

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 Szabó, Dániel József	2020. március 8.	

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	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	12
2.6. Helló, Google!	14
2.7. A Monty Hall probléma	15
2.8. 100 éves a Brun tétel	17
2.9. Minecraft MALMÖ (csiga)	21
3. Helló, Chomsky!	22
3.1. Decimálisból unárisba átváltó Turing gép	22
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	23
3.3. Hivatkozási nyelv	25
3.4. Saját lexikális elemző	27
3.5. Leetspeak	28

3.6. A források olvasása	30
3.7. Logikus	31
3.8. Deklaráció	32
3.9. 9. Minecraft MALMÖ (csiga diszkrét)	35
4. Helló, Caesar!	36
4.1. double ** háromszögmátrix	36
4.2. C EXOR titkosító	38
4.3. Java EXOR titkosító	38
4.4. C EXOR törő	39
4.5. Neurális OR, AND és EXOR kapu	39
4.6. Hiba-visszaterjesztéses perceptron	39
4.7. Minecraft MALMÖ (érzékelés)	39
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	43
5.4. A Mandelbrot halmaz CUDA megvalósítása	47
5.5. Mandelbrot nagyító és utazó C++ nyelven	47
5.6. Mandelbrot nagyító és utazó Java nyelven	48
6. Helló, Welch!	49
6.1. Első osztályom	49
6.2. LZW	49
6.3. Fabejárás	49
6.4. Tag a gyökér	49
6.5. Mutató a gyökér	50
6.6. Mozgató szemantika	50
7. Helló, Conway!	51
7.1. Hangyaszimulációk	51
7.2. Java életjáték	51
7.3. Qt C++ életjáték	51
7.4. BrainB Benchmark	52

8. Helló, Schwarzenegger!	53
8.1. Szoftmax Py MNIST	53
8.2. Mély MNIST	53
8.3. Minecraft-MALMÖ	53
9. Helló, Chaitin!	54
9.1. Iteratív és rekurzív faktoriális Lisp-ben	54
9.2. Gimp Scheme Script-fu: króm effekt	54
9.3. Gimp Scheme Script-fu: név mandala	54
10. Helló, Gutenberg!	55
10.1. Programozási alapfogalmak	55
10.2. Programozás bevezetés	55
10.3. Programozás	55
III. Második felvonás	56
11. Helló, Arroway!	58
11.1. A BPP algoritmus Java megvalósítása	58
11.2. Java osztályok a Pi-ben	58
IV. Irodalomjegyzék	59
11.3. Általános	60
11.4. C	60
11.5. C++	60
11.6. Lisp	60

Ábrák jegyzéke

2.1. A B_2 konstans közelítése	20
4.1. A <code>double **</code> háromszögmátrix a memóriában	38
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 (hogy lássuk mást is) példával.

Hogyan nyomjuk?

Ránts 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.xsl
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 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.

- Social Network - A közösségi háló, [link](#), benne a Facebook megalkotásának bemutatása.
- Mr. Robot (sorozat), [link](#), benne a Hackelés bemutatása.
- Westworld 1. évad (sorozat), [link](#), benne az AI bemutatása.
- Ex Machina, [link](#), benne megint csak az AI bemutatása.
- A nő, [link](#), benne az AI újszerű bemutatása.
- Szilícium-völgy (sorozat), [link](#), benne egy tipikus IT startup cég bemutatása.
- The Great Hack, [link](#), benne a Big Data, valamint az összegyűjtött információkkal való visszaélés 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.

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/enltfeOhGCA>

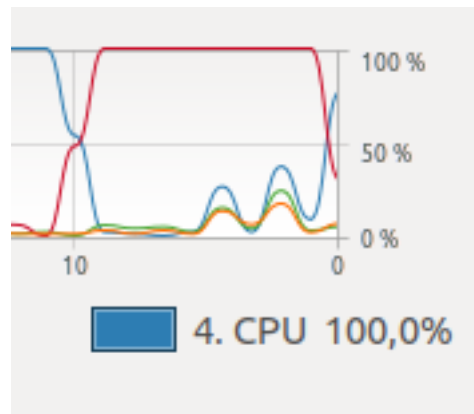
A ciklusok, vagy más néven iterációk a programozók egyik leghasznosabb eszközei. Alapjáraton a végtelen ciklus egy HIBA, kivéve, ha a programozó direkt végtelen ciklust szeretne létrehozni. Ezt számos módon megteheti, valamint számos oka lehet erre. Például egy szerveren bizonyos folyamatnak/szolgáltatásnak állandóan futnia kell, ennek a kivitelezésére pedig a végtelen ciklus egy tökéletes megoldás. Akkor is nagy hasznát veszi a programozó a végtelen ciklusoknak, ha a BHAX könyv ilyenek írására kéri...

Egy mag 100 százalékban:

Megoldás forrása: [vegtelen1.c](#)

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

Ebben a formában a kód minden C szabvánnyal lefordul. Sokat nem lehet róla mondani. Mivel a ciklusfejben nincs semmilyen feltétel megadva, így addig fog futni, amíg manuálisan le nem állítjuk.



1 mag 100%

Egy mag 0 százalékban:

Megoldás forrása: [vegtelen2.c](#)

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

Ehhez a feladathoz az első programkódot csak egy kicsit kellett megváltoztatnunk. Először is include-olni kell az unistd.h nevű header fájlt, hogy használhassuk a sleep() eljárást. Miután meghívjuk, lassítja az iterálás sebességét, így a processzor sose lesz 100%-os kihasználtságon.

```
dani@Ubi:~$ ps aux | egrep inf
dani    24047  0.0  0.0  23088  1004 pts/1    S+   22:52
```

1 mag 0%

Minden mag 100 százalékban:

Megoldás forrása: [vegtelen3.c](#)

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

Ehhez a feladathoz a végtelen ciklusunkat több szálon, párhuzamosan kell futtatnunk. Erre megoldást nyújthat az OpenMP, melynek használatához include-oljuk az omp.h headert. Dióhéjban úgy működik, hogy azt az utasítást, amit a #pragma omp parallel-el megjelölünk, rákényszeríti, hogy minden szálon,

egymástól függetlenül hajtódjon végre. Mivel ez a ciklus sose ér véget, így elérjük, hogy a programunk 100%-on kihasználja a CPU-t.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
7105	dani	20	0	4372	740	676	R	100,0	0,0	1:17.50	a

Minden mag 100%



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!

Tegyük fel, 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 az előző végtelen ciklust tartalmazó programunkra:

```
T100 (vegtelen1.c)
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 fog 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.

Habár a fenti indirekt bizonyításból is egyértelműen látszik, előttünk már Turing is bebizonyította azt, hogy nem létezik ilyen algoritmus. Ennek megállási probléma a neve. Lényege, hogy ha egy végtelenségig futó programot kap meg tesztelésre ez a nem létező algoritmus, akkor az ugyanúgy a végtelenségig fogja tesztelni a bemenetet, így ő maga is le fog fagyni előbb utóbb. Így könnyen beláthatjuk, hogy nem létezik ilyen program.

2.3. Változók értékének felcserélése

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

Megoldás videó: <https://youtu.be/Sd71L1OWhdY>

Megoldás forrása: [valtozo.c](#)

```
#include<stdio.h>
int main()
{
    int a = 4;
    int b = 6;

    a = b - a;
    b -= a;
    a = b + a;

    printf("%d, %d", a, b);
}
```

Egy rendkívül egyszerű, mégis praktikus matematikai megoldást alkalmazva felcserélhetjük két változó értékét segédváltozó bevezetése nélkül. A fenti példában az "a" változó értéke 4, a "b" változóé pedig 6. Első lépésnek az "a" értékét megváltoztatjuk ($b-a = 2$)-re. Ezután "b"-ből kivonjuk az "a"-t, így 4 lesz az értéke. Végül a "b"-hez hozzáadjuk az "a" értékét ($4+2 = 6$), majd ezt hozzárendeljük az "a"-hoz. És voilá, megcseréltük a két változó értékét.

Természetesen az ilyen egyszerű és triviális feladatok elvégzését a magas szintű programozási nyelvek elintézik a programozó helyett. Például C++-ban a `swap(a, b)` függvény segítségével egyszerűen cserélhetjük meg két változó értékét, igaz, ez a háttérben egy csereváltozó segítségével hajtodik végre.

2.4. Labdapattogás

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

Labdapattogás if-ekkel:

Megoldás videó: <https://youtu.be/6-C7KNiR2Ig>

Megoldás forrása: [labdaif.c](#)

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

int main()
{
    int x, y, x_meret, y_meret;
    x = y = 0;
    int sor = 1;
    int oszlop = 1;

    WINDOW *terminal;
```

```
terminal = initscr();

for(;;)
{
    getmaxyx(terminal, y_meret, x_meret);
    mvprintw(y, x, "HEYO!");
    refresh();
    usleep(100000);
    clear();
    x += sor;
    y += oszlop;

    if (x <= 0)                //bal oldal
        sor *= -1;
    if (x >= x_meret-1)        //jobb oldal
        sor *= -1;
    if (y >= y_meret-1)        //alja
        oszlop *= -1;
    if (y <= 0)                //teteje
        oszlop *= -1;
}
}
```

A feladat elvégzéséhez igénybe kell vennünk a `curses.h` headert, valamint az `stdio.h/unistd.h`-t is. Az ncurses könyvtárral Console Application-ök írásakor tudjuk kihasználni a terminál adta lehetőségeket, különféle színezést, grafikai elemek megjelenítését teszi lehetővé. If elágazásokkal nem nehéz feladat, hiszen csak meg kell vizsgálnunk, hogy a labda mikor ütközik "falhoz" (a terminál ablak széléhez), majd onnan milyen irányba fog visszapattanni. Mivel 4 oldala van, így négy elágazást írunk, ezeket egy végtelen ciklusba rakjuk. A ciklus előtt deklaráljuk a szükséges változókat, az x/y koordinátákat, melyeknek a kezdőértéke 0, az x_meret/y_meret a getmaxyx() függvényhez szükséges (a curses.h tartalmazza), mely a kurzor helyét határozza meg a terminálon, valamint kell egy sor/oszlop változó, melyekkel szabályozhatjuk, hogy milyen lépésközlőként jelenjen meg a labda a terminálablakban. Az alábbi

```
WINDOW *terminal;
terminal = initscr();
```

kódrészlet a terminál ablak méretét számolja ki. A refresh() folyamatosan frissíti a képernyőt, a usleep() a labdapattogás gyorsaságát határozza meg (jelenleg a másodperc egytizede). A clear() "kitörli" az előző labdát, így mindig csak az aktuálisan kirajzoltat látjuk a képernyőn (valójában nem törli ki, csak nem jeleníti meg). A többi részét a forráskódba ágyazott kommentekkel szerintem kellőképpen megmagyaráztam.

Labdapattogás if-ek nélkül: (A programot Dr. Bátfai Norbert Tanár Úr készítette, apró módosítás található csak benne.)

Megoldás forrása: [labdaifnelkul.c](#)

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

```
int main()
{
    int xj = 0, xk = 0, yj = 0, yk = 0;
    int mx = 80 * 2, my = 24 * 2;

    WINDOW *terminal;
    terminal = initscr();
    noecho();
    cbreak();
    nodelay(terminal, true);

    for (;;)
    {
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;

        yj = (yj - 1) % my;
        yk = (yk + 1) % my;
        clear ();
        mvprintw (0, 0,
                  " ←↵
                  -----
                  ");
        mvprintw (24, 0,
                  " ←↵
                  -----
                  ");
        mvprintw (abs ((yj + (my - yk)) / 2),
                  abs ((xj + (mx - xk)) / 2), "O");
        refresh();
        usleep (100000);
    }
}
```

Egy rendkívül ötletes megoldást láthatunk fentebb. Lényege, hogy a labda mozgásútjához két számsort tartunk számon, az egyik csökken, a másik növekszik. Ezen számsorok összehasonlításával megkapjuk a labda mozgásának pályáját (a függőleges és vízszintes koordinátákat). Ahhoz, hogy a labda a határon belül maradjon, a maradékos osztás (%) műveletét vesszük igénybe.

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 forrása: [bogomips.c](#)

Megoldás videó: <https://youtu.be/JJjiYUrCnh0>,

Megoldás forrása: [int1.cpp](#)

```
#include <iostream>

using namespace std;

int main()
{
    int a = 1;
    int osszeg = 0;

    while (a <= 1)
        osszeg++;
    cout << osszeg+1 << endl;
}
```

Megoldás forrása: [int2.cpp](#)

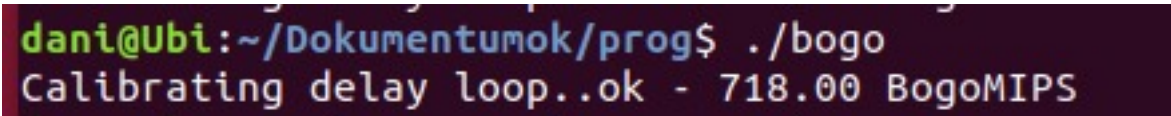
```
#include <iostream>

using namespace std;

int main()
{
    int a = 1;
    int osszeg = 0;
    do
    {
        osszeg++;
    } while(a <= 1);
    cout << osszeg << endl;
}
```

Két megoldást készítettem, az egyik előltesztelő, a másik hátultesztelő ciklust használ. Sokat nem lehet róluk mondani, egy int típusú változón bitshift műveleteket hajtanak végre, a lépések számát pedig kiírják az alapértelmezett outputra. Így ténylegesen megszámlálja, hogy az int típus 32 bit hosszú. Érdekesség, hogy a `sizeof(int)` beépített függvény ugyanígy visszaadja az int méretét, csak bájtban. Ezt a mennyiséget ha 8-cal megszorozzuk, akkor megkapjuk ugyanazt az eredményt, mint amit a fent látható két program ad eredményül.

A MIPS rövidítése: Microprocessor without Interlocked Pipeline Stages. A CPU sebességének egy kitalált mértékegysége. A fentebb linkelt algoritmussal úgy próbálják megsaccolni a processzorok sebességét, hogy megnézik, rendszerindításkor a benne lévő ciklus mennyi ideig fut. Mivel nem tényleges sebességet mér, így csupán egyfajta következtetés leszűrésére alkalmas, milyen gyors is a számítógépünkben található központi feldolgozóegység.



```
dani@Ubi:~/Dokumentumok/prog$ ./bogo
Calibrating delay loop..ok - 718.00 BogomIPS
```

Saját BogomIPS eredményem (Windows alatt 920)

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/jz71g8WpwFY>

Megoldás forrása: <src/Turing/google.c>

A PageRank a Google keresőmotor egyik legfontosabb összetevője. Lényegében a weblapokhoz rendel egy névleges értéket (számot), ami a weboldalak fontosságát reprezentálja. Larry Page és Sergey Brin, a Google alapítói találták ki, és a Google mind a mai napig használják ezt a módszert. A feltételezés az, hogy egy a készítő csak olyan hivatkozásokat helyez el egy oldalra, amiket megbízhatónak, fontosnak és relevánsnak talál. Ezzel úgymond szavazatot ad a hivatkozott oldalra, az algoritmus pedig azt veszi figyelembe, hogy az adott oldalra hány helyen "szavaztak" még. További érdekes tényeket olvashatunk a PageRank, valamint a Google megalapításáról a "How Google works" című könyvben.

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

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

double tavolsag(double PR[], double PRv[], int n){
    double osszeg = 0.0;
    for (int i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

    return sqrt(osszeg);
}

int main()
{
    int i, j;
    double T[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 PR_elozo[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};

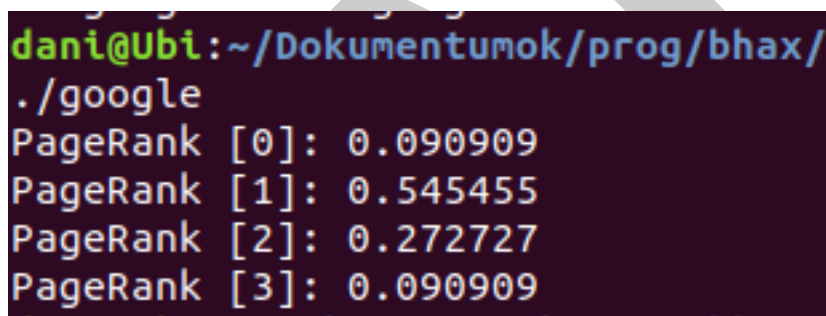
    for(;;)
    {
        for(i = 0; i < 4; i++)
        {
            PR[i] = 0.0;
            for(j = 0; j < 4; j++)
                PR[i] += (T[i][j] * PR_elozo[j]);
        }
    }
}
```



```
    }  
    if (tavolsag(PR, PR_elozo, 4) < 0.00000001)  
        break;  
    for(i = 0; i < 4; i++)  
        PR_elozo[i] = PR[i];  
}  
  
kiir(PR, 4);}
```

Mivel a PageRank értékek 0 és 1 közötti nem egész számok, így a double típust használjuk. Include-oljuk a math.h könyvtárat, hiszen szükség lesz az sqrt() függvény használatára. Létrehozunk egy mátrixot, ami az oldalak hierarchiáját szemlélteti. Ezen felül létrehozunk két másik tömböt. A PR nevűben tartjuk számon az oldal aktuális PageRank értékét, a PR_elozo nevűben pedig az előzőleg kiszámolt értékeket tárolja.

A végtelen ciklusunk tartalmaz két egymásba ágyazott for ciklust. Az elsőben kinullázzuk a PR tömb i-edik sorát, és miközben végigmegy a soron, hozzáadja az éppen aktuális tömbelem értékét megszorozva az adott oszlophoz tartozó oldal PR értékével. Végül egy feltétel segítségével megmondjuk, ha a 0.00000001 tetszőleges értéknél kisebb értéket ad vissza a tavolsag függvényünk, akkor megszakítja a végtelen ciklust, addig pedig a PR tartalmát átmásoljuk a PR_elozo tömbbe. Végül kiírjuk az eredményt a kiir() eljárás meghívásával.



```
dani@Ubi:~/Dokumentumok/prog/bhax/  
./google  
PageRank [0]: 0.090909  
PageRank [1]: 0.545455  
PageRank [2]: 0.272727  
PageRank [3]: 0.090909
```

4 Weblap PageRank értéke

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: <https://youtu.be/bzbUcWo8W2M>

Megoldás forrása: [monty.r](#)

```
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]){  
    ajtok = setdiff(c(1,2,3), kiserlet[i])  
  }else{  
    ajtok = setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))  
  }  
}
```

```
musorvezeto[i] = ajtok[sample(1:length(ajtok),1)]
}
nemvaltoztatesnyer = which(kiserlet == jatekos)
valtoztat = vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {
  holvaltoztat = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvaltoztat[sample(1:length(holvaltoztat),1)]
}

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

A Monty Hall probléma kifejezetten érdekes jelenség, valószínűleg sose gondoltam volna rá. Képzeljük el, hogy egy vetélkedőshow keretein belül választanunk kell 3 ajtó közül. Az egyik mögött ott rejlik az értékes nyeremény, de a másik kettő mögött nincs semmi. Mivel nem tudjuk, hogy melyik mögött van a díj, ezért random választunk a három közül egyet. Ekkor szimplán $1/3$ matematikai esélyünk van arra, hogy eltaláltuk a helyes ajtót. Ezután a műsorvezető kinyitja az egyik ajtót, ami mögött nincs semmi. Ekkor két lehetőségünk van: Vagy maradunk az eredeti döntésünk mellett, vagy megmászítjuk azt, és a másik ajtót választjuk.

Valószínűleg egy olyan ember, aki nincs tisztában a címben említett jelenséggel, rögtön rávágja, hogy mindegy, hogy megmászítjuk-e a döntésünket, hiszen nem változtat a nyerési esélyeinken. Viszont ha jobban belegondolunk, akkor könnyen beláthatjuk, hogy míg az első választási lehetőségnél 3 ajtó közül választhattunk (így $1/3$ esélyünk volt arra, hogy jól tippeltünk), ám a második lehetőségnél már csak kettő ajtó közül kell választanunk úgy, hogy a harmadikról már tudjuk, hogy nem jó (tehát ha megmászítjuk a választásunkat, akkor $2/3$ valószínűsége lesz annak, hogy a helyeset választjuk). Ettől függetlenül persze ha megmászítjuk a választásunkat, az nem lesz garancia arra, hogy eltaláljuk a helyes ajtót, de matematikailag nagyobb esélyünk lesz rá.

A fenti R program erre a scenárióra ad egy szimulációt. Az első sorban létrehoztunk egy változót, ami eltárolja, hogy hány kísérletet szeretnénk végrehajtani. Ezután létrehozunk két vektort, amik 1-től 3-ig vehetnek fel értékeket. Az első az ajtókat reprezentálja, a második a játékos döntését. Mindegyik annyiszor vehet fel értékeket, amennyi a kísérletek száma, valamint többször is felvehetik ugyanazt az értéket. Ezután a játékvezető döntéséről gondoskodunk. A vektor hossza akkora lesz, amennyi a kísérletek száma, ezután egy range-based for ciklussal végigmegyünk a kísérleteken, és a játékos döntésétől függően megadja a műsorvezető választását.

Ha a kísérletben a játékos által választott ajtó megegyezik a nyereményt rejtő ajtóval, akkor a $c(1,2,3)$ -ból kiveszi a kísérlet számát, és az ajtok-ban a másik kettő szám marad - ezek közül az egyiket kapja meg a műsorvezető. Viszont ha a játékos által választott mögött nincs nyeremény, akkor kizárásos alapon a műsorvezetőnek muszáj a másik, egyértelmű ajtót választania, ami mögött szintén nincs semmi. Így az ajtok vektor ezt a számot kapja meg, amit aztán a műsorvezető is.

Ezután megnézzük, hogy hányszor nyer a játékos, ha megváltoztatja az első döntését. Ehhez szintén létrehozunk egy vektort, ami a kísérletek számával lesz megegyező nagyságú, és hasonló módon, egy for

ciklussal végigmegy a kísérleteken. Ilyenkor csak azt az ajtót választhatja, amelyiket se a műsorvezető, se a játékos nem választott elsőre. A változtat és nyer vektorba bepakoljuk ezeket az eseteket. Vagyis azokat az eseteket, amikor a másodjára választott ajtó megegyezik azzal, amelyik mögött a nyeremény van.

A végén kiírjuk a képernyőre a kísérletek számát, valamint azoknak az eseteknek a számát, amikor a játékos megváltoztatta a döntését és nyert, illetve azokat, amikor nem. Láthatjuk, hogy az előbbiből lényegesen több van.

```
[1] "Kísérletek száma: 10000"  
> length(nemvaltoztatesnyer)  
[1] 3381  
> length(valtoztatesnyer)  
[1] 6619  
> length(nemvaltoztatesnyer) / length(valtoztatesnyer)  
[1] 0.5108022  
> length(nemvaltoztatesnyer) + length(valtoztatesnyer)  
[1] 10000
```

10000 kísérletre az eredmény

2.8. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó: <https://youtu.be/DjGwYuIwe2A>

(A programot Dr. Bátfai Norbert Tanár Úr készítette, apró módosítás található csak benne.) Megoldás forrása: [brun.r](#) 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:

$(n+1)!+2$, $(n+1)!+3$, ..., $(n+1)!+n$, $(n+1)!+(n+1)$ 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 $(n+1)*\text{valamennyi}+(n+1)$, 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 $(1/3+1/5)+(1/5+1/7)+(1/11+1/13)+\dots$ 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 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
  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-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 képz:

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

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

```
> 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] 0.5333333 0.3428571 0.1678322
```

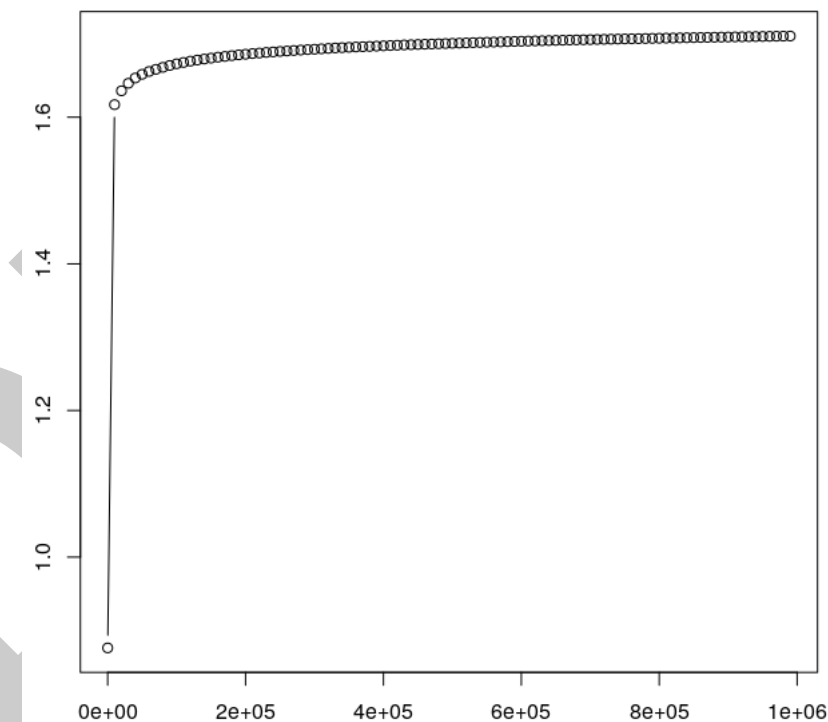
Nincs más dolgunk, mint ezeket a törtet összeadni a `sum` függvénnyel.

```
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>
-

2.9. Minecraft MALMÖ (csiga)

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

DRAFT

3. fejezet

Helló, Chomsky!

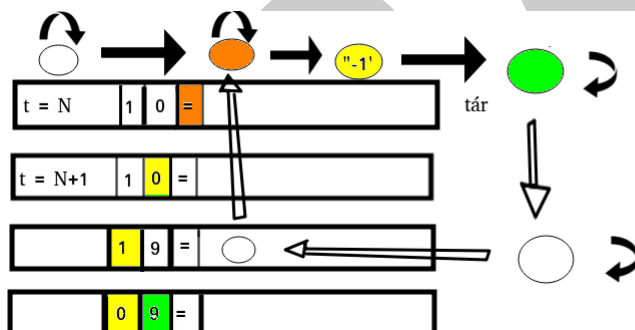
3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

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

Megoldás forrása: [unaris.cpp](https://github.com/unaris/cpp)

Az unáris (egyes) számrendszer a legegyszerűbb számrendszer. Lényegében az ujjunkon való számolásakor is ezt használjuk. Amekkora számot akarunk kifejezni így, annyi 1-est írunk le. Például a decimális 6-os számot úgy konvertáljuk unárisba, hogy hatszor leírunk egy 1-est: 111111.



A fenti állapotgráf egy "végtelen szalagos és memóriás" Turing gépet mutat be, ahogy decimálisból unáris számrendszerbe vált át. Ehhez azt csinálja, hogy a számból addig von ki 1-et, amíg az 0 nem lesz, és ezeket az egyeseket a művelet közben a tárho írja ki. A művelet az utolsó számjegyen kezdődik, ha az 0-ás, akkor 9-cel megy tovább, ha ettől eltérő, akkor mindig eggyel kevesebbel megy a zölddel jelölt állapotba. Ez addig folytatódik, amíg újra 0-ást kapunk, ezután a következő számjegyre tér át, és végrehajtja ugyanezeket a folyamatokat, míg a végén a szám első számjegyénél is 0-ást kap.


```
#include <iostream>

using namespace std;

void konvertal(int szam)
{
    cout << endl;
    for(int i = 0; i < szam; i++)
    {
        cout << "1";
    }
    cout << endl;
}

int main()
{
    int szam;

    cout << "Adj meg egy szamot!" << endl;
    cin >> szam;

    konvertal(szam);
}
```

Készítettem egy nagyon egyszerű átváltó C++ programot. Tényleg nagyon egyszerű, ezért nem szeretném túlmagyarázni. A main-ben bekérek a felhasználótól egy számot, ezután meghívom a konvertal eljárást erre a számra, ami egy for ciklus segítségével annyi 1-et ír az alapértelmezett outputra, amekkora a szám volt.

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 forrása: A fejezetbe ágyazva.

S, x, y leggyakoribb változók
 a, b, c leggyakoribb konstansok

$S \Rightarrow abc, S \Rightarrow aXbc, Xb \Rightarrow bX, Xc \Rightarrow Ybcc, bY \Rightarrow Yb,$
 $aY \Rightarrow aax, aY \Rightarrow aa$

$S (S \Rightarrow aX^bbc)$
 $aX^bc (Xb \Rightarrow bX)$
 $abXc (Xc \Rightarrow Ybcc)$
 $abYbcc (bY \Rightarrow Yb)$
 $aYbcc (aY \Rightarrow aa)$
 $aaabcc$

A, B, C legyenek változók
 a, b, c legyenek konstansok
 $A \Rightarrow aAB, A \Rightarrow aC, CB \Rightarrow bCc, cB \Rightarrow Bc, C \Rightarrow bc$
 $A (A \Rightarrow aAB)$
 $aAB (A \Rightarrow aAB)$
 $aaABB (A \Rightarrow aAB)$
 $aaaABBB (A \Rightarrow aC)$
 $aaaaCBBB (CB \Rightarrow bCc)$
 $aaaabCbBB (cB \Rightarrow Bc)$
 $aaaabC~~BBB~~ (cB \Rightarrow Bc)$
 $aaaaaCbBBc (CB \Rightarrow bCc)$
 $aaaaabCbBc (cB \Rightarrow Bc)$
 $aaaaabCbcc (cB \Rightarrow bCc)$
 $aaaaabbbccc (c \Rightarrow bc)$
 $aaaaabbbcccc$

Noam Chomsky, egy amerikai nyelvész nevéhez nem csak a fejezet címe, hanem a generatív grammatika, valamint a Chomsky-hierarchia is köthető. Utóbbival formális nyelveket szoktak lehet osztályozni. Úgy működik, hogy nyelv részeit kifejezőerő alapján osztályozza, és az erősebb osztályok elemei képesek megalkotni a gyengébb osztályok elemeit. Az alsóbb osztályokat a levezetési szabályok alkalmazásával állíthatjuk elő. Addig kell ezeket a szabályokat alkalmazni, míg a kifejezés csak konstansokból nem fog állni. Fentebb látható két általam levezetett példa, az első az $a^2b^2c^2$ nyelvet generálja a levezetésben, a második az $a^4b^4c^4$ nyelvet.

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://youtu.be/lBeq2_AnbsQ

Megoldás forrása: hiba.c

A Backus-Naur-Form egy nyelvek szintaxisának leírására szolgáló eszköz. A működését [ebből](#) a videóból tanultam meg, ezután nem volt bonyolult megalkotni a C utasítást így.

```
<utasítás>
<címkézett_utasítás> ::= <azonosító> | <case> | <default>
<kifejezés_utasítás> ::= <kifejezés>
<összetett_utasítás> ::= <deklarációs_lista> | <utasítás_lista>
<deklarációs_lista> ::= <deklaráció> | <deklarációs_lista deklaráció>
<utasítás_lista> ::= <utasítás> | <utasítás_lista utasítás>
<kiválasztó_utasítás> ::= if | if else | switch
<iterációs_utasítás> ::= while | do while | for
<vezérlésátadó_utasítás> ::= goto | continue | break | return
```

Néhány példa olyan utasításokra, amik bizonyos verziójú C fordítóknál hibát adnak:

```
#include <stdio.h>

int main()
{
    //komment                //-rel nem lehet kommentelni, csak /**/- ←
    rel jó

    for (int i = 0; i < 5; i++) //nem lehet for cikluson belül deklarálni
    {
        long long int a = 2;    //nem támogatott típus
    }
    gets(a);                    //warning, c11-gyel már hiba
}
```

```

dani@Ubi:~/Dokumentumok/prog/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/src/Chomsky$ cc hiba.c -std=c89
hiba.c: In function 'main':
hiba.c:5:5: error: C++ style comments are not allowed in ISO C90
    //komment                //rel nem lehet kommentelni, csak /**/-rel jó
    ^
hiba.c:5:5: error: (this will be reported only once per input file)
hiba.c:7:5: error: 'for' loop initial declarations are only allowed in C99 or C11 mode
    for (int i = 0; i < 5; i++) //nem lehet for cikluson belül deklarálni
    ^~~~~
hiba.c:7:5: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 to compile your code
hiba.c:11:5: warning: 'gets' is deprecated [-Wdeprecated-declarations]
    gets(a);                //warning, c11-gyel már hiba
    ^~~~~
In file included from hiba.c:1:0:
/usr/include/stdio.h:577:14: note: declared here
    extern char *gets (char *__s) __wur __attribute__ ((__deprecated__));
                  ^~~~~
hiba.c:11:10: error: 'a' undeclared (first use in this function)
    gets(a);                //warning, c11-gyel már hiba
    ^
hiba.c:11:10: note: each undeclared identifier is reported only once for each function it appears in
dani@Ubi:~/Dokumentumok/prog/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/src/Chomsky$ cc hiba.c -std=c99
hiba.c: In function 'main':
hiba.c:11:5: warning: 'gets' is deprecated [-Wdeprecated-declarations]
    gets(a);                //warning, c11-gyel már hiba
    ^~~~~
In file included from hiba.c:1:0:
/usr/include/stdio.h:577:14: note: declared here
    extern char *gets (char *__s) __wur __attribute__ ((__deprecated__));
                  ^~~~~
hiba.c:11:10: error: 'a' undeclared (first use in this function)
    gets(a);                //warning, c11-gyel már hiba
    ^
hiba.c:11:10: note: each undeclared identifier is reported only once for each function it appears in
dani@Ubi:~/Dokumentumok/prog/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/src/Chomsky$ cc hiba.c -std=c11
hiba.c: In function 'main':
hiba.c:11:5: warning: implicit declaration of function 'gets'; did you mean 'fgets'? [-Wimplicit-function-declaration]
    gets(a);                //warning, c11-gyel már hiba
    ^~~~~
    fgets
hiba.c:11:10: error: 'a' undeclared (first use in this function)
    gets(a);                //warning, c11-gyel már hiba
    ^
hiba.c:11:10: note: each undeclared identifier is reported only once for each function it appears in

```

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!

dr. Bátfai Norbert megoldása: https://youtu.be/9KnMqrkj_kU (15:01-től).

Megoldás videó: <https://youtu.be/BXsm8xHXSus>

Megoldás forrása: [lex.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);
}

```

```
}
```

A kód 3 részből áll, ezek dupla % jellel vannak elválasztva. Ez egy lexer, ami C programokat ír helyettünk. Az elején include-oljuk az stdio.h könyvtárat, valamint deklarálunk egy realnumbers nevű egész típusú változót. Ezután jön a definíció. A digit [0-9] a 0-tól 9-ig tartó számjegyekből álló karaktercsokrot fogja definiálni. A következő részben a fordítási szabályok találhatók. Úgy kell értelmezni, hogy a digitből bármennyi (akár 0 is) lehet (ezt a *-gal definiálja), aztán a „,” jelet írjuk mellé, ami a tizedesvesszőt (pontot) jelöli, aztán még egy digit jön, de ennek a végén már egy + jel van, ami azt jelenti, hogy akárhány darab számjegy lehet utána, de legalább egynek lennie kell. A végén az egész kérdőjelben van, vagyis opcionális (nem muszáj tizedesvesszővel elválasztani egy számot).

Ezután megnöveljük a változó értékét eggyel, majd kiíratjuk stringként az eddig feldolgozott adatokat, majd a számot is [] zárójelek között. Az atof() függvény stringből konvertál lebegőpontos (double számokat). Ezután a mainben elindítjuk a lexikális elemzőt, majd a végén kiírjuk, hogy hány darab szám volt a beolvasott adatban.

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

dr. Bátfa Norbert megoldása: https://youtu.be/06C_PqDpD_k

Megoldás videó: <https://youtu.be/B7Crc8x8lQE>

Megoldás forrása: leetspeak.com/

```
%{
#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", "|\\"}, {"n", "/\\"}, {"n", "/V"}},
{'o', {"o", "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", "\\"}, {"v", "\\"}, {"v", "\\"}},
{'w', {"w", "VV", "\\"}, {"w", "(/\\"}}},
{'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 program lényegében azt csinálja, hogy a beírt szöveget karakterenként átírja egy meghatározott szótár segítségével, ezáltal „titkosítva” azt. Az előző feladathoz hasonlóan ez a program is 3 részből áll. Az elején include-oljuk a szükséges könyvtárakat, valamint létrehozunk egy struktúrát, ami tartalmaz egy `char`-t, amiben az aktuális betű van, amit beírtunk, aztán egy mutató tömböt, ami a beírt karakterhez tartozó cserelehetőségeket tartalmazza, valamint egy tömböt, ami tartalmazza a „szótárat”, vagyis a karaktereket, valamint a hozzájuk tartozó cseréket. Ennek a tömbnek nincs meghatározva a mérete, ugyanis a fordító elvégzi ezt a számolást helyettünk.

A második részében definiáljuk a szabályokat, ami egy szimpla „.”. Ez annyit tesz, hogy a beolvasott szöveget karakterenként hasonlítjuk össze a tömbben lévő szótárral, és ha talált egyezést, akkor egy randomizált folyamat végén a 4 lehetőség közül egyre kicseréli a beírt karaktert. A harmadikban van a `main`, amiben inicializáljuk a randomizálást, valamint elindítjuk a lexikális elemzőt.

```
dani@Ubi:~/Dokumentumok/prog/bhax/thematic_tutorials/bhax_textbo
gramozod/src/Chomsky$ ./leet
Imádom a prog1-et, és imádom ezt a könyvet szerkeszteni! :)
1mád044 4 pr0gI-3t, és 1mád0m 3z'|' 4 kö|\|v3t sz3rk3szt3n1! :)
```

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. (Miótan 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 megváránzásra, elkapja valamelyiket esetleg a splint vagy a frama?

- i.

```
if(signal(SIGINT, SIG_IGN) != SIG_IGN)
    signal(SIGINT, jelkezezo);
```
- 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?

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

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

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

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

Megoldás forrása: A fejezetbe ágyazva.

Legnagyobb öröömömre előkerestem a logika jegyzetemet, az ott tanultak alapján pedig megoldottam a feladatot.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

Egy a-val jelölt egész típusú változót.

- ```
int *b = &a;
```

Egy "b"-vel jelölt egészre mutató pointert, ami az "a" változó címét tartalmazza.

- ```
int &r = a;
```

Egy "r"-rel jelölt egész típusú referencia, ami az "a" értékét veszi föl.

- ```
int c[5];
```

Egy "c"-vel jelölt 5 elemű, egészeket tartalmazó tömb.

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

Egy 5 elemű egész számokat tartalmazó tömbre mutató pointer, ami a "c" tömbre mutat. Vagyis egy egész számokat tartalmazó tömb referenciája.

- ```
int *d[5];
```

Egy "d"-vel jelölt 5 elemű egészekre mutató pointereket tartalmazó tömb.

- ```
int *h ();
```

Egy egészre mutató pointert visszaadó függvény.

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

Egy függvényre mutató pointer, aminek a visszatérési értéke egy egészre mutató pointer.

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

Egy egész számot kérő function ami egy pointert ad vissza egy function-re, ami két egészet kér, és egy egészet ad vissza.

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

Egy egészet kérő function-re mutató pointer, ami egy pointert ad vissza egy function-re, ami két egészet kér, és egy egészet ad vissza.

Megoldás videó:

Megoldás forrása:

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](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr.c), [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr2.c](https://github.com/bhax/thematic_tutorials/blob/master/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;  
}
```

3.9. 9. Minecraft MALMÖ (csiga diszkrét)

Megoldás videó:

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 tárbán!

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;

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

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

    for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
```

```
        tm[i][j] = i * (i + 1) / 2 + j;

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

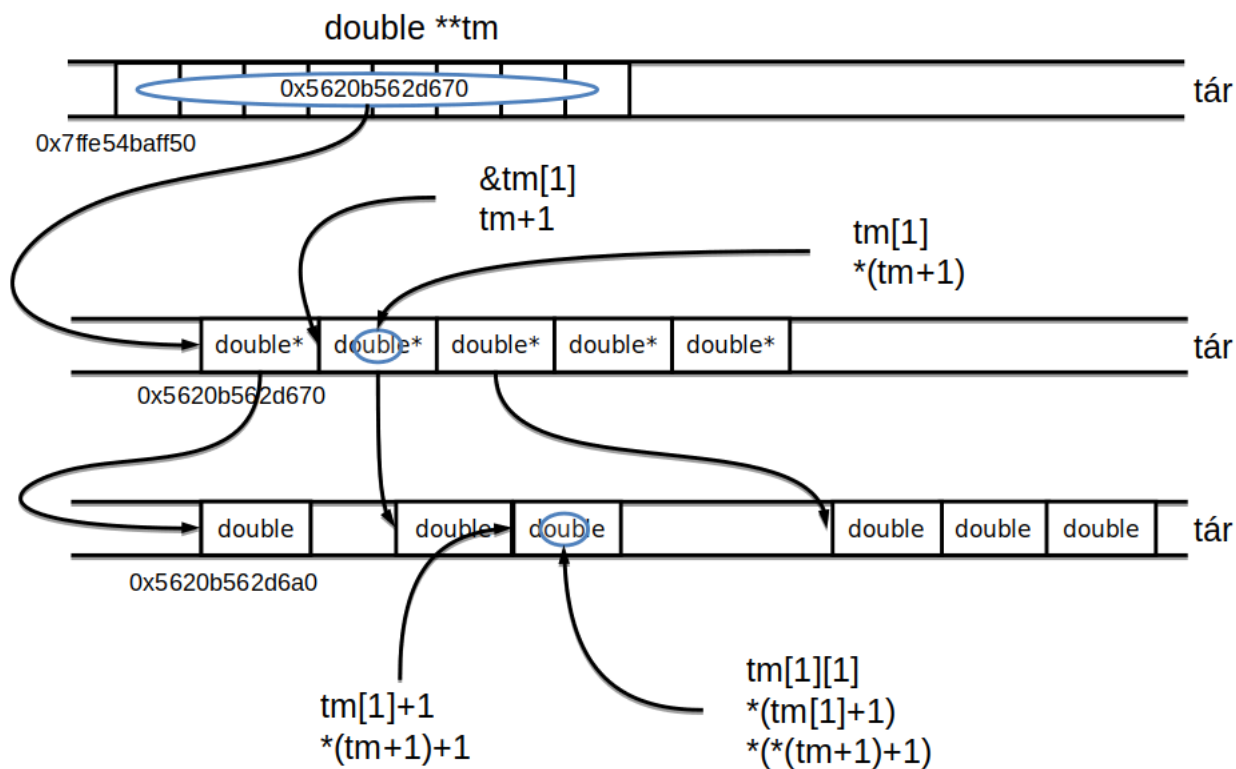
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

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

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

free (tm);

return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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:

Tanulságok, tapasztalatok, magyarázat...

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...

4.7. Minecraft MALMÖ (érzékelés)

Megoldás videó: <https://youtu.be/MkZlcV5Sdx8>

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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 $z_{n+1} = z_n^2 + c$, ($0 \leq n$) képlet alapján úgy, hogy a c az éppen vizsgált rácspont. A z_0 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 rácpont nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végtelen sok z -t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácponthoz. Ha közben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácpont a halmaz része. (Színes meg úgy lesz a kép, hogy változtatosan színezzük, például minél későbbi z -nél lép ki a körből, annál sötétebbre).

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

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

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

Megoldás forrása:

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhaxor/attention-raising/Mandelbrot/3.1.2.cpp](https://github.com/bhaxor/attention-raising-Mandelbrot) 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 ↵
-0.01947381057309366392260585598705802112818 ↵
-0.0194738105725413418456426484226540196687 ↵
0.7985057569338268601555341774655971676111 ↵
0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ↵
0.4127655418209589255340574709407519549131 ↵
0.4127655418245818053080142817634623497725 ↵
0.2135387051768746491386963270997512154281 ↵
0.2135387051804975289126531379224616102874
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --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>

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;
}
```

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbGRzY76E>

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

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

A különbség a **Mandelbrot halmaz** és a Julia halmazok között az, hogy a komplex iterációban az előbbiben

a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

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

        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam

        reC = a + k * dx;
        imC = d - j * dy;
        std::complex<double> c ( reC, imC );

        std::complex<double> z_n ( 0, 0 );
        iteracio = 0;

        while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
        {
            z_n = z_n * z_n + c;

            ++iteracio;
        }
    }
}
```

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

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon
    for ( int k = 0; k < szelesseg; ++k )
    {
        double reZ = a + k * dx;
        double imZ = d - j * dy;
        std::complex<double> z_n ( reZ, imZ );

        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {
            z_n = std::pow(z_n, 3) + cc;
            if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
            {
                iteracio = i;
                break;
            }
        }
    }
}
```

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ←
BATF41 HAXOR STR34M" --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/gvaqijHIRUs>

Megoldás forrása: [bhax/attention-raising/CUDA/mandelpngc_60x60_100.cu](https://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 iteráció bejárta z_n komplex számokat!

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

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJssY>, 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>

DRAFT

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 elspájzold és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltérve kiszámolt szám.

Megoldás videó:

Megoldás forrása:

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

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:

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. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

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

[?]

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

10.3. Programozás

[BMECPP]

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.

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

DRAFT

11.3. Általános

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

11.4. C

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

11.5. C++

[BMECPP] Benedek, Zoltán És 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.