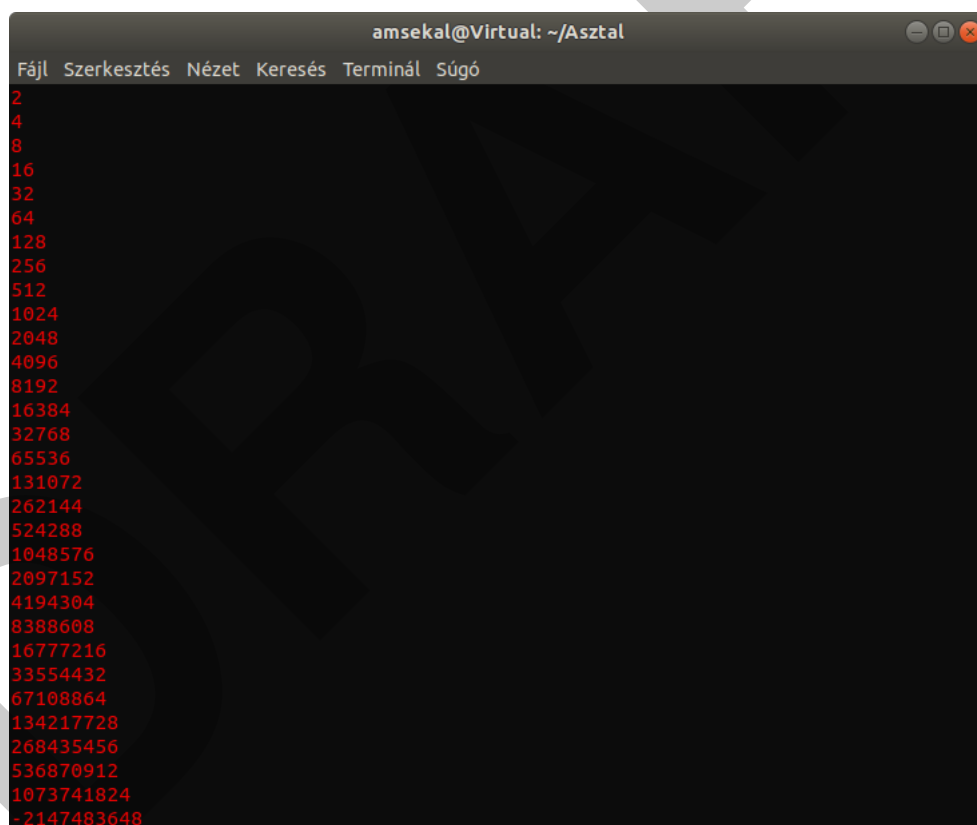
Két változót fogok használni: a szo változóban lesz az a szám, amit binárisan el fogok tolni, míg a db-ben fogom tárolni az eltolások számát.

Ezek után egy while ciklusban bináris shiftelés segítségével megnétem a szóhosszt. A db változót azért indítottam 1-től, mert a ciklusba nem a szo változó értékével kezdünk, hanem annak az eggyel eltolt változatával. Ezt az eltolást addig csinálom, amíg pozitív számokat kapok.

Példa bináris eltolásra:

Az 1 bináris értéke 1, ha ezt binárisa eggyel eltoljuk, az eredmény 10, ami a 2-nek felel meg. Ha még egy eltolást végzünk, a kapott szám 100 lesz, ami a 3-nak felel meg.

A program által elvégzett eltolások eredményei:



```
amsekal@Virtual: ~/Asztal
Fájl Szerkesztés Nézet Keresés Terminál Súgó
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
2147483648
```

2.6. Helló, Google!

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

Megoldás videó: https://youtu.be/9C_KZFyODLY

Megoldás forrása : bhex/thematic_tutorials/bhex_textbook_IgyNeveldaProgramozod/Turing/pag.c

A PageRank az internetes weboldalakot hivatott "rangsorolni", ezt a rendszert használja például a Google is. Az algoritmus lényege az, hogy végignézze egy oldalra hány hyperlink mutat, ugyanis az a koncepció, hogy egy 'A' oldalon csak akkor van egy másik 'B' oldalra vezető hyperlink, ha a 'B' oldalt jónak tartja az 'A' oldal szerkesztője. Tehát a hyperlink felfogható egyfajta szavazatként is.

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

void
PageRankKiiras (double tomb[], int db)
{
    int i;
    for (i=0; i<db; i++)
        printf("PageRank [%d]: %lf\n", i+1, tomb[i]);
}

double tavolsag(double pr[], double pr_temp[], int db)
{
    double tav = 0.0;
    int i;
    for(i=0; i<db; i++)
        tav += (pr[i] - pr_temp[i]) * (pr[i] - pr_temp[i]);

    return sqrt (tav);
}

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

    double PageRank[4] = {0.0, 0.0, 0.0, 0.0};
    double Temp_PageRank[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};

    long int i, j;
    i=0; j=0;

    for (;;)
    {
        for(i=0; i<4; i++)
            PageRank[i] = Temp_PageRank[i];
        for (i=0; i<4; i++)
        {
            double temp=0;
```

```
for (j=0; j<4; j++)
temp+=Lapok[i][j]*PageRank[j];
Temp_PageRank[i]=temp;
}

if ( tavolsag(PageRank,Temp_PageRank, 4) < 0.000001)
break;
}
PageRankKiiras (PageRank,4);
return 0;

}
```

A programban található mátrix mutatja meg nekünk, hogy melyik oldal melyik oldalra mutat:

Az első oszlop megmutatja, hogy az 'A' oldal önmagára és a 'D' weboldalra mutat, a 'B' oldal önmagára, a 'C' és a 'D' oldalra mutat, stb.

A PageRank, kezdetben üres tömb, szerepe az eredmények eltárolása lesz, a Temp_PageRank pedig, nevéből is ítélhetően ideiglenes eredményeket fog eltárolni, amiket majd később felhasználunk. Minden lap ugyanakkora értékkel indul, ezek fognak egy ciklusban egészen addig változni, amíg nem teljesül a ciklus megtörésének feltétele. Ha a ciklusnak vége, akkor megvannak az eredmények, amit a külön erre a feladatra szánt alprogrammal ki is íratunk.

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

A Monty Hall paradoxon először az amerikai Let's Make a Deal nevű műsorban került bemutatásra, nevét is a műsor vezetőjéről kapta. Maga a paradoxon alaphelyzete nagyon egyszerű:

A játékos három ajtó közül választhatott: 2 ajtó mögött egy-egy kecske rejtőzött, míg a harmadik ajtó mögött egy autó.

Miután a játékos választott, a műsorvezető kinyitott egy ajtót, ami mögött egy kecske rejtőzött.

Ezek után a játékosnak lehetősége nyílt válztatni a döntésén.

A józan ész azt diktálja, hogy teljesen mindegy, hogy változtat-e a döntésén a játékos, 50 százaléka lesz nyerni. Azonban ez a gondolatmenet nem helyes, és ez a tény egy program segítségével is kimutatható. Míg a választás előtt minden ajtónak 1/3 esélye van, hogy ő rejtse az autót, miután a műsorvezető kinyitotta az egyik vesztes ajtót, a játékos által választottnak továbbra is 1/3 esélye van, de a másik zárt ajtónak már 2/3.


```
kiserletek_szama=10000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

  if(kiserlet[i]==jatekos[i]){

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

  }else{

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

  }

  musorvezeto[i] = mibol[sample(1:length(mibol),1)]

}

nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

  holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvalt[sample(1:length(holvalt),1)]

}

valtoztatesnyer = which(kiserlet==valtoztat)
```

A fenti kódban is pontosan ezt vizsgáljuk. Abban az esetben, ha a játékos jól választ, a nemvaltozasnyer lesz a helyes, míg ha a változtatás után fog a helyes ajtóra mutatni, akkor a valtoztatesnyer.

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 $1*2*3*\dots*(n-1)*n*(n+1)$ szorzatot, aminek a neve $(n+1)$ faktoriális, jele $(n+1)!$.

Majd vizsgáljuk meg az a sorozatot:

$(n+1)!+2, (n+1)!+3, \dots, (n+1)!+n, (n+1)!+(n+1)$ ez n db egymást követő szám, ezekre (a jól ismert bizonyítás szerint) rendre igaz, hogy

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

tehát ebben a sorozatban egy prím nincs, akkor a $(n+1)!+2$ -nél kisebb első prím és a $(n+1)!+(n+1)$ -nél nagyobb első prím 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 $(1/3+1/5)+(1/5+1/7)+(1/11+1/13)+\dots$ 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](https://github.com/bhax/attention_raising/Primek_R/stp.r) nevű állománya kiszámolja az ikerprímeket, összegzi a reciprokaikat és vizualizálja a kapott részeredményt.

```
# Copyright (C) 2019 Dr. Norbert Bاتفai, 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
  rtlplust2 = 1/t1primes+1/t2primes
  return(sum(rtlplust2))
}

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

Soranként értelmezzük 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 képz, 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-ből 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 képi:

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

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

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

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

```
> 1/t1primes+1/t2primes  
[1] 0.53333333 0.3428571 0.1678322
```

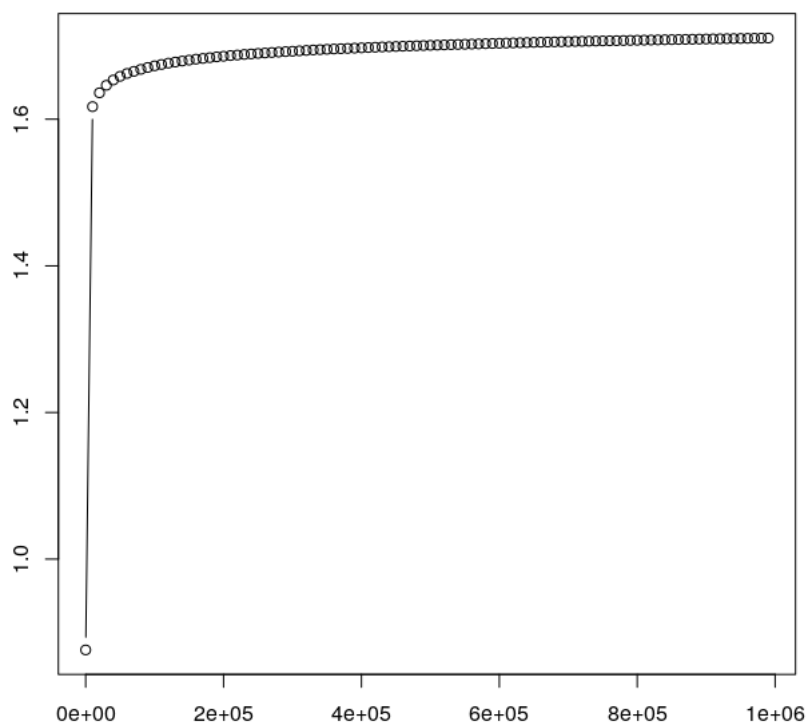
Nincs más dolgunk, mint ezeket a törtet összeadni a `sum` függvénnyel.

```
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 gráfiával 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.

Megoldás videó: <https://www.youtube.com/watch?v=KAhxMUW1rVc>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/hivatkozas.c

Először írok egy for ciklust.

```
%{  
#include <stdio.h>  
  
int main()  
{int s=0;  
for ( int i=0;i<5;i++)  
s++;  
printf("Vege\n");  
return 0;  
}
```

A leírt program nem fut le C98-as szabványt használva, mivel ebben a szabványban még nem volt lehetséges ciklusváltozó deklarálása. Ha ezt a kódot szeretnénk C89-es szabvánnyal futtatni, az alábbi módon lehetne:

```
%{ #include <stdio.h>  
  
int main()  
{int i, s=0;  
for ( i=0;i<5;i++)  
s++;  
printf("Vege\n");  
return 0;  
}
```

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). Futás közben: https://www.youtube.com/watch?v=_y

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.1)

A használt program 3 részből áll:

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

Az első rész sima C kód, amelyben deklarálunk egy realnumbers nevű változót, ami a valós számok előfordulását fogja számolni. Ezt a változót 0-tól indítjuk.

```
%{  
digit [0-9]  
%%  
{digit}* (\. {digit}+)? {++realnumbers;
```

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

A második részben már nem csak C kód található. Itt megadjuk a keresési feltételeket: egy olyan szövegrészletet keresünk, ahol 0 vagy több számjegy van (ezt jelzi a digit *), amit egy pont követ, ami után 1 vagy annál több számjegyet találunk. A pont előtti \ azt a célt szolgálja, hogy a program felismerje, hogy mi a pontra, mint karakterre gondolunk, nem egy bármilyen karakterre. Ezek után ismét C kód következik, ezzel kiíratjuk a szövegbe, ha találunk egy valós számot, és növeljük a realnumbers változó értékét eggyel.

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

A harmadik kódrészletben hívjuk meg a lexikális elemzőt, ami az általunk megadott feltételek alapján fog keresni a szövegben. Egy lexikális elemző nagyon hasznos tud lenni, ugyanis a segítségével a programunknak nem betűnként kell beolvasnia a forrást, így sok időt és erőforrást tudunk megspórolni.

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

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

Futás közben: <https://www.youtube.com/watch?v=-OrREI6GZKI>

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/1337d1c7.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/IgyNeveldaProgramozod/Chomsky/1337d1c7.1)

```
%{  
    #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;
}
```

A leet nyelv lényege, hogy a betűket rájuk hasonló számmal vagy más karakterekkel helyettesítsünk, például az a helyett használjuk: 4, @, /-\. A program hasonlít az előző feladathoz, azzal a különbséggel, hogy most nem számokat keresünk, hanem betűket. Létrehozunk egy tömböt, ahol eltároljuk a betűket és leet megfelelőiket. Ezután egy for ciklus segítségével megkeressük a betűt ebben a tömbben, és egy randomizáló rész segítségével véletlenszerűen választunk egy leet megfelelőt a megadottakból. Ha egy betűnek nincs leet megfelelője, változtatás nélkül kiíratásra kerül.

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelő)==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 jelkezelő 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 megránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

```
if(signal(SIGINT, SIG_IGN) != SIG_IGN)
    signal(SIGINT, jelkezelő);
//INTERACT signal esetén a jelkezelő eldönti, hogyan reagáljon a ←
    program
```

ii.

```
for(i=0; i<5; ++i)
//Inditunk egy for ciklust, ahol i 0-tól indul, és a ciklusban lévő ←
    utasítások elvégzése előtt nő az értéke.
```

iii.

```
for(i=0; i<5; i++)
//For ciklus, ahol az i értéke a ciklusban lévő utasítások elégzése ←
    után nő
```

iv.

```
for(i=0; i<5; tomb[i] = i++)
// Egy for ciklus, ahol egy tömbi i-edik eleme felveszi az i+1 értéket ←
    , és az i nő a ciklus végén
```

v.

```
for(i=0; i<n && (*d++ = *s++); ++i)
//Egy for ciklus, ami addig megy amíg i kisebb mint n és a d pointer ←
    egyenlő az s pointerrel.
```

vi.

```
printf("%d %d", f(a, ++a), f(++a, a));
//Kiirunk két számot, amit az f függvény állít elő.
```

vii.

```
printf("%d %d", f(a), a);
//Kiirunk két számot, az egyiket az f függvény állítja elő az a ←
    változó segítségével, a másik szám pedig az a változó.
```

viii.

```
printf("%d %d", f(&a), a);
//Kiir két számot, az egyiket az f függvény segítségével, a másik ←
    pedig az a értéke lesz, ami ha a függvényben megváltozott, a ←
    változás megmarad.
```

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\forall x \exists y ((x < y) \wedge (y \text{ prim})))$  
$(\forall x \exists y ((x < y) \wedge (y \text{ prim}) \wedge (\exists z (y < z \wedge z \text{ prim}))) \leftrightarrow$  
  )$  
$(\exists y \forall x (x \text{ prim}) \supset (x < y))$  
$(\exists y \forall x (y < x) \supset \neg (x \text{ prim}))$
```

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

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

1. Minden x esetén igaz, hogy van olyan y , amely nagyobb mint x és primszám.
2. Minden x esetén igaz, hogy van olyan y , amely nagyobb mint x , primszám és az $y+2$ is primszám.
3. Létezik olyan y , amely minden x primnél nagyobb
4. Létezik olyan y , amelynél egyik x sem kisebb és prim.

3.8. Deklaráció

Megoldás videó: <https://www.youtube.com/watch?v=rjaIqjzDVC4>

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/deklaracio.cpp](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/deklaracio.cpp)

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- `int a; // integer típusú változó`
- `int *b = &a; // b pointer, ami a memóriacímére mutat`
- `int &r = a; // r hivatkozás a-ra`
- `int c[5]; // c tömb`
- `int (&tr)[5] = c; // tr tömb referencia c tömbre`
- `int *d[5]; // d pointerekből álló tömb`
- `int *h (); // egészre mutató visszaadó függvény`
- `int *(*l) (); // mutatóra mutató függvény`
- `int (*v (int c)) (int a, int b) // Függvénytmutató, ami egy egészet ↵
visszaadó függvényre mutató mutatóval visszatérő függvény`
- `int (*(z) (int)) (int, int); // Függvénytmutató, ami egy egészet visszaadó ↵
függvényre mutató mutatót visszaadó függvényre mutat`

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összehasonlítása 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](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c)

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

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

Meghatározom, hogy a háromszögmátrix, amit használni fogok, 5 soros legyen és deklarállok egy tm ponttert, ami pointerre fog mutatni

```
#include <stdio.h>
if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
{
    return -1;
}
}
```

Ezzel az iffel megpróbálok helyet foglalni a memóriában a tm-nek, ha ez nem sikerül, -1-et fogunk visszatenni, ez fogja számunkra jelezni, hogy a művelet nem sikerült

```
#include <stdio.h>
for (int i = 0; i < nr; ++i)
{
```



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

Hasonlóan az előző ifhez, helyet foglalunk a tm-ben a pointereknek, ha ez nem sikerül, a visszatérési érték -1 lesz.

```
#include <stdio.h>  
for (int i = 0; i < nr; ++i)  
    for (int j = 0; j < i + 1; ++j)  
        tm[i][j] = i * (i + 1) / 2 + j;  
}
```

Étréket adunk az alsó háromszögmátrixunknak

```
#include <stdio.h>  
for (int i = 0; i < nr; ++i)  
{  
    for (int j = 0; j < i + 1; ++j)  
        printf ("%f, ", tm[i][j]);  
    printf ("\n");  
}
```

Kiiratjuk az alsó héromszögmátrixunkat

```
#include <stdio.h>  
tm[3][0] = 42.0;  
(* (tm + 3)) [1] = 43.0;  
*(tm[3] + 2) = 44.0;  
* (* (tm + 3) + 3) = 45.0;  
}
```

Értéket adunk a negyedik sor első (42), második (43), harmadik(44) és ötödik(45) elemének. A különbség, hogy az első esetben konkrétan megmondjuk melyik elem értékét változtatjuk, a többi esetben a pointerek mozgatózásának segítségével változtatunk az értéken.

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

```

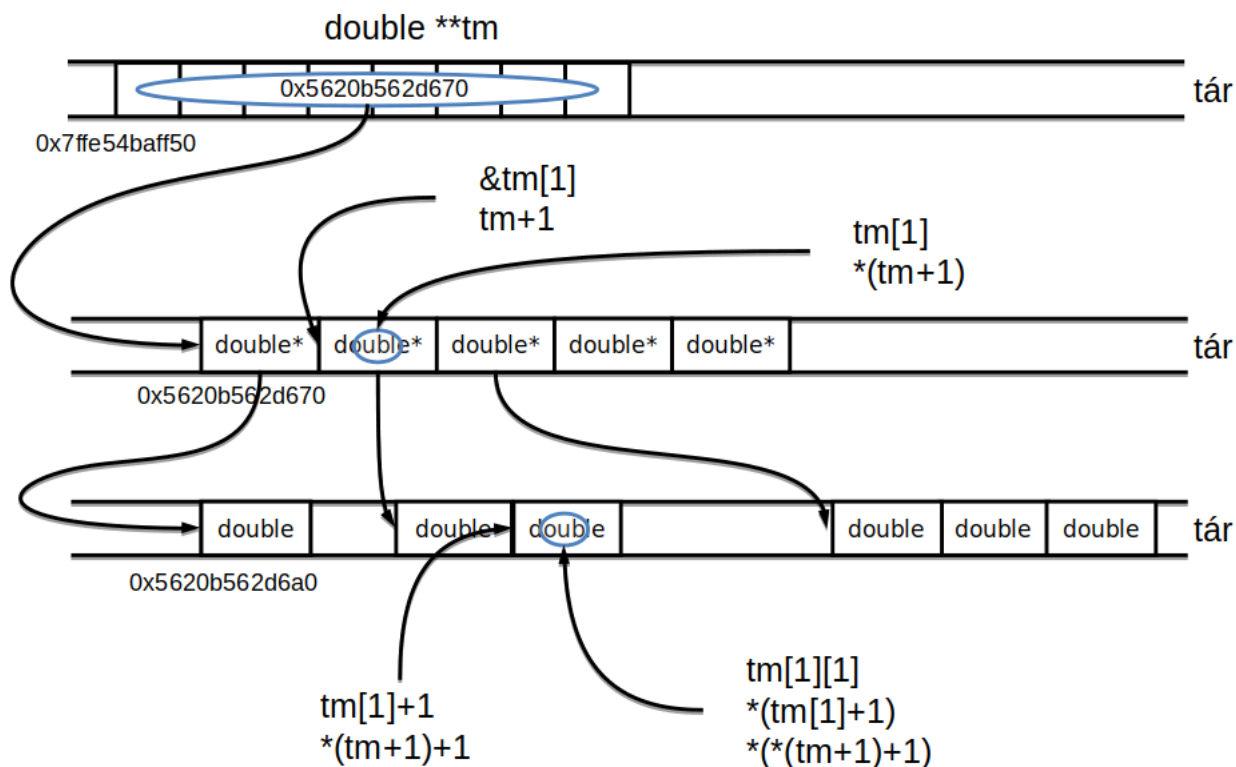
    free (tm[i]);

    free (tm);

    return 0;
}

```

Kiiratjuk a mátrixot az új értékekkel, majd felszabadítjuk a tárhelyet.



4.1. ábra. A `double **` háromszögmátrix a memóriában

A `tm` különlegessége, hogy egy olyan vektorra mutat, amiben szintén pointerek vannak. Ahogy a felső ábrán is látható a `tm[1]` egy `double` pointert adna vissza. A második pointer a mátrix egyik sorára mutat, tehát a `tm[1][1]` a második sor második elemére mutat, mivel a számozás 0-tól kezdődik. A mátrix elemeire többféleképpen is hivatkozhatunk a pointerek segítségével, ez a feladat ezt hivatott bemutatni. Ámbár rengeteg lehetőségünk van megnevezni a mátrix egy elemét, szerintem a legjobb döntés a legrövidebb és legkönnyebben megérthető leírási formát alkalmazni, ami a `tm[x][y]`

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó: https://www.youtube.com/watch?v=RribkJ_s-7w.

Megoldás forrása: [bham/thematic_tutorials/bham_textbook_IgyNeveldeProgramozod/Caesar/e.c](https://bham.ac.uk/~bham/teaching/comp101/tutorial/bham_textbook_IgyNeveldeProgramozod/Caesar/e.c).

```
#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];
}
```

Definiálok 2 állandó értéket, a MAX_KULCS és a BUFFER_MERET néven. A MAX_KULCS fogja jelenteni, hogy maximum hány karakterből állhat a kulcs, a BUFFER_MERET pedig, hogy hány karaktert tárolhatunk a bufferben.

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

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

Mivel a program meghívásakor adom meg a használni kívánt kulcsot, ezért használnom kell az argv változót, ami azt fogja mutatni hanyadik szónál tartunk. Az első szó, azaz a 0. elem a program indítása lesz, a második szó maga a kulcs, a harmadik a titkosítani kívánt file neve, míg a negyedik a kimenet neve.

```
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;
    }
}
```

Ameddig tart a szöveg, a kulcs és a bitenkénti eltolás segítségével megváltoztatjuk a szöveget, mondhatni beleolvasztjuk a kulcsot a szövegbe. Ha ezt a műveletet még egyszer használnánk a titkosított szövegen, az eredetit kapnánk vissza.

```
write (1, buffer, olvasott_bajtok);
```

```
}  
}  
}
```

Végül kiírjuk a titkosított szöveget.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó: <https://www.youtube.com/watch?v=6cJVzgdVa5U>.

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

A java EXOR törő is hasonlóan működik a C-s változathoz, lehetséges például egy Java exorral titkosított file-t a C titkosítóban feltörni, természetesen csak akkor, ha ismert a kulcs.

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

Létrehozzuk az ExorTitkosító osztályt, amibe a program elindításakor elmetjük a kulcsot (kulcsSzöveg), azt a szöveget amit titkosítani akarunk (bejövőCsatorna) és létrehozzuk a kimeneti file-t (kimenőCsatorna). Az utolsó sor a hibakezelést végzi.

```
        byte [] kulcs = kulcsSzöveg.getBytes();  
        byte [] buffer = new byte[256];  
        int kulcsIndex = 0;  
        int olvasottBájtok = 0;  
  
        while((olvasottBájtok =  
            bejövőCsatorna.read(buffer)) != -1) {  
  
            for(int i=0; i<olvasottBájtok; ++i) {  
  
                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);  
                kulcsIndex = (kulcsIndex+1) % kulcs.length;  
  
            }  
  
            kimenőCsatorna.write(buffer, 0, olvasottBájtok);  
  
        }  
  
    }  
}
```

Hasonlóan a C EXOR titkosítóhoz, itt is használni fogunk egy tömböt amiben eltároljuk a kulcsot, és egy tömböt, amibe beolvassuk a bejövő szöveget, ez lesz a buffer. Egy while ciklusban végezzük ez a logikai vagy műveletet, addig, amíg van szöveg a bufferben. Ha befejeztük a szöveg titkosítását, a kimenetet kiírjuk a megfelelő helyre.

```
public static void main(String[] args) {  
  
    try {  
  
        new ExorTitkosító(args[0], System.in, System.out);  
  
    } catch (java.io.IOException e) {  
  
        e.printStackTrace();  
  
    }  
  
}
```

Már csak a főfüggvény megírása maradt. A main megpróbálja elindítani a programot a try utasítással. A próba során meghívja a titkosítót. Ha valami hiba van, a catch fogja elkapni az üzenetet.

4.4. C EXOR törő

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

Megoldás videó: <https://www.youtube.com/watch?v=sRDQ0CX1hko> .

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/t.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/IgyNeveldaProgramozod/Caesar/t.c)

Az egyszerűség kedvéért, és hogy a program biztosan lefusson bármilyen gépen, anélkül hogy túlságosan megterhelje a rendszert egy kicsit egyszerűsíték a problémán. A kódomban maximum 4 betűs kulcsokat keresek, úgy, hogy magukat a betűket ismerem a kulcsból, csak a sorrendet nem. Ezzel megkimélem a gyengébb gépeket, de emiatt a program mechanizmusa nem változik.

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256  
#define KULCS_MERET 4  
#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
tisza_lehet (const char *titkos, int titkos_meret)
{
    double szohossz = atlagos_szohossz (titkos, titkos_meret);

    return szohossz > 6.0 && szohossz < 9.0
        && strcasestr (titkos, "hogya") && strcasestr (titkos, "nem")
        && strcasestr (titkos, "az") && strcasestr (titkos, "ha");
}
}
```

Ismét szükségem lesz egy alpból definiált számra, ami meghatározza a kulcs méretét és a buffert. Két alprogramot fogok használni, hogy megállapítsam, a kigenerált kulcs jó e. Ezt úgy teszem meg, hogy meg-nézem szerepel e a szavak között a "hogya", "nem", "az" és a "ha" szó. Ezek a szavak szoktak a legtöbbször szerepelni a mondatokban, így ha a szövegünk tartalmazza ezeket a szavakat, majdnem biztosak lehetünk benne, hogy jó kulcsot találtunk.

```
void
xor (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
xor_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);
}
}
```

Feltörés esetén is a bitenkénti vagyot, ezzel kapjuk meg a titkosított szöveg formáját, ha használjuk rajta a kulcsot.

```
int
main (void)
{

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

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

    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\\0';
    char str[4] = {'t', 'i', 'k', 'a'};

    for (int ii = 0; ii <= 4; ++ii)
    for (int li = 0; li <= 4; ++li)
        for (int ki = 0; ki <= 4; ++ki)
            for (int ji = 0; ji <= 4; ++ji)
            {
                kulcs[0] = str[ii];
                kulcs[1] = str[li];
                kulcs[2] = str[ki];
                kulcs[3] = str[ji];

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

```
        exor (kulcs, KULCS_MERET, titkos, p - ←  
            titkos);  
    }  
  
    return 0;  
}
```

Mostmár csak meg kell hívni az elkészített alprogramokat a főfüggvényben. Itt adom meg azt is, hogy melyik az a 4 karakter aminek szeretném ha kipróbálná az összes variációját, jelen esetben ezek a t,i,k,a . Négy ciklust használok, ezek segítségével írom át a szöveget. Ahogy megkapom a fordítást és tiszta is, megvan a végeredményem, a program feltörte a kódot. Ha nem négy megadott betűre szeretném lefuttatni, annyi dolgom van, hogy az összes lehetséges betű kombinációját kell vennem, ez megoldható ciklusok használatával is. A kód feltörése sokkal bonyolultabb és hosszabb folyamat, mint a titkosítás, de ha az ember egy jó géppel rendelkezik, akkor egy 16 betűs kulcs feltörése sem jelent nagy kihívást, pontosan ezért nem használják ezt a fajta titkosítást a modern időkben.

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

Neurális hálók alatt két dolgot érthetünk: biológiai értelemben véve, a neurális hálózatot neuronok alkotják, amik összeköttetésben vannak egymással - ezek a hálók megtalálhatóak az idegrendszerekben - informatikai értelemben véve pedig mesterséges neurális hálózatra gondolunk, ez lehet akár gép vagy program, ami egy biológiai neurális hálózatot mintáz. A mesterséges neurális hálókat általában a technikában használják.

A programban mi egy mesterséges neurális hálózattal foglalkozunk, neki próbálunk "megtanítani" különböző műveleteket, név szerint az OR-t, az AND-et és az EXOR-t. Azért, hogy ezeket a műveleteket használni is tudja, adatokra is szükség van, ezt a programban ax-el jelöljük, ahol x egy természetes szám. A plot függvénnyel ki is tudjuk rajzolni a hálót, és az is megfigyelhető, hogy egészen jól tudja használni az OR-t, az értékek megközelítőleg pontosak.

Az AND is, hasonlóan az OR-hoz, egészen jó megoldásokat ad vissza, így az ember azt gondolná, hogy az EXOR-al sem lehet semmi baj. Ez egy hatalmas tévedés, ugyanis itt a hibahatár már nem egy rendkívül kis szám lesz, a program egész nagyokat tud tévedni. Rengeteg gondolkodás után, végül rájöttek a programozók, hogyan is hidalható át ez a probléma, rejtett- vagy hidden neuronokkal. Ezt a megoldást használva, a hálónak már az EXOR sem jelent igazi nehézséget, az eredmények ismét nagyon pontosak lesznek.

4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

Perceptronokról informatikában leginkább a mesterséges intelligencia, azon belül a neurális hálók témakörében találkozhatunk. A perceptronnak fontos szerepe van, leellenőrzi a bemenetet, majd egy feltétel alapján dönt, mi legyen a kimenet.

Tehát a perceptronok hasznosak, segítségükkel létrehozhatunk egy hibahatárt, amit a program be fog tartani. Feltételezzük, hogy 100 emberből legalább 80-nak magánhangzóval kezdődik a neve. Egyértelmű, hogy azt várjuk el, hogy legyen 80 olyan bemeneti adatunk, ahol a személy neve magánhangzóval kezdődik. Azonban ha a Perceptron megtalálja a 21. személyt akinek mássalhangzóval kezdődik a neve visszadob egy olyan értéket, ami jelzi nekünk, hogy a feltétel nem teljesült, például a (-1)-et. Ha ez történt meg, akkor a hibahatárt át kell állítanunk, finomhangolást kell végeznünk rajta.

Általában a finomhangolást nagyon magas határnál kezdi, majd addig csökkentik, amig elfogadható a hibák mennyisége.

5. fejezet

Helló, Mandelbrot!

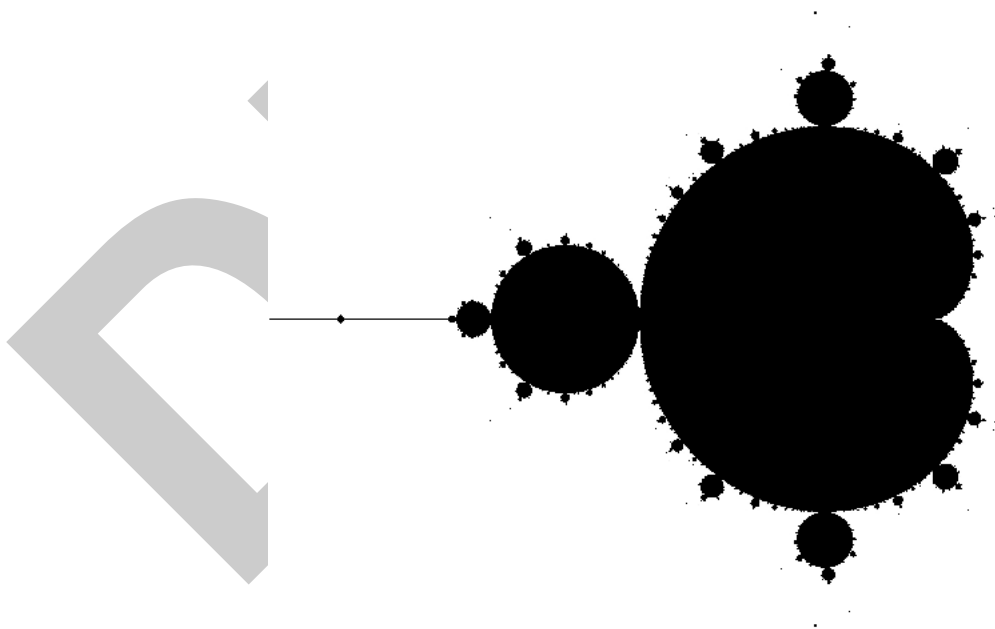
5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás videó: <https://youtu.be/gvaqijHIRUs>

Futtatás: <https://www.youtube.com/watch?v=dJKAj5ebrGg>

Megoldás forrása: https://github.com/Amsekal/Bhax/blob/master/bhax-master/attention_raising/CUDA/mandelp



5.1. ábra. A Mandelbrot halmaz a komplex síkon

A Mandelbrot halmaz egy olyan komplex számok halmaza, amelyet ha kivetítünk a síkra egy látványos és egyben érdekes alakzatot kapunk. Ezt a halmazt olyan számok alkotják, amelyeknek van valós és imaginárius részük is, így lehetséges az is, hogy gyök alatt negatív számunk legyen.

A kód segítségével megállapítjuk, hogy egy pont mikor eleme a halmaznak. Ha eleme, a pontot feketére színezzük.

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

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

Megoldás videó: <https://youtu.be/gvaqijHIRUs>

Forrás: https://github.com/Amsekal/Bhax/blob/master/bhax-master/attention_raising/Mandelbrot/3.1.2.cpp

Hasonlóan az előző feladathoz, itt is az a célunk, hogy a Mandelbrot halmazt kirajzoljuk, tehát a két végeredménynek azonosnak kell lennie. A különbség csak annyi, hogy ebben a programban használni fogunk egy osztályt, ami segít a komplex számok kezelésében.

```
// 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 ↵
-0.01947381057309366392260585598705802112818 ↵
-0.0194738105725413418456426484226540196687 ↵
0.7985057569338268601555341774655971676111 ↵
0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ↵
0.4127655418209589255340574709407519549131 ↵
0.4127655418245818053080142817634623497725 ↵
0.2135387051768746491386963270997512154281 ↵
0.2135387051804975289126531379224616102874
// Nyomtatás:
// 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 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/>.
```

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

Látható, hogy továbbra is szükség van a png++ -ra, ennek köszönhetően vagyunk képesek kirajzolni a halmazt a sikra ebben és az előző programban is.

```
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";

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 különbség tehát a complex osztály használatában van. Ez egyszerűbbé is teszi a kódot, ugyanis nem kell azzal bajlódni a programozónak, hogy külön változót használjon egész és imaginárius résznek. Az osztály segítségével elég egy változót használni, és egy utána irt zárójelbe megadni a két részt.

Fontosnak tartom még megjegyezni, hogy a programot ugyanúgy kell futtatni, mint az előző esetben.

5.3. Biomorfok

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

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

A Mandelbrot és Júlia halmazok igen hasonlóak, a legszembevetőbb különbség az, hogy míg a c változó valós és imaginárius része változott a program továbbhaladtával a Mandelbrot halmaz esetén, addig a Júlia halmaznál ezeket az értékeket rögzítjük. Ezt a kódokban a két for ciklusban figyelhetjük meg.

Most következi egy igazán fontos kérdés: Ez nekünk miért jó? Erre a válasz egészen egyszerű: amíg a Mandelbrot halmaz ugyanúgy néz ki, addig Júlia halmazból jóval több van, és ezzel a megoldással ki is tudjuk őket rajzolni. Nincs más dolgunk csak a C változó értékét módosítsuk. Azonban nem értő kezek között nagy annak a valószínűsége, hogy a kapott halmaz nem lesz látványos, szerencsére az internet bőven szolgál számértékekkel, amivel csodás látványt érhetünk el.

Ha célunk a biomorfok pontos megértése akkor a következő cikk a segítségünkre lehet. https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf. A kód megírása nehézséget is jelenthet, szóval nyugodtan használjuk fel az előző feladatokban írt kódot, és a fentebb említetteket írjuk át.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó: <https://www.youtube.com/watch?v=ihZBrMlpVbM>

Megoldás forrása: https://github.com/Amsekal/Bhax/blob/master/bhax-master/attention_raising/CUDA/mandelpngc_60x60_100.cu

Ez a feladat egy kicsivel különlegesebb, mint az előzőek, ugyanis CUDA magokat használó kártyára van szükségünk a futtatásához. Konceptióban a program megegyezik az előző Mandelbrot feladatokkal, a különbség abban feezhető fel, hogy itt a CUDA magok segítségével futtatjuk le a programot.

Mint észrevehető a videókban, a különbség nem triviális, sokkal gyorsabban képes elkészíteni a képet a program, ha CUDA magokat használ. Minél jobb a kártya, annál gyorsabb képgenerálás figyelhető meg.

Mivel a programot a videokártyának írtuk, ezért megfigyelhető, hogy a megszokott gcc vagy g++ helyett most nvcc-t használunk. Hogy használni tudjuk ezt a parancsot szükségünk lesz az nvidia-cuda-toolkitre.

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 iteráció bejárta z_n komplex számokat!

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

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/binom/Batfai-Barki/frak/>

Visszatérünk a Mandelbrot halmazokhoz, de most nem csak egy képet szeretnénk végeredményként kapni, hanem egy olyan ablakot, ahol képesek vagyunk ránagyítani a halmazra. Ez lehetséges png file esetén is, de ott egy idő után a kép pixeles lesz, és nem lehet kivenni belőle semmit.

A megoldás amit használni fogunk ennél egy kicsit bonyolultabb, emiatt talán ez a fejezet legösszetettebb feladata. Az ötlet az, hogy az egérrel kijelölt részre az ablak nem ránagyít, hanem újragenerálja azt a részt. Így kiküszöbölhető az a probléma, hogy nagyon pixelesse válik a kép, ugyanis a pixelszám azonos lesz.

Az első probléma még a kódolás előtt megjelenik, hogyan készítsük el ezt az ablakot. Egy nagyszerű megoldás a Qt eszköztár használata, amelyben elkészíthető egy gui(grafikus interfész). Ennek az interfésznek a futtatásához 5 kódrészre van szükségünk, a következőekben ezek szerepeit mutatom be:

frakablak.h

Ebben a kódrészben egy osztályt hozunk létre, aminek van publikus, védett és privát tagja is. A publikus részben a tartományokat/határokat szabjuk meg, amik között mozoghatunk. A védett részben egy függvényt deklarálunk, amely figyelemmel követi, mit csinál a felhasználó. Ennek fontos szerepe lesz, hogy a program tudja, mikor kell nagyítania. Végül a privát tagban a nagyítandó terület van meghatározva.

frakszal.h

Ismét egy osztályt láthatunk és még pár változót. Ezek segítenek majd a programnak a számolásban és rajzolásban

frakszal.cpp

Ebben a kódrészben történik a legfontosabb dolog: a Mandelbrot halmaz elemeinek kiszámolása, ehhez hasonló kódot írtunk az első feladatokban is.

main.cpp

Itt kerül meghívásra qt konstruktora, és itt tesszük a kódhoz a Qt könyvtárat.

frak.pro

A frak.pro lesz az úgynevezett király, alatta fognak futni az előbb bemutatott kódok, az ő feladata lesz a teljes folyamat kezelése.

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>

A feladat itt is ugyan az, mint amit az előbb láthattunk, annyi különbséggel, hogy itt JAVA nyelvben kell megírunk a programot. A kód szerkezete rendkívül hasonló az előző feladatban megírtakéhoz, itt is az egérrel szeretnénk kiválasztani egy részt a nagy egészéből, amire ráközelítenénk/ahová utaznánk.

Azonban ez a program többre is képes, mint amit az előbb létrehoztunk. Nem csak kirajzolja azt amire ránagyítunk, de képes pillanatképet is készíteni arról, ezt a kódrészt a public void pillanatfelvétel() alatt találhatjuk meg. A programrész nem olyan bonyolult, lényegében kiment a jelenleg aktív ablakot, és elnevezi. Ezt az elnevezést segíti egy számláló is, így ha több képet akarunk csinálni, az egyik nem fof rámentődni a másikra a néveggyezés miatt.

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 C++:<https://www.youtube.com/watch?v=-0uMOANtCPA>

Megoldás videó Java:<https://www.youtube.com/watch?v=TGpmeDn7KBY>

Megoldás forrása: https://github.com/Amsekal/Bhax/tree/master/bhax_textbook_IgyNeveldaProgramozod/Welch
Illetve <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsojava/PolarGen.java#l10>

A kód maga igen egyszerű. Először is, szükség lesz egy osztályra:

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

private:
    bool nincsTarolt;
    double tarolt;

};
```


Az osztályt PolarGennek nevezzük el. Ebben az osztályban 2 privát tag lesz, a boolean típusu változó, a nincsTarolt, ami azt fogja jelezni számunkra, hogy van e eltárolt érték, és egy double típusu tarolt nevű változó. A konstruktorban megváltoztatjuk a nincsTarolt értékét igazra, majd létrehozunk egy random számot a srand() függvény segítségével. A destruktort üresen hadjuk. Ezen kívül a kovetkezo alprogram fejlécét találjuk az osztály definíciójában.

```
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 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);

        double r = std::sqrt ((-2 * std::log (w)) / w);

        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

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

Hogy a program működjön is, szükség van a kovetkezo alprogramra. Itt, ha a nincsTarolt értéke igaz, akkor egy matematikai képletet fog alkalmazni, hogy végbemenjen a polártranszformáció. A tarolt kap egy értéket a képlet alapján, majd a boolean változót hamisra állítjuk. Ha a nincsTarolt értéke hamis, igazra állítjuk, és visszaadjuk a tarolt értékét.

```
int
main (int argc, char **argv)
{
    PolarGen pg;

    for (int i = 0; i < 10; ++i)
        std::cout << pg.kovetkezo () << std::endl;
```

```
return 0;  
}  
}
```

Ezek után már csak egy dolgunk maradt, megírni a főfüggvényt. Deklarálunk egy változót ami a PolarGen osztályhoz tartozik, a pg-t, majd egy for ciklusban többször is meghívjuk a következő függvényt, ennek következtében több eredményt is kapunk. A for ciklus után véget ér a program.

A java kód is pontosan ugyanezen alapok mellett íródott, egyetlen különbség, hogy ott csak egyszer lesz meghívva az alprogram, így csak egy eredményt kapunk.

Érdekesség, hogy létezik a Sun programozói által készített Random.java file, ami nagyban hasonlít az általunk elkészített megoldáshoz.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: 6.6-os feladatban

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: 6.6-os feladatban

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: 6.6-os feladatban

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: 6.6-os feladatban

6.6. Mozgató szemantika

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

Megoldás videó: https://www.youtube.com/watch?v=a_K2EWM53tg

Megoldás forrása: https://gitlab.com/nbatfai/bhax/-/blob/master/distance_learning/ziv_lempel_welch/z3a18qa_fr

Ahogy látható, az előző feladatokban el kellett készítenünk egy binfát, amit folyamatosan fejlesztünk. Az egyszerűség kedvéért, én az összes alpontot ebben a feladatban fogom bemutatni.

A Bináris fa egy különleges szerkezet, mivel minden gyökér maximum 2 utóddal rendelkezhet, egy jobb-oldalival és egy baloldalival. Előfordulhat olyan eset is, amikor csak az egyik oldalon rendelkezik utóddal, vagy nem rendelkezik egyel sem. A mi binfánk esetében a csomópontokban tárol értékek egyesek és nullák lesznek.

Amikor megkapjuk azt az értéket, legyen most ez 1, amit be szeretnénk tenni a fába, a következő folyamat fog lezajlani: megnézzük, hogy az aktuális csomópontnak/ helyi gyökérnek (legelső gyökér egy '/'-t tárol) van-e 1-es utódja. Ha van, a dolgunk egyszerű, az 1-es lesz a következő helyi gyökér. Ha nincs, akkor létrehozunk egy új csomópontot, és az ő utódja lesz az 1. Azt, hogy melyik elemnél tartunk egy pointer mutatja, mikor új csomópontot hozunk létre, ez a pointer visszatér az eredeti gyökérre, a /-re. Ugyanez az algoritmus akkor is, ha az érték 0.

A feladat teljes megítéséhez pár fogalom tisztázásra szorul:

Rekurzió: Rekurzió a programozásban nem jelent mást, minthogy egy alprogram úgy végzi el a feladatot, hogy segítségül saját magát hívja meg. Fontos megjegyezni, hogy a rekurzió egy úgynevezett verem memóriát használ, ami képes megtelni, esetenként túlszordulni. Rekurzívan lehet megoldani különböző kereséseket, de akár a backtracking megírásában is segíthet.

Konstruktor: Egy osztály metódusa, úgynevezett tagfüggvény, amely akkor hívódik meg, ha létrejön az objektum. Fontos, hogy a neve azonos kell legyen az osztályéval, nem hívható meg közvetlenül, és nem tartalmazhat visszatérési értéket.

Destruktor: A konstruktorhoz hasonlóan ez is egy tagfüggvény, akkor hívódik meg, ha az objektum megszűnik. A neve azonos az osztályéval, de előtte egy ~-nek kell szerepelnie. Használati szabályai azonosak a konstruktorral.

Hogy ezt meg tudjuk valósítani, szükségün lesz egy BinFa osztályra, amit jelen esetben BinTree-nek fogunk hívni. Ebben az osztályban használni fogunk még egy osztályt, ami a csomópontok használatában fog segíteni, ezt Node-nak nevezzük el.

```
#include <iostream>

template <typename ValueType>
class BinTree {

protected:
    class Node {

    private:
        ValueType value;
        Node *left;
```

```
Node *right;
int count{0};

// TODO rule of five
Node(const Node &);
Node & operator=(const Node &);
Node(Node &&);
Node & operator=(Node &&);

public:
    Node(ValueType value): value(value), left(nullptr), right(nullptr) ↔
    {}
    ValueType getValue(){return value;}
    Node * leftChild(){return left;}
    Node * rightChild(){return right;}
    void leftChild(Node * node){left = node;}
    void rightChild(Node * node){right = node;}
    int getCount(){return count;}
    void incCount(){++count;}
};

}
```

A Node osztályban a value értékben lesz eltárolva az érték amivel dolgozni szeretnénk, utána pedig 2 mutatót is deklarálunk, ami a két lehetséges utódra mutat, végül pedig egy integer változót, amivel számolni fogunk. A következő sorokban tiltjuk a másoló/mozgató konstruktor és másoló értékadást.

Publicban található meg a konstruktor, ahol értéket adunk az előbb bevezetett változóknak, a value változó megkapja az értéket, a pointerek, mivel kezdetben nincs utódjuk, nem mutatnak semmire. Ezek után létrehozunk alprogramokat, az első 3 egyszerű érték visszaadás, az utánnuk következő kettővel a pointert tudjuk majd állítani, a getCount visszaadja a count értékét, az incCount pedig a count értékét növeli. Most, hogy készen vagyunk a Node osztállyal, következhet a BinTree osztály befejezése.

```
Node *root;
Node *treep;
int depth{0};

private:
    // TODO rule of five

public:
    BinTree(Node *root = nullptr, Node *treep = nullptr): root(root), treep ↔
    (treep) {
        std::cout << "BT konstruktor" << std::endl;
    }

    BinTree(const BinTree & old) {
        std::cout << "BT masolo konstruktor" << std::endl;

        root = cp(old.root, old.treep);
    }
}
```

```
}

Node * cp(Node *node, Node *treep)
{
    Node * newNode = nullptr;

    if(node)
    {
        newNode = new Node(node->getValue());

        newNode->leftChild(cp(node->leftChild(), treep));
        newNode->rightChild(cp(node->rightChild(), treep));

        if(node == treep)
            this->treep = newNode;
    }

    return newNode;
}

BinTree & operator=(const BinTree & old) {
    std::cout << "BT masolo ertekadas" << std::endl;

    BinTree tmp{old};
    std::swap(*this, tmp);
    return *this;
}

BinTree(BinTree && old) {
    std::cout << "BT mozgato konstruktor" << std::endl;

    root = nullptr;
    *this = std::move(old);
}

BinTree & operator=(BinTree && old) {
    std::cout << "BT mozgato ertekadas" << std::endl;

    std::swap(old.root, root);
    std::swap(old.treep, treep);

    return *this;
}

~BinTree() {
    std::cout << "BT destruktork" << std::endl;
    deltree(root);
}
```

```
    BinTree & operator<<(ValueType value);
    void print(){print(root, std::cout);}
    void print(Node *node, std::ostream & os);
    void deltree(Node *node);

};

template <typename ValueType, ValueType vr, ValueType v0>
class ZLWTree : public BinTree<ValueType> {

public:
    ZLWTree(): BinTree<ValueType>(new typename BinTree<ValueType>::Node(vr) ←
        ) {
        this->treep = this->root;
    }
    ZLWTree & operator<<(ValueType value);

};
}
```

Kezdeként készítünk még 2 pointert, egyet a gyökérnek, egyet a fának, és egy integer változóra is szükség lesz, hogy a mélységet is nyomon tudjuk követni. Itt is szükség lesz egy konstruktorra, és hogy nyomon tudjuk követni a dolgokat, egy kiiratás is szerepet kap. Létrehozunk még egy másoló- és mozgó értékek mozgatás konstruktor is. A mozgatást egy swap függvénnyel oldjuk meg, itt meg kell adni a két csomópontot amit cserélnénk. Fontos észrevenni, hogy nem csak a csomópontok cserélődnek, hanem a mutatók is.

A másolás egy kicsit bonyolultabb, itt használjuk majd a `Node * cp-t` is. Ezzel a kóddal rekurzió segítségével tudjuk létrehozni a fa másolatát. Végezetül a shiftelést definiáljuk, ennek következtében tudunk a main függvényben értéket adni majd a programnak, ami alapján felépül a fa. Azonban ezt egy template-ben csináljuk meg.

```
template <typename ValueType>
BinTree<ValueType> & BinTree<ValueType>::operator<<(ValueType value)
{
    if(!treep) {

        root = treep = new Node(value);

    } else if (treep->getValue() == value) {

        treep->incCount();

    } else if (treep->getValue() > value) {

        if(!treep->leftChild()) {

            treep->leftChild(new Node(value));

        }

    }

}
```

```
        } else {

            treep = treep->leftChild();
            *this << value;
        }

    } else if (treep->getValue() < value) {

        if(!treep->rightChild()) {

            treep->rightChild(new Node(value));

        } else {

            treep = treep->rightChild();
            *this << value;
        }

    }

    treep = root;

    return *this;
}
}
```

A fenti kódban látható a fa felépítése, az alapján amit a fentiekben leírtam. A programban található még egy template ami a fa mélységét számolja ki és kiírja (bejárja) a fát, de mivel ez csak számolás és kiírás, erre nem térnék ki külön.

```
template <typename ValueType>
void BinTree<ValueType>::deltree(Node *node)
{
    if (node)
    {
        deltree(node->leftChild());
        deltree(node->rightChild());

        delete node;
    }
}
}
```

Ez a template felelős a fa törléséért, először az utódokat törli, majd végül a helyi gyökeret.

```
int main(int argc, char** argv, char ** env)
{
    bts << "alma" << "korte" << "banan" << "korte";

    bts.print();
}
```

```
}  
}
```

A main függvényben már semmi dolgunk nincs, csak felhasználni az eddig készített kódokat. A bts-be shiftelünk szavakat, de mivel a shifteléshez irtunk egy programot, így a szavakból fát fog készíteni, amit a print utasítással irtunk ki.

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...

DRAFT

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

[?]

A programozási nyelveket 3 részre oszthatjuk: assembly, gépi és magas szintű nyelv, amivel forrásszöveget írhatunk. A forrásszövegből a gép által is értelmezhető kódot fordítóprogramok segítségével érhetünk el. A fordítóprogram lépései: lexikális, szintaktikai, szemantikai elemzés, kódgenerálás. Minden programnyelv rendelkezik jól meghatározott szabvánnyal melyben megtalálhatóak a nyelv szabályai. Az implementációk inkompatibilitását a hordozhatóság problémájának nevezzük, ez még a mai napig vár egy teljes megoldásra. A programnyelvek lehetnek: imperatív, deklaratív, egyéb nyelvek. Imperatív nyelvek: algoritmusokat használnak. Deklaratív nyelvek: nem használnak algoritmusokat, pl logikai nyelv. Más nyelvek: nincs jellemzőjük.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Alapismeretek

A könyv első fejezetében a C nyelv alapjaival ismerkedhetünk meg. Egyszerű feladatok segítségével ismerjük meg a nyelv szematikáját, szabványát, a változók használatát és bemutatásra kerülnek az ismétlődő ciklusok is. Ezen felül foglalkozunk még alapvető függvényekkel és pár napi szinten használt eljárással.

Tipusok, operátorok, kifejezések

Ahogy a fejezet címéből is kiderül, a változók tipusaival, elnevezési szabályaival és felhasználásával találkozunk. Megtudjuk a pi és az Euler féle szám értékét és megtanulunk mi is ezekhez hasonló állandót létrehozni. Találkozunk a logikai és matematikai operátorokkal, és ezek felhasználásával, de a bitenkénti logikai operátorok sem maradnak ki.

10.3. Programozás

[BMECPP]

Az első fejezet betekintést enged nyerni a c++ történelmébe.

A második fejezet már több információval szolgál programozás terén, bemutatja a c++ nyelvet és az alapvető különbségeket közte és a C között. Megimserkedünk a main függvénnel, és olyan dolgokkal, amik C-ben nincsenek: bool adattípus, stb. Kitér az új nyelvi szabályokra is, ami c után egy kicsit furcsa lehet, például struct helyett class-t használunk, de rengeteg módosítás történt.

10.4. Programozás, Python

Python nyelv bemutatása

A Python egy, a többi nyelvhez képest gyors fejlesztési sebességgel rendelkező nyelv, rengeteg támogatott eszközzel rendelkezik, ezen felül egy általános célú programozási nyelv

Python

Pythonban nem található meg a jól ismert `begi`, `end`, de még a pontos vessző sem. Nem szükséges változókat deklarálni, azok bármikor használhatóak, így elmondható, hogy az egyik leg kezdőbarátab prog. nyelv. A nyelv rendkívül különleges szemantikával és szabályrendszerrel rendelkezik, főleg ha az ember C vagy C++-ról tér át.

III. rész

Második felvonás

DRAFT

**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.

DRAFT

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. & Ritchie, Dennis M., *A C programozási nyelv*, Bp., Műszaki, 1993.

11.5. C++

[BMECPP] Benedek, Zoltán & 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.