

# **Magasabb szintű programozási nyelvek I.**

## **Labor jegyzőkönyv**

Ed. BHAX, DEBRECEN,  
2020. február 22, v. 0.0.5

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>  Magasabb szintű programozási nyelvek I.		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Bátfai, Margaréta, Ács Human, Person	2020. március 3.	

## 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 <a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2020-02-15	Forked and initied bhax derived <a href="#">bhax-derived</a>	rkeeves

## 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

<b>I. Bevezetés</b>	<b>1</b>
<b>1. Vízió</b>	<b>2</b>
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
<b>II. Tematikus feladatok</b>	<b>4</b>
<b>2. Helló, Turing!</b>	<b>6</b>
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	10
2.3. Változók értékének felcserélése	13
2.4. Labdapattogás	16
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	20
2.6. Helló, Google!	24
2.7. A Monty Hall probléma	33
2.8. 100 éves a Brun tétel	34
<b>3. Helló, Chomsky!</b>	<b>40</b>
3.1. Decimálisból unárisba átváltó Turing gép	40
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	40
3.3. Hivatkozási nyelv	40
3.4. Saját lexikális elemző	41
3.5. Leetspeak	41
3.6. A források olvasása	43
3.7. Logikus	44
3.8. Deklaráció	45

<b>4. Helló, Caesar!</b>	<b>48</b>
4.1. double ** háromszögmátrix	48
4.2. C EXOR titkosító	50
4.3. Java EXOR titkosító	50
4.4. C EXOR törő	51
4.5. Neurális OR, AND és EXOR kapu	51
4.6. Hiba-visszaterjesztéses perceptron	51
<b>5. Helló, Mandelbrot!</b>	<b>52</b>
5.1. A Mandelbrot halmaz	52
5.2. A Mandelbrot halmaz a std::complex osztállyal	53
5.3. Biomorfok	55
5.4. A Mandelbrot halmaz CUDA megvalósítása	59
5.5. Mandelbrot nagyító és utazó C++ nyelven	59
5.6. Mandelbrot nagyító és utazó Java nyelven	60
<b>6. Helló, Welch!</b>	<b>61</b>
6.1. Első osztályom	61
6.2. LZW	61
6.3. Fabejárás	61
6.4. Tag a gyökér	61
6.5. Mutató a gyökér	62
6.6. Mozgató szemantika	62
<b>7. Helló, Conway!</b>	<b>63</b>
7.1. Hangyaszimulációk	63
7.2. Java életjáték	63
7.3. Qt C++ életjáték	63
7.4. BrainB Benchmark	64
<b>8. Helló, Schwarzenegger!</b>	<b>65</b>
8.1. Szoftmax Py MNIST	65
8.2. Mély MNIST	65
8.3. Minecraft-MALMÖ	65

<b>9. Helló, Chaitin!</b>	<b>66</b>
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .	66
9.2. Gimp Scheme Script-fu: króm effekt . . . . .	66
9.3. Gimp Scheme Script-fu: név mandala . . . . .	66
<b>10. Helló, Gutenberg!</b>	<b>67</b>
10.1. Programozási alapfogalmak . . . . .	67
10.1.1. Adattípusok . . . . .	67
10.1.2. A nevesített konstans . . . . .	76
10.1.3. A változó . . . . .	76
10.1.4. Alapelemek az egyes nyelvekben . . . . .	77
10.2. Programozás bevezetés . . . . .	77
10.3. Programozás . . . . .	77
<b>III. Második felvonás</b>	<b>78</b>
<b>11. Helló, Arroway!</b>	<b>80</b>
11.1. A BPP algoritmus Java megvalósítása . . . . .	80
11.2. Java osztályok a Pi-ben . . . . .	80
<b>IV. Irodalomjegyzék</b>	<b>81</b>
11.3. Általános . . . . .	82
11.4. C . . . . .	82
11.5. C++ . . . . .	82
11.6. Lisp . . . . .	82
11.7. My Little Ponys . . . . .	82

# Ábrák jegyzéke

2.1. For syntax . . . . .	7
2.2. Alma no flag fail . . . . .	9
2.3. Alma With flag . . . . .	9
2.4. builtins . . . . .	10
2.5. shiftl_uchar . . . . .	21
2.6. shiftr_uchar . . . . .	21
2.7. 2s complement . . . . .	23
2.8. ullshift . . . . .	23
2.9. bogomips kimeneten . . . . .	24
2.10. pagerank kapcsolati irányított multigráf . . . . .	25
2.11. pagerank link mátrix sanity check . . . . .	26
2.12. pagerank vektor . . . . .	27
2.13. pagerank trf . . . . .	28
2.14. pagerank cpp konvergál . . . . .	30
2.15. pagerank openoffice konvergál . . . . .	30
2.16. A $B_2$ konstans közelítése . . . . .	37
2.17. Az c++ $B_2$ konstans közelítés eredményei . . . . .	39
4.1. A double ** háromszögmátrix a memóriában . . . . .	50
5.1. A Mandelbrot halmaz a komplex síkon . . . . .	52
10.1. Foo és Foo Main . . . . .	72
10.2. FooABI és FooABI Main . . . . .	73



# 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 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**

DRAFT

# 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**

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

## 2. fejezet

# Helló, Turing!

### 2.1. Végtelen ciklus

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

Megoldás videó: No link

Megoldás forrása:

- [Basic infinite for loop](#)
- [For with linux sleep](#)
- [While infinite](#)
- [Macros for OS call handling](#)

Végtelen ciklusokra gyakorlatban sok példa akad: szerverek, game loop-ok, interaktív interpreter terminálhoz. A végtelen ciklus azonban olykor nem várt. Például lehet bug, de akár lehet a természetes működés része. Erre jó példa a Tanár Úr által említett PageRank algoritmus. Én azonban, a hozzáadott munka miatt másik példát hoznék: mérnöki CAE végeselemes számítások (pl. Ansys) esetén egyes esetekben a megoldás nem konvergál  $n$  lépés után sem, és ez előre nem látható be. Erre a célra a felhasználó megadhat vészhelyzeti kilépési konvergencia kritériumokat. Ez nem matematikai kényszerrel jelent, hanem szimplán, ha  $n$  iteráció, vagy  $t$  idő elteltével a megoldáshoz nem jutunk közelebb, akkor a program ki break-el ciklusból.

Alább egy minimális végtelen ciklus látható for-ral. Amit vegyünk észre, az az hogy az egész "üres". Ezt a [for loop szintaktikai definíciója teszi lehetővé](#), hisz nyíltan kifejezi az optional, hogy nem kötelező megadni. Ez persze egyéb hatásokkal is jár, például, hogy a `for ( ; ; )`-ban használhatjuk az ehhez képest külső scope-ban deklarált változókat.



## for loop

Executes *init-statement* once, then executes *statement* and *iteration\_expression* repeatedly, until the value of *condition* becomes false. The test takes place before each iteration.

### Syntax

*formal syntax:*

```
attr(optional) for ( init-statement condition(optional) ; iteration_expression(optional) ) statement
```

*informal syntax:*

```
attr(optional) for ( declaration-or-expression(optional) ; declaration-or-expression(optional) ; expression(optional) )  
statement
```

2.1. ábra. For syntax

```
1 main ()  
2 {  
3     for (;;) ;  
4  
5     return 0;  
6 }
```

Alább a fenti példa látható egy `while` loop-pal. Értelmszerűen a `conditional`-ban mivel egy `true` bool literal-t írtuk, így ez mindig igazra fog kiértékelődni.

```
1 #include <stdbool.h>  
2 int  
3 main ()  
4 {  
5     while(true);  
6  
7     return 0;  
8 }
```

Alább egy `while`-t használtunk, de mostmár alszunk is. A `sleep` viszont sajnos OS dependens call (mi sem mondja el jobban mint az `unistd` include-olása). Mivel az OS feladata az ütemezés, plusz a megszakítások kezelése (hisz ugye az egész lelke az időzítés ezáltal a timer). `sleep` esetén jelezzük az OS-felé, hogy nem kérünk CPU időt (azaz váltsa a process status-t), viszont emellett viszont a megfelelő idő elmultával (megszakításos alapon) újból kerülünk számításra várakozóba. A probléma természetesen annyi, hogy scheduling-tól függően, lehet soha nem kerülünk vissza még az idő letelte után sem CPU-ra.

```
1 #include <unistd.h>  
2 int  
3 main ()  
4 {  
5     for (;;) ;
```

```
6     sleep(1);
7
8     return 0;
9 }
```

Mindenesetre az alvás egy OS specifikus dolog, pl.: [nanosleep](#). Ha több fajta OS-n kell futni, ahhoz sajnos trükközni kell. Egy barbár megoldás a preprocesszor használata. Alábbi példában az látható, hogy preprocessor által értelmezett szöveget is elhelyeztem a fájlban. Technikailag ez annyit tesz, hogy compile előtt a preprocessor elődolgozza a forrásfájlt, és ez példánkban azzal jár

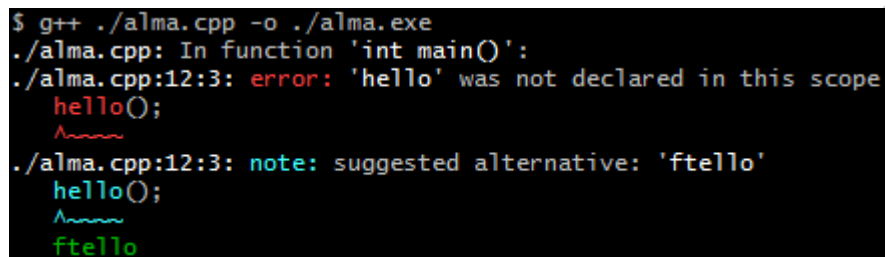
```
1  #ifdef __linux__
2      #include <unistd.h>
3  #elif _WIN32
4      #include <windows.h>
5  #else
6      #error Unsupported
7  #endif
8
9  void SleepProxy(int sleepMs)
10 {
11     #ifdef __linux__
12         usleep(sleepMs * 1000);
13     #elif _WIN32
14         Sleep(sleepMs);
15     #endif
16 }
17
18 int
19 main ()
20 {
21     for (;;)
22         SleepProxy(1000);
23
24     return 0;
25 }
```

Egyébként a gcc mellé betudunk passzolni flageket. Például `-DALMA`-val gyakorlatilag azt mondjuk a preproceszornak, hogy vegye úgy mintha `#define ALMA`. Ez pontosan annyira szép mint amennyire elegáns, de ez van. Nézzük csak meg ezt az almát!

```
1  #include <iostream>
2
3  #ifdef ALMA
4  void hello()
5  {
6      std::cout << "Hello world!" << std::endl;
7  };
8  #endif
9
10 int main ()
11 {
```

```
12  hello();  
13  return 0;  
14  }
```

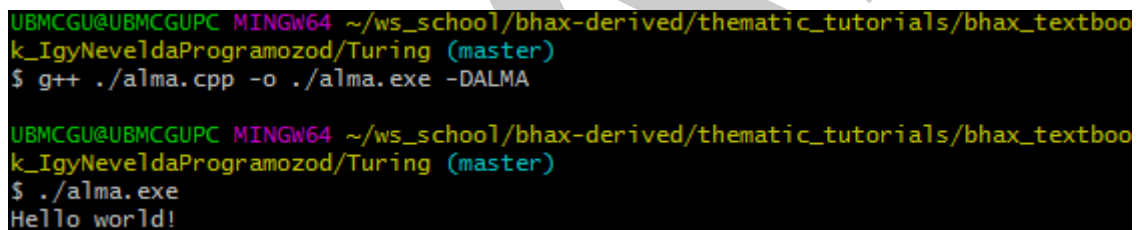
Láthatóan ez nem fog lefordulni ha nincs definiálva az ALMA, de azért tegyünk próbát!



```
$ g++ ./alma.cpp -o ./alma.exe  
./alma.cpp: In function 'int main()':  
./alma.cpp:12:3: error: 'hello' was not declared in this scope  
    hello();  
    ~~~~~  
./alma.cpp:12:3: note: suggested alternative: 'ftello'  
    hello();  
    ~~~~~  
    ftello
```

2.2. ábra. Alma no flag fail

Miért azt írja hogy `hello was not declared in scope`? Nos a preprocesszor végigment a forrásfájlon, és, mivel nem volt ALMA definiálva, ezért a compilation stage-ben az a kód részlet ami `#ifdef`-be volt zárva nem létezett. Próbáljuk ki azt hogy `g++ alma.cpp -o alma -DALMA`! Az alábbi screenshot-ról látszik, hogy mivel lusta vagyok, ezért a git bash-ből csináltam.



```
UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo  
k_IgyNeveldaProgramozod/Turing (master)  
$ g++ ./alma.cpp -o ./alma.exe -DALMA  
  
UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo  
k_IgyNeveldaProgramozod/Turing (master)  
$ ./alma.exe  
Hello world!
```

2.3. ábra. Alma With flag

Ha belenézel az STL-be, akkor az is tele lesz ilyenekkel, hisz különböző OS-ekkel kell tudni használni, és a rendszerhívások általában eltérőek. Egyébként ha valami IDE-vel dolgozol, ott gyakran még szépen be is szürkíti azokat a részeket a kódban amelyeket ki fog hagyni. Itt semmi mágia nem történik, hanem simán nem a forráskódod megy a compiler-nek, hanem előtte van egy preprocesszor, ami feldolgozza ezeket az utasításokat. Makrók ugyanilyenek. Tehát nem runtime kiértékelhető az `#ifdef` hanem egy utasítás a preprocesszornak. Leggyakrabban `#ifdef`-el header guard-ként fogsz találkozni (vele kb. ekvivalens a nem mindenhol támogatott `#pragma once`).

Na jó, de ez nem válaszolja meg a kérdést, hogy miért nem bírja a gépem OBS-el együtt a notepad++-t.

Mármint, akarom mondani, milyen ügyes kis dolog ez a preprocesszor! De csak ennyit tud? A válasz nem. Alábbi példában filename, linenumber-t íratunk ki.

```
1  #include <iostream>  
2  
3  int main ()  
4  {
```

```
5  std::cout << "Hello from " << __FILE__ << ", I'm from line " << __LINE__ << "\n";  
    << std::endl;  
6  return 0;  
7  }
```

```
$ ./builtins.exe  
Hello from ./builtins.cpp, I'm from line 5
```

## 2.4. ábra. builtins

Most pedig következzen egy összevagdossott code snippet egy régi projektéből. A lényeg annyi, hogy unit test-eléshez, egy makró alapú library-ról volt szó. (Csak poénból, tudom hogy van gtest).

```
1  #define EXPECT(case, arg0, arg1, pred, msg) \  
2  { bool res = pred(arg0, arg1); case.do_test(false, __FILE__, __LINE__, res, msg, \  
    arg0, arg1); }  
3  
4  #define EXPECT_EQ(case, arg0, arg1) EXPECT(case, arg0, arg1, pred_eq, "Expect \<br>failed ==")  
5  
6  void test_is_subs(TestCase& tc)  
7  {  
8      EXPECT_EQ(tc, true, is_ss Subs("ab", "ab"));  
9  }
```

A fenti snippetben látszik, hogy a `#define`-al meghatározott dolgok többek mint "változók". Ezeket preproceszor makróknak hívjuk, és az előfeldolgozáskor expandáljuk őket. Azaz először az `EXPECT_EQ` expandálódik `EXPECT(case, arg0, arg1, pred_eq, "Expect failed ==")`. Az `Expect`-ben meg egy pici scope-ot létrehozunk. A lényeg ebből annyi, hogy nem kell megijedni tőle, jóra is lehet használni. Annyi hogy az expanzióval óvatosan, mert elég nehéz debuggolni. (ugyanolyan szívás mint az STL-es template alapú hibákat, azaz nem egy one liner lesz.)

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

Lehet-e írni olyan programot amely minden programról megmondja, hogy lefut-e?

Megoldás videó: [No link](#)

Megoldás forrása:

- [Turing 100 \(pseudocode\)](#)
- [Turing 1000 \(pseudocode\)](#)

A probléma elég fontos. Miért is? Nos, a probléma az, hogyha ez nem lehetséges, akkor soha nem fogjuk tudni bizonyítani az összes program helyességét. 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:

A borbély az aki azok haját vágja, akik nem teszik meg ezt saját maguknak. De a borbély vágja-e a saját haját? Nos, tegyük fel levágja: Ebben az esetben ő levágta a haját egy olyan embernek aki egyébként ezt megteszi saját magának. Ok, tegyük fel nem vágja: Viszont ebben az esetben nem vágja a saját haját, aka lehetne a saját ügyfele, viszont ha most levágja akkor kezdődik az egész előlről [Russel's paradox](#).

Ezzel a problémával sok helyen találkozhatunk. Például [\[DENOTATIONALSEMANTICS\]](#) 4.1.1. fejezet, vagy akár [\[SICP\]](#) Exercise 4.15.

```
1 Program T100
2 {
3
4     boolean Lefagy(Program P)
5     {
6         if(P has infinite loop)
7             return true;
8         else
9             return false;
10    }
11
12    main(Input Q)
13    {
14        Lefagy(Q)
15    }
16 }
```

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:

```
1 Program T1000
2 {
3
4     boolean Lefagy(Program P)
5     {
6         if(P has infinite loop)
7             return true;
8         else
9             return false;
10    }
```

```

11
12 boolean Lefagy2(Program P)
13 {
14     if(Lefagy(P))
15         return true;
16     else
17         for(;;);
18 }
19
20 main(Input Q)
21 {
22     Lefagy2(Q)
23 }
24
25 }

```

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.

Persze ha már lefagyott, nem fagyott témánál vagyunk, akkor érdemes kicsit beszélni a kiértékelésről. Alább látható egy egyszerű kifejezés. Mi lesz vajon a vége?

$$(\lambda x.y)((\lambda z.zz)(\lambda z.zz)) \quad (2.1)$$

Redukáljuk!

$$(\lambda x.y)((\lambda z.zz)(\lambda z.zz)) \quad (2.2)$$

Humph, ugyanoda jutottunk, kezdjük előlről!

$$(\lambda x.y)((\lambda z.zz)(\lambda z.zz)) \quad (2.3)$$

Redukáljuk az x-t kötő baloldali lambdát a jobb oldallal a jobb oldal redukciója nélkül.

$$y \quad (2.4)$$

Azaz addig késleltettük a kiértékelést, amíg lehetett, és a végén kiderült, hogy abszolút nem is volt szükséges! Ez a módszer zseniális! Innentől mindent késleltetni fogunk!

Nézzünk egy példát a [SICP]-ből! Alábbi definíciók alapján ki fogjuk értékelni a  $(f\ 5)$ -t!

```

(define (square x) (* x x))
(define (sum-of-squares x y)
  (+ (square x) (square y)))
(define (f a)
  (sum-of-squares (+ a 1) (* a 2)))

```

Annyira jó ez a késleltetés, hogy alkalmazzuk a  $(f\ 5)$  kiértékelésénél!

```
(f 5)
(sum-of-squares (+ 5 1) (* 5 2))
(+ (square (+ 5 1)) (square (* 5 2)) )
(+ (* (+ 5 1) (+ 5 1)) (square (* 5 2)))
(+ (* (+ 5 1) (+ 5 1)) (* (* 5 2) (* 5 2)))
(+ (* 6 (+ 5 1)) (* (* 5 2) (* 5 2)))
(+ (* 6 6) (* (* 5 2) (* 5 2)))
(+ (* 6 6) (* 10 (* 5 2)))
(+ (* 6 6) (* 10 10))
(+ 36 (* 10 10))
(+ 36 100)
136
```

Mint ahogy azt látjuk  $(+ 5 1)$  és  $(* 5 2)$  kétszer kerül redukcióra. Hát így már nem is olyan jó...

Hogy fut le vajon egy másik út?

```
(f 5)
(sum-of-squares (+ 5 1) (* 5 2))
(sum-of-squares 6 (* 5 2))
(sum-of-squares 6 10)
(+ (square 6) (square 10))
(+ (* 6 6) (* 10 10))
(+ 36 (* 10 10))
(+ 36 100)
136
```

Kevesebb számítást végzünk mint a késleltetős előző esetben. Azaz látjuk, hogy a redukciós stratégiának a teljesítményre is van hatása.

## 2.3. Változók értékének felcserélése

Írjunk egy olyan programot, amely felcseréli két változó értékét, bármiféle logikai utasítás vagy kifejezés használata nélkül, és ez ne bug hanem feature legyen!

Megoldás videó: No Link

Megoldás forrása:

- [Swap XOR](#)
- [Swap Subtract](#)
- [Swap Multiplication](#)
- [Swap Ptr](#)

Több megoldási lehetőség is van, ezeket fogjuk a következő bekezdésekben bemutatni.

Nézzünk valami szorzásos osztásosot!

```
1 #include <stdio.h>
2
3 int main(){
4     int a = 6;
5     int b = 7;
6     printf("Let a = %d, b = %d\n",a,b);
7     a = a * b;
8     printf("(1) a = a * b = %d\n",a);
9     b = a / b;
10    printf("(2) b = a / b = %d\n",b);
11    a = a / b;
12    printf("(3) a = a / b = %d\n",a);
13    printf("Result after swap by multiplication and division is a = %d, b = ↵
        %d\n",a,b);
14    return 0;
15 }
```

Kövessük nyomon mi történik!

(a, 6) (b, 7)  
a = a \* b => (a, 42) (b, 7)  
b = a \ b => (a, 42) (b, 7)  
a = a \ b => (a, 6) (b, 7)

Hát ez működik, de 0-val nem osztunk, plusz a szorzás osztás nehéz. Nézzünk mást!

Nézzünk valami összeadás kivonásosot!

```
1 #include <stdio.h>
2
3 int main(){
4     int a = 6;
5     int b = 7;
6     printf("Let a = %d, b = %d\n",a,b);
7     a = a + b;
8     printf("(1) a = a + b = %d\n",a);
9     b = a - b;
10    printf("(2) b = a - b = %d\n",b);
11    a = a - b;
12    printf("(3) a = a - b = %d\n",a);
13    printf("Result after swap by addition and subtract is a = %d, b = %d\n" ↵
        ,a,b);
14    return 0;
15 }
```

Kövessük nyomon mi történik!

(a, 6) (b, 7)



```
a = a + b => (a, 13) (b, 7)
b = a - b => (a, 13) (b, 6)
a = a - b => (a, 7) (b, 6)
```

Ez is jó, és ráadásul csak összeadással és kivonással terheljük a CPU-t!

De esetleg van valami más is? Nos nézzük a xor műveletet!

```
1 #include <stdio.h>
2
3 int main(){
4     int a = 6;
5     int b = 7;
6     printf("Let a = %d, b = %d\n", a, b);
7     a = a ^ b;
8     printf("(1) a = a ^ b = %d\n", a);
9     b = a ^ b;
10    printf("(2) b = a ^ b = %d\n", b);
11    a = a ^ b;
12    printf("(3) a = a ^ b = %d\n", a);
13    printf("Result after swap by xor is a = %d, b = %d\n", a, b);
14    return 0;
15 }
```

Kövessük végig!

```
(a, 110) (b, 111)
a = a XOR b => (a, 001) (b, 111)
b = a XOR b => (a, 001) (b, 110)
a = a XOR b => (a, 111) (b, 110)
```

Egy nagyon furcsa dolgot láthatunk. Az összes előző esetenél több operátort kellett használni (\*) és (/) illetve (+) és (-). Most azonban egy olyan esetet látunk, amikor egyetlen operátorral megtudtuk oldani (kommutatív, asszociatív, identitás elem létezik,  $f(a,a)=\text{Identitás elem}$ ).

Alábbi utolsó példa csak egy picit behozza a pointer-eket a képbe. Ezekkel ha még nem értjük mi van, ne aggódjunk. Minden pointert érdemes egy címként felfogni. Azaz ha létezik egy int változónk foo néven akkor foo-nak van ugye egy értéke, illetve egy címe. A cím az amit a ptr megtestesít (na jó nem, de most így elég...). Természetesen funckiókra is mutathatunk. Sőt, ha már valaha láttunk C kódot, akkor valószínűleg belefutottunk már callback-ekbe stb. Aka akár funckiókat és bepasszolhatunk funckiókba, stb.

```
1 #include <stdio.h>
2
3 void swap1(int *pa, int* pb ){
4     int t = *pa;
5     *pa = *pb;
6     *pb = t;
7 }
8
9 void swap2(int *pa, int* pb ){
10    *pa = *pa + *pb;
```

```
11     *pb = *pa - *pb;
12     *pa = *pa-*pb;
13 }
14
15 int main()
16 {
17     int a = 6;
18     int b = 7;
19     printf("Let a = %d, b = %d\n", a, b);
20     swap1(&a, &b);
21     printf("Swap1 %d, %d\n", a, b);
22     swap2(&a, &b);
23     printf("Swap2 %d, %d\n", a, b);
24     return 0;
25 }
```

## 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! Nem írhatasz hozzá DLC-t, és ne legyen benne lootbox, mert nem ez EA vagyunk.

Megoldás videó:

Megoldás forrása:

- [Helper header](#)
- [Ball](#)
- [Ball no if](#)

Ennek a feladatnak az volt a lényege, hogy elágazás nélkül kicsikarjunk a gépből valami féle döntés alapú útválasztást. Először nézzük az alapot.

```
1 #include "pubghypetrain.h"
2
3 void collide(int posX, int posY, int &velX, int &velY, int xmax, int ymax){
4     bool outX = (posX <= 0 || posX >= xmax-1);
5     velX += -2 * velX * outX;
6     bool outY = (posY <= 0 || posY >= ymax-1);
7     velY += -2 * velY * outY;
8 };
9
10 void moveBall(int &posX, int &posY, int velX, int velY){
11     posX += velX;
12     posY += velY;
13 };
14
```

```

15 void draw(int posX, int posY, int xmax, int ymax){
16     ClearScreen();
17     for(int y = 0; y < ymax; y++){
18         for(int x = 0; x < xmax; x++){
19             if( (posX == x) && (posY == y) ){ std::cout << 'O'; }else{ std::cout << ' '; }
20         };
21         std::cout << std::endl;
22     };
23 };
24
25 int main(){
26     int T_SLEEP_MILLIS = 50;
27     int GLOBAL_H = 10;
28     int GLOBAL_W = 10;
29     int posX = 5;
30     int posY = 2;
31     int velX = 2;
32     int velY = -1;
33     for(int i = 0; i < 50; i++){
34         moveBall(posX, posY, velX, velY);
35         collide(posX, posY, velX, velY, GLOBAL_H, GLOBAL_W);
36         draw(posX, posY, GLOBAL_H, GLOBAL_W);
37         sleep(T_SLEEP_MILLIS);
38     };
39     return 0;
40 };

```

Nyilvánvalóan valahogyan trükközni kell, hisz az `if`-et ha kihagynánk, akkor nem tudnánk például dönteni, hogy mely karaktert küldjük `std::out`-ra. Nézzük egy lehetséges megoldást.

```
1 #include "pubghypetrain.h"
2 #include <iostream>
3
4 void collide(int posX, int posY, int &velX, int &velY, int xmax, int ymax);
5
6 void moveBall(int &posX, int &posY, int velX, int velY);
7
8 void draw(int posX, int posY, int xmax, int ymax, const char* syms);
9
10 int main()
11 {
12     const char SYMBOLS[2] = {'.','O'};
13     int T_SLEEP_MILLIS = 50;
14     int GLOBAL_H = 10;
15     int GLOBAL_W = 10;
16     int posX = 5;
17     int posY = 2;
18     int velX = 2;
19     int velY = -1;
20     for(int i = 0; i < 50; i++){
```

```

21     moveBall(posX, posY, velX, velY);
22     collide(posX, posY, velX, velY, GLOBAL_H, GLOBAL_W);
23     draw(posX, posY, GLOBAL_H, GLOBAL_W, SYMBOLS);
24     sleep(T_SLEEP_MILLIS);
25 };
26     return 0;
27 };
28
29
30 void moveBall(int &posX, int &posY, int velX, int velY){
31     posX += velX;
32     posY += velY;
33 };
34
35 void collide(int posX, int posY, int &velX, int &velY, int xmax, int ymax){
36     velX += -2 * velX * (posX <= 0 || posX >= xmax-1);
37     velY += -2 * velY * (posY <= 0 || posY >= ymax-1);
38 };
39
40 void draw(int posX, int posY, int xmax, int ymax, const char* syms){
41     ClearScreen();
42     for(int y = 0; y < ymax; y++){
43         for(int x = 0; x < xmax; x++){
44             std::cout << syms[(posX == x) && (posY == y)];
45         };
46         std::cout << std::endl;
47     };
48 };

```

Simán csak implicit konverzió történik, mind `collide`, mind `draw` esetén.

Ha már itt tartunk, akkor nézzünk más jellegű `if` encode-olást is! (használhatjuk például a [\[CHISOMORPH\]](#)-t)

```

true = λx.λy.x
false = λx.λy.y
if(B,P,Q) = B P Q

```

Az `if true then P else Q` redukciója

```

if true then P else Q
= true P Q
= λx.λy.x P Q
= λy.P Q
= P

```

Az `if false then P else Q` redukciója

```

if false then P else Q
= false P Q
= λx.λy.y P Q

```

```
= λy.y Q
= Q
```

Vezessünk be számokat!

```
if false then P else Q
0 = λf.λx.x
1 = λf.λx.f x
2 = λf.λx.f (f x)
...
```

Illetve egy isZero predikátumot!

```
IsZero = λn.n (λx.false) true
```

Nézzük meg pár esetre a predikátumunk működését!

```
IsZero 0
= (λn.n (λx.false) true) (λf.λx.x)
= ( (λf.λx.x) (λx.false) true)
= ( λx.x true)
= true
```

```
IsZero 1
= (λn.n (λx.false) true) (λf.λx.f x)
= ( (λf.λx.f x) (λx.false) true)
= ( (λx.(λx.false) x) true)
= (λx.false) true
= false
```

Mostmár mondhatunk olyat, hogy osztásnál ne legyen nulla a nevezőben, azaz `if IsZero b then 0 else (div a b)`. A lényeg az, hogy dobjon vissza egy `(div valami valami)-t` ha el lehet végezni, különben pedig egy 0-t. Például 4-el és 0-val hívjuk.

```
( (λx.λy.(IsZero y) 0 (div x y)) 4 0
= ( (λy.(IsZero y) 0 (div 4 y)) 0
= ((IsZero 0) 0 (div 4 0))
= (true 0 (div 4 0))
= ((λx.λy.x) 0 (div 4 0))
= ((λy.0) (div 4 0))
= 0
```

Most nézzük meg 4 és 2-vel.

```
( (λx.λy.(IsZero y) 0 (div x y)) 4 2
( (λy.IsZero y) 0 (div 4 y)) 2
((IsZero 2) 0 (div 4 2))
(false 2 (div 4 2))
((λx.λy.y) 0 (div 4 2))
((λy.y) (div 4 2))
(div 4 2)
```

## 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/9KnMqrkj_kU), <https://youtu.be/KRZlt1ZJ3qk>, .

Megoldás forrása:

- [Wordsize](#)
- [Bogomips](#)

Először is nézzük a biteket! A számokat kettes számrendszerben tároljuk. A shift az értékek "tologatása". De pontosan mi az a shift?

Amikor left shiftelünk, akkor mi is történik? Nézzük unsigned char-ra! „For unsigned lhs, the value of LHS << RHS is the value of LHS \* 2<sup>RHS</sup>, reduced modulo maximum value of the return type plus 1 (that is, bitwise left shift is performed and the bits that get shifted out of the destination type are discarded).” Alább látható hogy miről van szó.

1	<< 1 =>	(1*2 <sup>1</sup> ) % (255+1)	= 2	(00000010)
2	<< 1 =>	(2*2 <sup>1</sup> ) % (255+1)	= 4	(00000100)
4	<< 1 =>	(4*2 <sup>1</sup> ) % (255+1)	= 8	(00001000)
8	<< 1 =>	(8*2 <sup>1</sup> ) % (255+1)	= 16	(00010000)
16	<< 1 =>	(16*2 <sup>1</sup> ) % (255+1)	= 32	(00100000)
32	<< 1 =>	(32*2 <sup>1</sup> ) % (255+1)	= 64	(01000000)
64	<< 1 =>	(64*2 <sup>1</sup> ) % (255+1)	= 128	(10000000)
128	<< 1 =>	(128*2 <sup>1</sup> ) % (255+1)	= 0	(00000000)

Alábbi kóddal nézzük meg:

```
1 #include <iostream>
2
3 int main()
4 {
5     unsigned char a = 1;
6     for(int i = 0; i<8; i++)
7     {
8         a=a<<1;
9         std::cout<< ((int)a) << std::endl;
10    }
11    return 0;
12 }
```

```
$ ./shiftr_uchar.exe
127
63
31
15
7
3
1
0
```

2.5. ábra. shiftr\_uchar

Mi történik a right shift esetben? „For unsigned lhs and for signed lhs with nonnegative values, the value of  $LHS \gg RHS$  is the integer part of  $LHS / 2^{RHS}$ .”

```
255 >> 1 => intp(255 div 2^1) = 127 (01111111)
127 >> 1 => intp(127 div 2^1) = 63  (00111111)
63  >> 1 => intp(63 div 2^1)  = 31  (00011111)
31  >> 1 => intp(31 div 2^1)  = 15  (00001111)
15  >> 1 => intp(15 div 2^1)  = 7   (00000111)
7   >> 1 => intp(7 div 2^1)   = 3   (00000011)
3   >> 1 => intp(3 div 2^1)   = 1   (00000001)
1   >> 1 => intp(1 div 2^1)   = 0   (00000000)
```

Alábbi kóddal nézzük meg:

```
1 #include <iostream>
2
3 int main()
4 {
5     unsigned char a = 255;
6     for(int i = 0; i<8; i++)
7     {
8         a=a>>1;
9         std::cout<< ((int)a) << std::endl;
10    }
11    return 0;
12 }
```

```
$ ./shiftr_uchar.exe
127
63
31
15
7
3
1
0
```

2.6. ábra. shiftr\_uchar

Alábbi kód az előzőeket hajtja végre csak int-et használunk.

```
1  #include <iostream>
2
3  void shift_l()
4  {
5      unsigned int a = 1;
6      unsigned int i = 0;
7      while(a!=0){
8          a = a<<1;
9          i++;
10     }
11     std::cout<<"Shift_L count was " << i << std::endl;
12     std::cout<<"Size " << ((i)/8) << std::endl;
13     std::cout<<"Sanity Check " << sizeof(int) << std::endl;
14 }
15
16 void shift_r()
17 {
18     unsigned int a = 0;
19     a = ~a;
20     int i = 0;
21     while(a!=0){
22         a = a>>1;
23         i++;
24     }
25     std::cout<<"Shift_R count was " << i << std::endl;
26     std::cout<<"Size " << ((i)/8) << std::endl;
27     std::cout<<"Sanity Check " << sizeof(int) << std::endl;
28 }
29
30 int main()
31 {
32     shift_l();
33     shift_r();
34     return 0;
35 }
```

Előjeles esetben figyelembe kell venni, hogy például a felső bit előjelet is jelenthet akár. Alábbi ábra szép illusztrációt ad arra, mire is kell gondolni (2s complement).



Positive numbers		Negative numbers	
0	0000		
1	0001	1111	-1
2	0010	1110	-2
3	0011	1101	-3
4	0100	1100	-4
5	0101	1011	-5
6	0110	1010	-6
7	0111	1001	-7
		1000	-8

2.7. ábra. 2s complement

Most áttérünk a bogomips-re. Itt egy új dologgal találkozunk `<<=` de ez igazából csak annyit tesz, hogy a shift rhs-t egyből az lhs oldalon lévő változóba vissza is tároljuk. A delay egy for loop ami elszámol loops-ig, azaz addig növeli (`++`) unáris operátorral `i` értékét amíg a `i < loops` conditional igaz. A while ciklus fejben pedig már a jól ismert left shift zajlik (`loops_per_sec` 1-ről indul). De hogy biztosak legyünk nézzük az alábbi programot (bár nagyon ismerős lesz...)

```

1  #include <iostream>
2
3  int main()
4  {
5      unsigned long long a = 1;
6      int i = 0;
7      while(a!=0)
8      {
9          a<<=1;
10         std::cout<< a << std::endl;
11         i++;
12     }
13     std::cout<< i << std::endl;
14     return 0;
15 }
```

```

1152921504606846976
2305843009213693952
4611686018427387904
9223372036854775808
0
64
```

2.8. ábra. ullshift

A bogomips while body egyébként annyiról szól, hogy lekérjük hány tick telt el idáig, majd nyomunk

n mennyiség lépést a for ciklusban, visszajövünk és megint lekérdezzük az eltelt tick-ek számát. Ha a tickek száma nem kevesebb mint amennyit a CLOCKS\_PER\_SEC konstanssal állít magáról a gép, akkor elkezdjük a kiszállást. A loops\_per\_sec pedig a következő módon jön ki:

```
loop_per_masodperc = ( loopok_szama / tick_szam ) * CLOCKS_PER_SEC;  
[1/s] = ( [1] / [1] ) * [1/s]
```

Ezután következik zsonglőrködés a számokkal. Ez a rész a bogus a bogomips-ben. Ha sok időnk van akkor meg is mérhetjük a saját gépünkét. Alább egy random példa.

```
$ ./bogomips.exe  
Calibrating delay loop..ok - 225.38 BogoMIPS
```

2.9. ábra. bogomips kimeneten

## 2.6. Helló, Google!

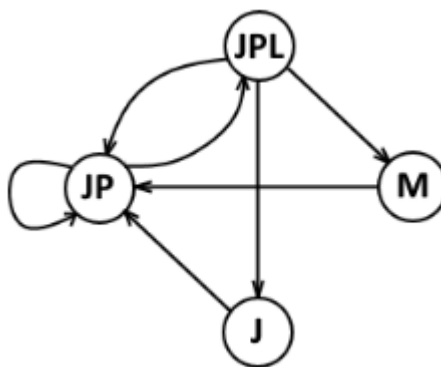
Írj olyan C programot, amely egy több milliárd(haha ne ne...elég lesz kevesebb srsly) honlapból álló hálózatra kiszámolja az N lap Page-Rank értékét, és [nem lesz belőle baj egy szenátusi kihallgatáson!](#)

Megoldás videó:

Megoldás forrása:

- [Pagerank Példa](#)
- [Pagerank C](#)
- [STL Iterator](#)
- [Purely STL Vector](#)

Pagerank a Google által használt adóelk...keresési algoritmus magja. A feladat arról szólt, hogy egy négy honlapból álló kapcsolati hálón kellene vele elemezni. A pageranktól azt várjuk, hogy az összes honlapnak mondjuk adjon egy 0-1-ig terjedő értéket (normalizált, azaz az összes lap rankjének összege 1 for sanity check). Van több implementáció is, de mi egy konkrétan kitenyésztett példa progival fogunk dolgozni. Ez a pagerank vektor értékeit fő iterációs lépésenként kiírja egy fájlba. A pagerankkel a példában egy irányított multigráfon fogunk számolni. (irányított mert számít hogy honnan-hova megy az él, multigráf pedig a több él egy pontból plusz hurkok is lehetnek)



2.10. ábra. pagerank kapcsolati irányított multigráf

Ez ugyan emberként érthető, de próbáljuk valahogy szervezettebb formába hozni. Csináljunk egy kvadratus mátrixot, és töltsük ki a következő szabály szerint: Először mindenhova írjunk be nullát majd Ha A-ból B-be megy él, akkor A oszlop B sorába írjunk 1-et Először nézzük csak meg J-re(J-ből egy él megy ki JP felé...)

	J	JP	JPL	M
J	0	0	0	0
JP	1	0	0	0
JPL	0	0	0	0
M	0	0	0	0

Most pedig csináljuk meg JP-re(magába és JPLbe)

	J	JP	JPL	M
J	0	0	0	0
JP	1	1	0	0
JPL	0	1	0	0
M	0	0	0	0

Jöhet JPL (mindenkibe csak magába nem)

	J	JP	JPL	M
J	0	0	1	0
JP	1	1	0	0
JPL	0	1	1	0
M	0	0	1	0

Jöhet M (csak JP-be)

	J	JP	JPL	M
J	0	0	1	0
JP	1	1	0	1
JPL	0	1	1	0
M	0	0	1	0

Most normalizáljuk, azaz adjuk össze az egy oszlopban lévő számokat, nevezzük ezt *szummának*. Ha meg van, akkor utána ugyanezen az oszlopon menjünk végig és mindenkit osszunk el *szum*-mal, ha *szum* nem nulla.

	J	JP	JPL	M
J	0	0	1/3	0
JP	1	1/2	1/3	1
JPL	0	1/2	0	0
M	0	0	1/3	0

Vessük össze az előadás diával.

	J	JP	JPL	M
J	0	0	1/3	0
JP	1	1/2	1/3	1
JPL	0	1/2	0	0
M	0	0	1/3	0

2.11. ábra. pagerank link mátrix sanity check

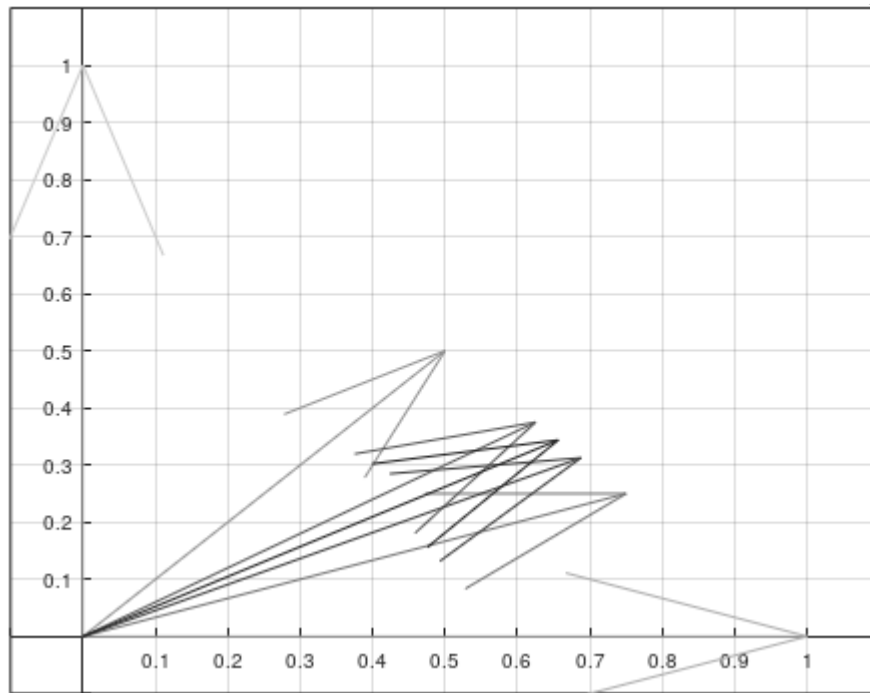
Most hogy végre van link mátrixunk elkezdhetünk számolni. De mi is a lényeg, mit fogunk számolni? Gondoljunk vissza a [sajátérték sajátvektorra](#), és [próbálgassuk](#) is ha tetszik! Magyarul amikor Tanár Úr azt mondja Tehát ha  $h$  jelöli a PR vektort, akkor  $h = Lh$ . Linalg kedvelőknek: a PageRank vektor az  $L$  linkmátrix 1 sajátértékhez tartozó sajátvektora. akkor gondoljunk a következőre

$$\begin{array}{c}
 \begin{array}{c|c|c|c|c}
 & \text{PR}[1] & & & \\
 & \text{PR}[2] & & & \\
 & \text{PR}[3] & & & \\
 & \text{PR}[4] & & & 
 \end{array} \\
 \\
 \begin{array}{c|c|c|c|c|c|c|c|c}
 \begin{array}{c}
 0 \quad 0 \quad 1/3 \quad 0 \\
 1 \quad 1/2 \quad 1/3 \quad 1 \\
 0 \quad 1/2 \quad 0 \quad 0 \\
 0 \quad 0 \quad 1/3 \quad 0
 \end{array}
 & \begin{array}{c}
 \\ \\ \\ 
 \end{array}
 & \begin{array}{c}
 ? \\ ? \\ ? \\ ?
 \end{array}
 & = & \begin{array}{c}
 \text{PR}[1] \\ \text{PR}[2] \\ \text{PR}[3] \\ \text{PR}[4]
 \end{array}
 \end{array}
 \end{array}$$

De milyen vektorról is beszélünk? Mi a végcél ezzel az egész saját vektor dologgal? Vizualizáljuk!

Jelen példában 4 node van szóval PR 4 elemű, ezt nehéz lenne ábrázolnom. Viszont gondoljunk egy két node-ból álló esetre! Ezen esetben PR vektor 2 elemű lesz, szóval már plottolhatjuk 2dbbe. Alábbi ábrán simán fogtuk magunkat és a PR vektor 1. számát  $x$ -nek 2. számát  $y$ -nak vettük csináltunk bele egy nyilat

az origóból. Színkódoltuk is: A legelső iter utáni PR vektor halvány szürke, az utolsó pedig fekete, közte pedig graduálisan változtattuk a feketeséget (hsv mert lusta voltam, de ne menjünk bele).

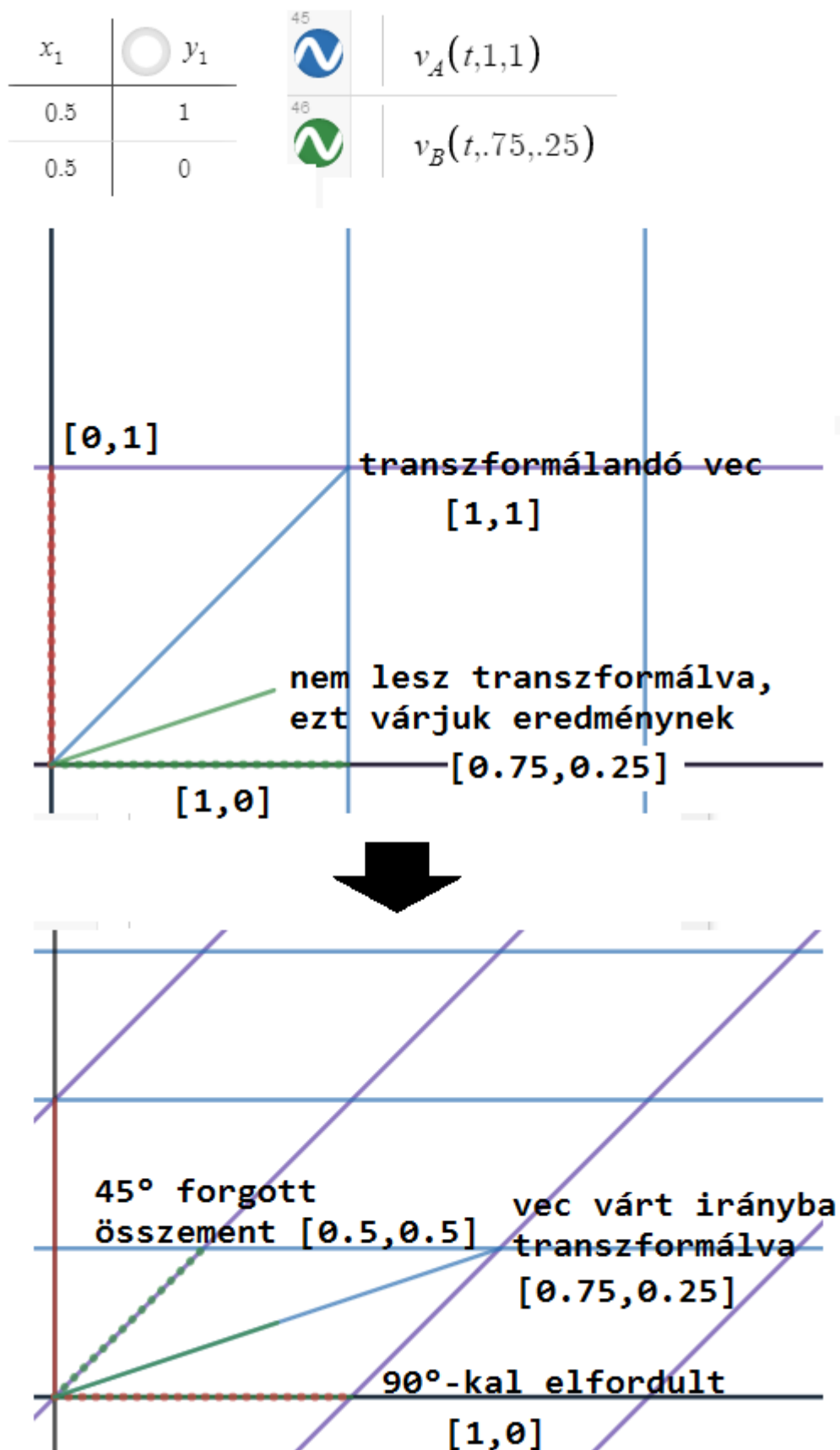
$$\begin{aligned}x &= [0 \quad 1 \quad 0.5 \quad 0.75 \quad 0.625 \quad 0.6875 \quad 0.65625 \quad ] \\y &= [1 \quad 0 \quad 0.5 \quad 0.25 \quad 0.375 \quad 0.3125 \quad 0.34375 \quad ]\end{aligned}$$


2.12. ábra. pagerank vektor

Fenti ábrán látható, hogy ahogy beindul az iteráció vadul  $[0,1]$ -ből  $[1,0]$ -ba vált, majd  $[0.5,0.5]$  és szépen lassan kezd beállni  $[0.66 \ 0.33]$ -ba!

Magyarul a nem pontos PR vektor közelítésünk egyre kisebb cikázással kezd beállni abba az irányba ami egyébként az "igazi" sajátvektor irány. Ez alapján világos, hogy az egész számításnak az a célja hogy megtaláljuk azt a vektort, amire ha alkalmazzuk a Link mátrixot transzformációként, akkor iránya már nem fog változni. (Ha valakit érdekel akkor a 2d-s eset az oppenoffice [fájl](#)-ban a twod munkalapon van, de semmi különös.)

Ha valaki mégsem értené, nézzünk egy konkrét példát: Hogyan lett pl  $[0.5,0.5]$  irányú vektorból  $[0.75,0.25]$ ? Alábbi ábrán felveszünk egy  $[0.75,0.25]$  vektort(zöld folytonos vonal), és ezt változatlanul fogjuk hagyni. Ezután veszünk egy  $[1,1]$  vektort(kék folytonos vonal). Azért  $[1,1]$  mert csak az irány a fontos, és azt akartam hogy nyúljon túl a zöldön trafó után. Ezekután transzformáljuk a linkmátrix-szal a kéket. Azt várjuk hogy a transzformáció után a zöld folytonos és kék folytonos egyirányú legyen.



2.13. ábra. pagerank trf

Vissza matekra! Mátrixok vektorok stb. szorzást már tanultunk szóval, oldjuk is meg!

$$\begin{array}{cccc|cccc}
 & & & & & & \text{PR}[1] & \\
 & & & & & & \text{PR}[2] & \\
 & & & & & & \text{PR}[3] & \\
 & & & & & & \text{PR}[4] & \\
 \hline
 0 & 0 & 1/3 & 0 & & & \text{PR}[3]/3 & \\
 1 & 1/2 & 1/3 & 1 & & & \text{PR}[1] + \text{PR}[2]/2 + \text{PR}[3]/3 + \text{PR}[4] & \\
 0 & 1/2 & 0 & 0 & & & \text{PR}[2]/2 & \\
 0 & 0 & 1/3 & 0 & & & \text{PR}[3]/3 & 
 \end{array} = \begin{array}{cccc}
 \text{PR}[1] \\
 \text{PR}[2] \\
 \text{PR}[3] \\
 \text{PR}[4]
 \end{array}$$

Szét is robbanthatjuk akár egyenletekre...

$$\begin{array}{lcl}
 \text{PR}[3]/3 & = & \text{PR}[1] \\
 \text{PR}[1] + \text{PR}[2]/2 + \text{PR}[4] & = & \text{PR}[2] \\
 \text{PR}[2]/2 + \text{PR}[3]/3 & = & \text{PR}[3] \\
 \text{PR}[3]/3 & = & \text{PR}[4]
 \end{array}$$

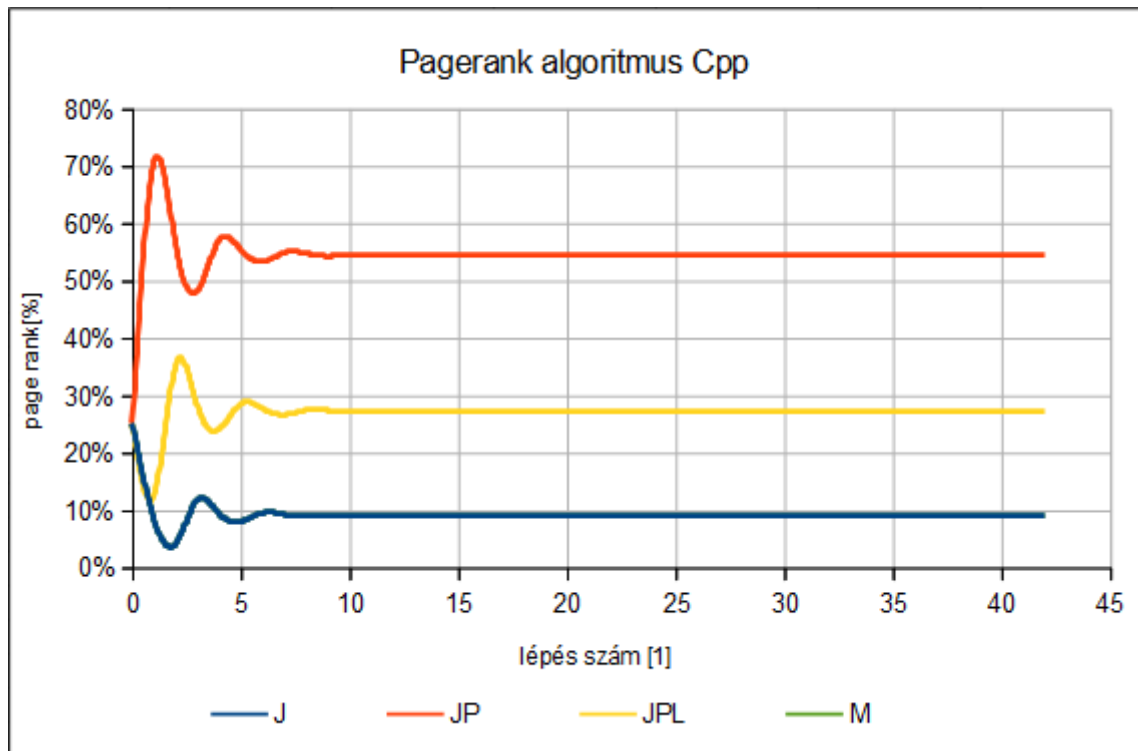
Most pedig jön a kérdés, hogy hogyan oldjuk meg?  $\text{PR}[2]$  önmagából számolja önmagát!

A válasz nem túlzottan bonyolult...tároljuk el az előző lépés értékeit minden ciklusban és használjuk azokat. (Ez lesz a megoldásban  $\text{PRv}$ )

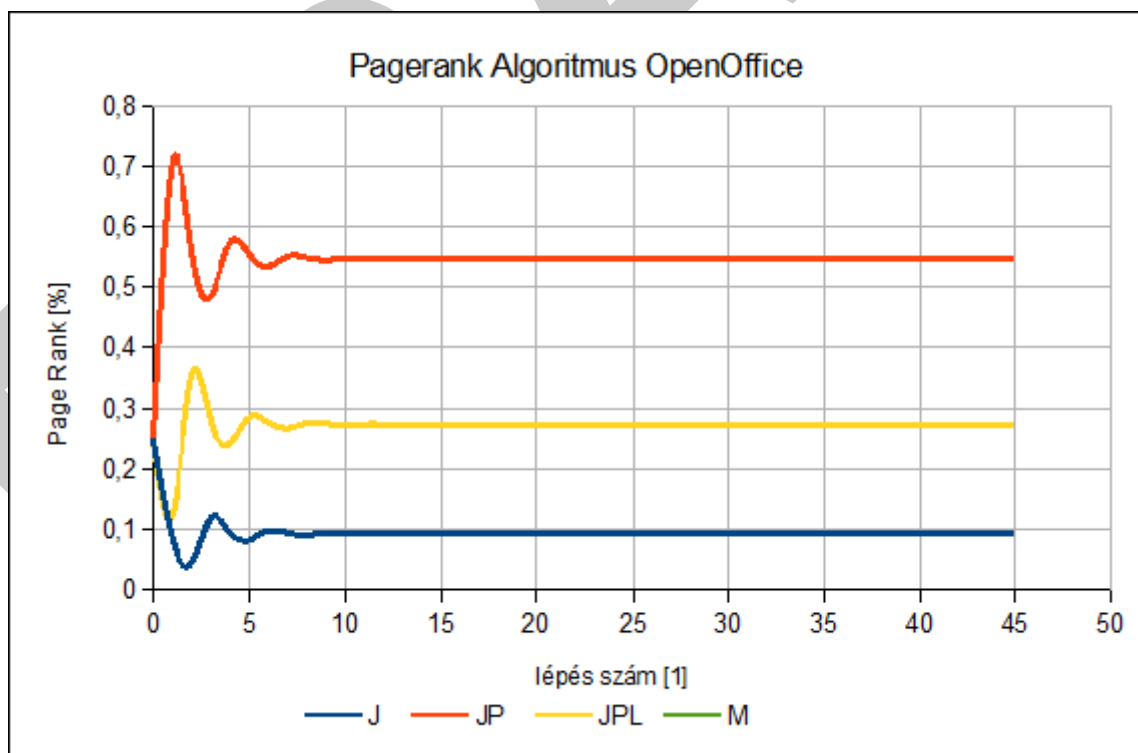
$$\begin{array}{lcl}
 \text{PRv}[3]/3 & = & \text{PR}[1] \\
 \text{PRv}[1] + \text{PR}[2]/2 + \text{PR}[4] & = & \text{PR}[2] \\
 \text{PRv}[2]/2 + \text{PR}[3]/3 & = & \text{PR}[3] \\
 \text{PRv}[3]/3 & = & \text{PR}[4]
 \end{array}$$

Egy kérdés még maradt: Mennyi legyen  $\text{PRv}$  iniciális értéke? A válasz annyi, hogy mindegyik érték 1 aztán benormáljuk, hogy összegük 1 legyen (Mat viz).

Ha midezt megértettük, akkor már könnyű lesz megérteni az alábbi kódomat! A refekkel, constokkal, inicializer listekkel meg a többi szeméttel ne törődjünk még nem kell őket érteni, csak a pagerank-re fókuszáljunk. Ha esetleg nem lenne érthető a kód, akkor beraktam a repóba egy openoffice calc (olyan mint az excel csak kevesebb benne a fluff) doksit, ami ugyanezt a pagerank számítást csinálja... A fejezet végén beillesztettem a C++ kódot, viszont most következzen két diagram. Az első a C++ algoritmus  $\text{PR}$  vektorának értékeit mutatja iterációnként, a másik pedig az [open office](#)-ost, ha valaki valami miatt szereti az irodai munkát.



2.14. ábra. pagerank cpp konvergál



2.15. ábra. pagerank openoffice konvergál



Kicsit azért vigyázzunk...ugyanis van két kis probléma. Tanár Úr azért mondta hogy rázzuk a vizet az internetben, mert arra gondolt, hogy ez az iteratív megoldás olyan mintha vödörök között öntögetnénk vizet, vagy mondjuk egy fémlemez hőmérsékletét akarnánk úgy kiszámolni, hogy az egyik szélén tudjuk a hőmérsékletet a többit pedig úgy számoljuk, hogy a négy (felső-alsó-bal-jobb egy kockás lapon mondjuk) szomszédját átlagoljuk. Nos első esetben az a probléma, hogy az egyik vödörbe folyik víz be, vissza is folyik magába, de kifelé nem. A példa fájlban egy `get_dangling` funkció csinál olyan mátrix-ot amiben van egy nyelő jellegű node.

```
{ 1, 0, 1, 0, 1, 0 },  
{ 1, 1, 0, 1, 0, 0 },  
{ 0, 1, 1, 0, 1, 0 },  
{ 1, 0, 1, 1, 0, 0 },  
{ 0, 1, 0, 1, 1, 0 },  
{ 1, 0, 1, 0, 1, 0 }
```

Most pedig következzen az igazi vizes locspocs. Vegyük a következő kapcsolati gráfot:

	A	A
A	0	0
B	1	1

A locspocs-t indítsuk a következő PRv-vel:

```
PRv  0  
PRv  1
```

Érezhető hogy ennek mi lesz a vége, ha valaki elképesztően kíváncsi az open office fájlban-ban látható egy Infinite nevű munkalap.

```
1  #include <iostream>  
2  #include <vector>  
3  #include <cmath>  
4  #include <fstream>  
5  #include <iomanip>  
6  
7  
8  void kiir(std::ofstream& os, const std::vector<double> &v) {  
9      int sz = v.size();  
10     if(sz<1) return;  
11     os << v[0];  
12     for(int i=1; i<sz; i++)  
13         os << ";" << v[i];  
14     os << std::endl;  
15 }  
16  
17 double tavolsag(const std::vector<double> &PR, const std::vector<double> & ←  
18     PRv) {  
19     int i;  
20     double osszeg = 0;  
21     for(i=0; i<PR.size(); ++i) osszeg += (PRv[i]-PR[i])*(PRv[i]-PR[i]);  
22     return sqrt(osszeg);
```

```
22 }
23
24 std::vector<std::vector<double>> get_batfai_graph()
25 {
26     return std::move(std::vector<std::vector<double>>{
27         /*J*/ { 0, 0, 1, 0},
28         /*JP*/ { 1, 1, 1, 1},
29         /*JPL*/ { 0, 1, 0, 0},
30         /*M*/ { 0, 0, 1, 0}
31     });
32 }
33
34 std::vector<std::vector<double>> get_dangling()
35 {
36     return std::move(std::vector<std::vector<double>>{
37         { 1,0,1,0,1,0},
38         { 1,1,0,1,0,0},
39         { 0,1,1,0,1,0},
40         { 1,0,1,1,0,0},
41         { 0,1,0,1,1,0},
42         { 1,0,1,0,1,1}
43     });
44 }
45
46 std::vector<std::vector<double>> get_symm()
47 {
48     return std::move(std::vector<std::vector<double>>{
49         { 0,1 },
50         { 1,0 }
51     });
52 }
53
54 void normalize(std::vector<std::vector<double>>& quadmx)
55 {
56     int size = quadmx.size();
57     for(int j=0; j<size; j++){
58         double sum = 0;
59         for(int i=0; i<size; i++){
60             sum+=quadmx[i][j];
61         }
62         if(sum!=0){
63             for(int i=0; i<size; i++){
64                 quadmx[i][j]/=sum;
65             }
66         }
67     }
68 }
69
70 int main(void){
71     // ha probalgatod es ne adj isten valami miatt ne konvergalna
```

```
72 // akkor ennyi iter után auto kilep a fo loopbol
73 int SAFE_LIMIT = 100;
74 std::ofstream outf;
75 outf.open ("pagerank.csv");
76 // at this point L is not normalized
77 std::vector<std::vector<double>>L{std::move(get_dangling())};
78 normalize(L);
79 int size = L.size();
80 std::vector<double> PR;
81 for(int i = 0; i<size; i++)PR.push_back(0.0);
82 std::vector<double> PRv;
83 for(int i = 0; i<size; i++)PRv.push_back(1.0/size);
84 for(int c;c<SAFE_LIMIT;c++){
85     for(int i = 0; i < size; ++i){
86         PR[i] = 0.0;
87         for(int j = 0; j < size; ++j) PR[i] += L[i][j] * PRv[j];
88     }
89     kiir(outf,PR);
90     if(tavolsag(PR,PRv)<0.0000000001) break;
91     PRv = PR;
92 }
93 outf.close();
94 return 0;
95 }
```

## 2.7. A Monty Hall probléma

Írj szimulációt a Monty Hall problémára!

Megoldás videó: [No link](#)

Megoldás forrása: [No link](#)

Gondolkodjunk úgy hogy két stratégia van: "nem választunk újra" és "újra választunk". Alábbi kódban "nem választunk újra" esetben N lehetőség közül  $1/N$  valószínűségünk van nyerni. Viszont ha tudjuk hogy "nem választunk újra" veszített, és a fazon már kinyitott egy ajtót, akkor  $N-2$ -ből kell csak választanunk. Utolsó sorban kiírja az nbatfai és ezen kód is sanity check jelleggel a két startégia nyertes kimeneteleinek összegét, ami  $N=3$  esetben egyenlő  $N$ -nel. Magasabb  $N$ -nél ez nem így van, hisz ha "nem választunk újra" veszített és a fazon kinyitottegy ajtót és  $N>3$  akkor nekünk kisebb mint 100% esélyünk van "újra választunk"-kal nyerni.

```
1 #include <iostream>
2 #include <stdlib.h>
3
4 int main()
5 {
6     const int trial_count = 100000;
7     const int door_count = 3;
8     int count_keptbet_won = 0;
```

```
9  int count_newbet_won = 0;
10 for (int i = 0; i < trial_count; i++)
11 {
12     if(rand() % door_count == 0){
13         count_keptbet_won++;
14     }else{
15         if(rand() % (door_count-2) == 0) count_newbet_won++;
16     }
17 }
18 std::cout<< "Monty Hall"<< std::endl;
19 std::cout<< "Trial count " << trial_count << std::endl;
20 std::cout << "Wins(first bet) " << count_keptbet_won << std::endl;
21 std::cout << "Wins(next bet) " << count_newbet_won << std::endl;
22 std::cout << "Win Ratio " << ((count_newbet_won==0) ? 0.0 : (static_cast<double>(count_keptbet_won)/static_cast<double>(count_newbet_won)))<<
    double>(count_keptbet_won)/static_cast<double>(count_newbet_won)))<<
    std::endl;
23 std::cout << "Sum (sanity check)" << count_keptbet_won+count_newbet_won <<
    << std::endl;
24 return 0;
25 }
```

## 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](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  $(n-1)*\text{valamennyi}+(n-1)$ , ami osztható  $(n-1)$ -el
- $(n+1)!+n=1*2*3*\dots*(n-1)*n*(n+1)+n$ , azaz  $n*\text{valamennyi}+n$ , 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 Bاتفai, nbاتفai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>

library(matlab)

stp <- function(x) {

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

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

Soronként 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 képzi, tehát 3-2, 5-3, 7-5, 11-7, 13-11.

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

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

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

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

Kivette a primes-ból a párok első tagját.

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

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

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

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

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

Az  $1/t_1\text{primes} + 1/t_2\text{primes}$  pedig ezeket a törteket rendre összeadja.

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

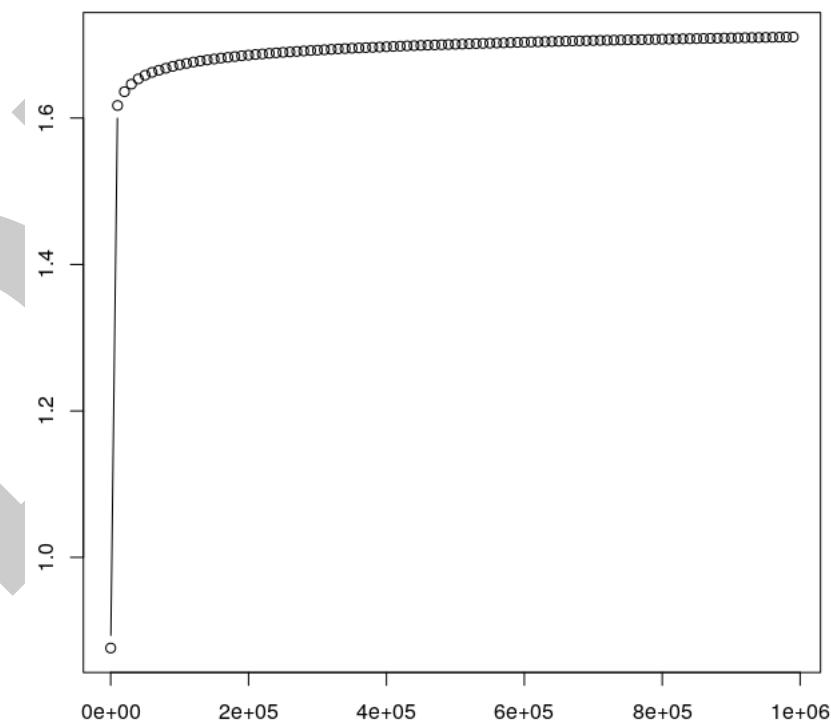
Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvénnyel.

```
sum(rt1plust2)
```

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

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

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

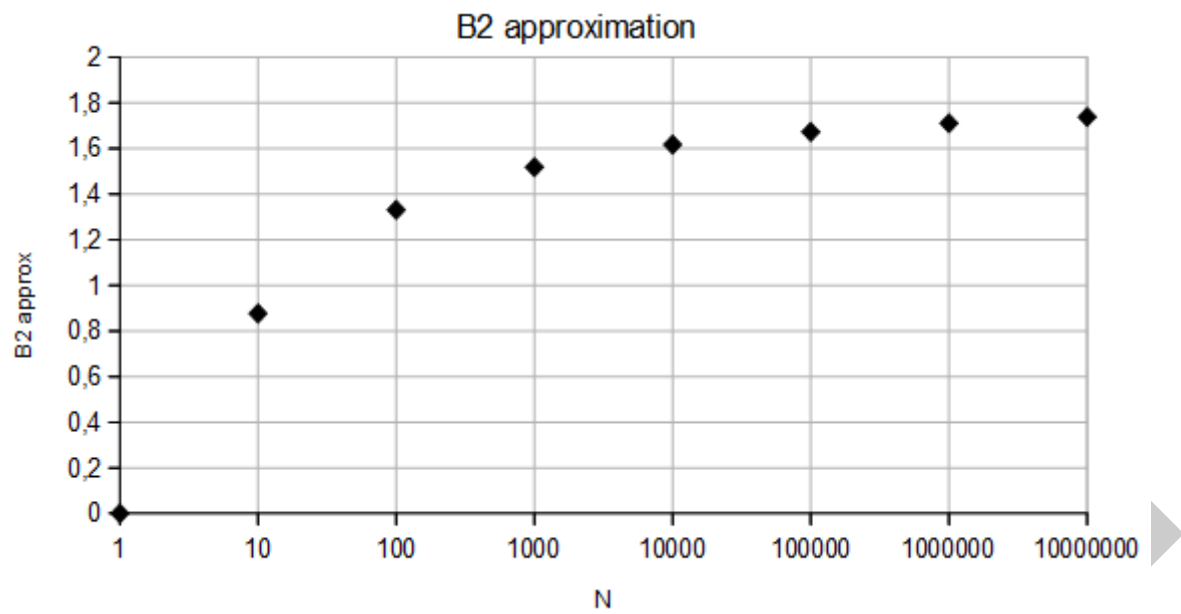


2.16. ábra. A  $B_2$  konstans közelítése

Az nbatfai program C++-ban bitset-tel. (Bitset miatt compile time tudnunk kell N-et.)

```
1  #include <iostream>
2  #include <bitset>
3  #include <math.h>    // sqrt
4
5  const int N = 100; // The catch is we have to know N compile time
6
7  int main()
8  {
9      auto bs = std::move(std::bitset<N+1>{}.set());
10     bs.set(0, false);
11     bs.set(1, false);
12     int m = sqrt(N);
13     for (int p=2; p<=m; p++)
14         if (bs.test(p))
15             for (int i=p*2; i<=N; i += p)
16                 bs.set(i, false);
17     double b2approx = 0.0;
18     for(int i = 2; i <bs.size();i++)
19         if(bs.test(i) && bs.test(i-2))
20             b2approx+=1.0/(i-2)+1.0/i;
21     std::cout<<"B2 approximation for "<<N<<" is "<< b2approx <<std::endl;
22     return 0;
23 }
24 /* N      B2approx
25 * 10      0.87619
26 * 100     1.33099
27 * 1000    1.51803
28 * 10000   1.61689
29 * 100000  1.6728
30 * 1000000 1.71078
31 * 10000000 1.73836
32 */
33 /*
34 * Didn't use accu, functional or lambdas
35 * But you could do it based on nbatfais R code...
36 * std::accumulate(t1.begin(), t1.end(), 0.0, [](const double &a, const ←
37 * double &b){return a+(1.0/b)+(1/(b+2.0));});
38 */
```





2.17. ábra. Az c++  $B_2$  konstans közelítés eredményei



#### Werkfilm

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

## 3. fejezet

# Helló, Chomsky!

### 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával 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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 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](https://youtu.be/9KnMqrkj_kU) (15:01-től).

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)

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

Tanulságok, tapasztalatok, magyarázat...

### 3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás videó: [https://youtu.be/06C\\_PqDpD\\_k](https://youtu.be/06C_PqDpD_k)

Megoldás forrása: [bhax/thematic-tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/Chomsky/l337d1c7.1](https://github.com/bhax/thematic-tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.1)

```
%{
#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;
}
```

Tanulságok, tapasztalatok, magyarázat...

## 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 megy ránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

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

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?

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

$\$ (\backslash \text{forall } x \backslash \text{exists } y ((x < y) \wedge (y \text{ \textit{prím}})) \wedge (\neg (y \text{ \textit{prím}}))) \leftrightarrow$   
 $\$$

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

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/MatLog\\_LaTeX](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX)

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, [https://youtu.be/AJSXOQFF\\_wk](https://youtu.be/AJSXOQFF_wk)

Tanulságok, tapasztalatok, magyarázat...

### 3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

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

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

Megoldás videó:

Megoldás forrása:

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://bhax.thematic-tutorials.com/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr.c), [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/Chomsky/fptr2.c](https://bhax.thematic-tutorials.com/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;

    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");
    }

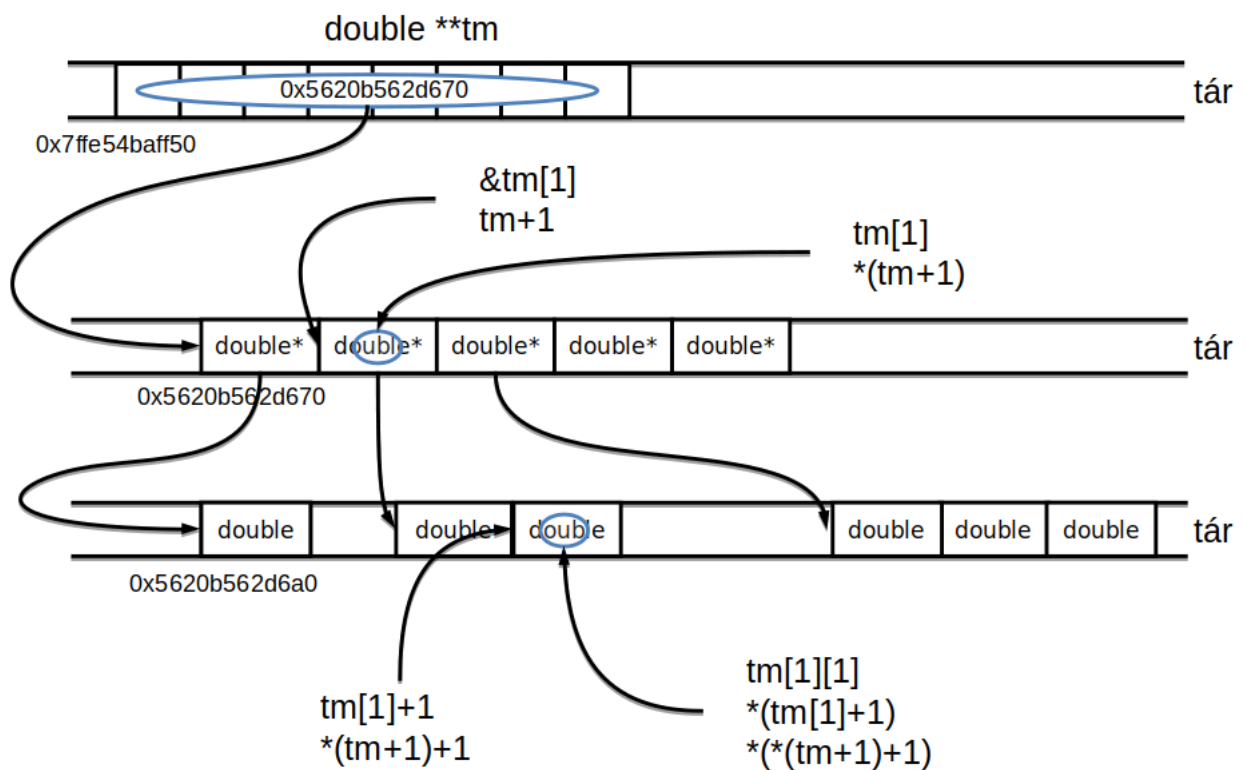
    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
    *(tm[3] + 2) = 44.0;
    (*(tm + 3) + 3) = 45.0;

    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](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](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...

## 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++](https://github.com/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 rácspont nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végtelen sok  $z$ -t megvizsgálni, ezért csak véges sok  $z$  elemet nézünk meg minden rácsponthoz. Ha közben nem lép ki a körből, akkor feketére színezzük, hogy az a  $c$  rácspont a halmaz része. (Színes meg úgy lesz a kép, hogy változatosan színezzük, például minél későbbi  $z$ -nél lép ki a körből, annál sötétebbre).

## 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 [bhax/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
// 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>

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](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](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](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 -l --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\\_halmazzal](https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal).

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](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 elspájzolod 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](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](https://youtu.be/OKdAkI_c7Sc)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Chrome](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](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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala)

Tanulságok, tapasztalatok, magyarázat...

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Programozási alapfogalmak

#### 10.1.1. Adattípusok

(28) tartomány, műveletek, reprezentáció, egyszerű, összetett, mutató

Mielőtt beszélünk a típusokról nézzünk egy nem típusos nyelvet! A [NANDTOTETRIS]-hez írtunk egy c++ interpreter jellegű programot, mely úgy viselkedik mintha egy vlós CPU lenne(csak jóval egyszerűbbek az opcode-ok). A lényeg, hogy 2 regiszter van. A és D. A "Adress" ugyanis a jump-ok mindig az A-ban lévő értékű címre ugranak. D "Data" register pedig egy "sima" regiszter. "A" regiszterrel a trükközés azért kell, mert így borzasztóan egyszerűvé válnak az opcode-ok. 0-kat és 1-eseket nem akarunk írni, ezért írtunk rá egy assemblert. Alább látható egy szuper egyszerű assembly kód erre a teljesen minimalista kis gépre.

```
@2
D=A
@3
D=D+A
@0
M=D
```

Direkt a fenti egyszerűbbet mert látható a példán egy gcc -S -el készült sima main-ből ez lesz a körítés miatt.

```
.file "one.c"
.def __main; .scl 2; .type 32; .endef
.text
.globl main
.def main; .scl 2; .type 32; .endef
.seh_proc main
main:
    pushq %rbp
    .seh_pushreg %rbp
    movq %rsp, %rbp
    .seh_setframe %rbp, 0
```

```
subq $32, %rsp
.seh_stackalloc 32
.seh_endprologue
call __main
movl $1, %eax
addq $32, %rsp
popq %rbp
ret
.seh_endproc
.ident "GCC: (Rev1, Built by MSYS2 project) 7.2.0"
```

Sajnos, el kellett engednem a teljes leírást, mert egyszerűen nincs rá idő, de komolyan ajánlom mindenkinek a [\[NANDTOTETRIS\]](#)-t. Alább például látszik egy szuper bugyuta kis ALU. Akármennyire bugyuta és tele van csalással a lényeg, hogy közelebb visz a szoftver és hardware találkozásához, ahol az igazi varázslat történik. (Hisz papíron ugyan Gödel megcsinálta, de sok idő kellett mire mindekinek lett macskáskép nézegetője.) Másrésztől nincs jobb érzés, mint amikor megcsinálja az ember a kapukat, majd ráküldi a kódot és megtudja vele csinálni a "for"-t! Tényleg fáj a szívem hogy nincs módomban berakni a doksiba. De őszintén ajánlom a könyvet, mert valójában az NEM EGY KÖNYV. Minden fejezet egy minimális elméleti alapozó és utána szuper egyértelmű task-ok vannak, TESZTEKKEL és platformmal együtt. Annyi, hogy én nem szeretem a Java-t mert az Oracle gonosz, ezért csak az assembler-es részt rossz minőségű c++-ban reprodukáltam a [ide](#).

Ha valakit abszolút nem érdekel a dolog, akkor is egyszer javaslom, csak amiatt, hogy átérezzük, hogy mennyire komoly segítséget adnak a mérnökinfósok és villamos mérnökök nekem illetve nekünk.

Ha pedig valakit a mérnökinfósok sem érdekelnek és nem szeret olvasni, legalább vessen egy pillantást [erre](#).

```
CHIP ALU {
  IN
    x[16], y[16], // 16-bit inputs
    zx, // zero the x input?
    nx, // negate the x input?
    zy, // zero the y input?
    ny, // negate the y input?
    f, // compute out = x + y (if 1) or (x and y) (if 0)
    no; // negate the out output?

  OUT
    out[16], // 16-bit output
    zr, // 1 if (out == 0), 0 otherwise
    ng; // 1 if (out less than 0), 0 otherwise

  PARTS:

  // X INP
    Mux16(a=x, b[0..15]=false, sel=zx, out=xt);
    Not16(in=xt, out=xtn);
    Mux16(a=xtn, b=xtn, sel=nx, out=xarg);
  // Y INP
```

```
Mux16(a=y,b[0..15]=false,sel=zy,out=yt);
Not16(in=yt,out=ytn);
Mux16(a=yt,b=ytn,sel=ny,out=yarg);
// F
And16(a=xarg,b=yarg,out=oand);
Add16(a=xarg,b=yarg,out=oadd);
Mux16(a=oand,b=oadd,sel=f,out=o);
// OUT POST
Not16(in=o,out=onot);
Mux16(a=o,b=onot,sel=no,out=oo);
// ZR
Or16Way(in=oo,out=tzr);
Not(in=tzr,out=zr);
// NG
And16(a[0..15]=true,b=oo,out[15]=ng,out[1..14]=drop);
// OUT
Or16(a=oo,b[0..15]=false,out=out);
}
```

A lényeg, hogy az assembler (label és egyéb dolgok mellett) elsősorban azt a célt szolgálja, hogy a fenti szöveg átforduljon bytecode-ra. Alább látható a fordított gépkód.

```
00000000000000010
1110110000010000
00000000000000011
1110000010010000
0000000000000000
1110001100001000
```

Nincsenek típusok, minden "szó" N mennyiségű bitből álló rendezett 16-os. Műveleteket nem definiálhatunk magunk, hisz azt a CPU csinálja.

Innentől kezdve bármit tanulunk emlékezzünk arra, hogy hasonló lesz a vége. (Persze a valóságban jóval összetettebb, de 1-esek és 0-k lesznek a legadvancedebb cpp kódból a nap végén.)

A típus megadja a gépnek hogy mikor írtunk egy programot és ráengedjük a lexert, parsert, compilert vagy interpretert akkor mit fogadjon el egyáltalán. Azaz hogy milyen elemei lehetnek. Azaz a típus egy halmazként is felfogható, melynek elemei a lehetséges értékek. Halmazoknál ugye felsorolhatjuk, de akár ha pro-k vagyunk szabályokkal is megadhatjuk (emlékezzünk a természetes számok halmazán successor-ra, vagy akár a modulo kongruencia osztályokra egészeknél)

A típus megadja a gépnek hogy milyen műveleteket és hogyan kell végezni. Például egy bool-t ha negálunk más történik, mintha egy int-et. Sőt, sokszor nem is lehet bizonyos dolgokat értelmezni, például Várterész Tanárnő nem nagyon szorozgatott igaz-t hamis-sal (majd később belemegyünk a szorzásba, most simán csak gondoljunk gyerekkorunkban tanultakra).

A típus megadja a gépnek hogy hogyan kell interpretálni az adatot. Például gondoljunk egy egyszerű C struct-ra, van két char fieldje "foo" és "bar". Elrakjuk valahova a memóriába (és tároljuk a címét), majd kis idő múlva kellene az "b" field. Honnan fogjuk tudni, hogy a sok bit közül hol kezdődnek a "b" field bitjei illetve, hogy hány bitből is áll? Például erre (is) ad választ a primitív char típus.

Amit még nagyon fontos lefektetni, az az hogy inheritance, primitív típus, template mind csak fluff és eyecandy a CPU szempontjából. Előbb utóbb mindenből 0101 0011 1111 0000 lesz. Igen igen 32 64, plusz valójában nem egy szó kerül be stb. de a lényeg hogy mindent számokra képezünk le. Az összes többi dolog csak és kizárólag azért kell, mert az ember biológiailag nem 0 és 1 olvasásra és nagy sebességű aritmetikai műveletek elvégzésére fejlődött hanem az ágakon tekergő kígyók elől való elugrálásra.

A tankönyv említi hogy forrásszöveget írunk, amelyből aztán két mágiával lehet gépi kód. Compiler-es és interpreter-es. Ez a valóságban sajnos nem ilyen egyértelmű. Nézzünk például egy Java-s példát. Igen compiled, de...mégis a VM stack machine-en fut. A stack machine csak egy absztrakció, nem a tiszta vas. Ez is a középpontja az Java azon ígretének mi szerint "write once, run before Oracle sues you for using VM without paying your subscription for server side usage".

Másik Java példa: Project Lombok. Fel annotáljuk meta nyelven a forrás szöveget, és a class file-ba belegeneráljuk a boilerplate code-ot, anélkül, hogy telenyomnánk vele a source-t.

Másik Java példa: Spring, xml vagy reflection (annotation) based meta adatok. Igen a forrás fájl része, de egy framework használja az adatokat...

És a akkor a kedvencem: Írok egy progit C-ben. Mondjuk egy macskáskép játék. A business logic-ot direkt C helyett Lua-ban írom, magyarul a C programom tartja számon a Lua state-t. A programom compiled, viszont ha a lua szkriptet változtatok alatta akkor gond nélkül hot swappelhetem mondjuk szerver oldalon. Most akkor része a business logic a programomnak? Vagy a programom egy hyper program ami önmaga nem a macskás játék? De hát a grafikus funkciók C-ben vannak írva? A lényeg, hogy nem ilyen egyértelmű a dolog.

Arról pedig már ne is beszéljünk, amit egy JIT compiler egy átlagos hétfő délután csinál.

A tankönyv ezután belemegy a fordító programok világába. Ez ahogy láttuk nem egy merev dolog, de ennél még rosszabb is történhet. Egyes "compiler"-ek azért vannak hogy C-kódot generáljanak valami deklaratív jellegű nyelvből. De ennél még rosszabb, hogy van aminek az a célja hogy C kódot fordítson Javascriptre. A fordítás általános feladatai a tankönyv szerint a következők:

- lexikális elemzés
- szintaktikai elemzés
- szemantikai elemzés
- kódgenerálás

A könyv kiemeli, hogy lehet szó előfordítókról. Most egy tanulságos történet: Java spring-based web server. Hibernate előtti időszak, szóval perzisztenciát from scratch. Amerikaiak úgy döntöttek, hogy egy perzisztens class-t annotációkkal fognak "dekorálni" (ez akkor még nagy szó volt, mert ez még az xml config-os spring era), és technikai okokból, ha ez megtörtént, az annotációkban megjelölt információk alapján a SUPER class auto generálni fogjuk. Igen `Derived extends Base` és `Base` még nem volt kész, hanem `Derived` alapján jött létre "automatán" a `Derived` annotált source kód alapján. Például olyan célt szolgált, hogy a null check-ek validálások, propertyeventchanged küldések stb. ne kézzel íródjanak. **Ő az.** Az már egy másik cseresznye a tortán, hogy nem a hétköznapi módon csináltak `Product` táblát, `Employee` táblát stb., hanem például `TypeIdentifier`, `AttributeTypeIdentifier` és hasonló táblák voltak, azaz runtime lehetett új "típusokat" létrehozni, úgy hogy ezek nem csak a field-eket örökölték, hanem viselkedést is (igen a munka nagyrésze az application layerben ment).



A könyv kitér rá, és tényleg nagyon fontos a linker. Egyébként így elsőre prog 1-ből az lesz az előnye, hogy nem kell mindig az egészet újra fordítani. Persze ha nem kell mindig újra fordítani. Persze a compile-olgatásnak és linkelésnek is meg van a maga ára. Például ha ritkán változó dologról van szó, akkor lehet precompiled header-ekkel dolgozni.

Na jó...de mi ez az egész linkelés? Pl. C++ esetén a compilation unit Foo, illetve van egy FooMain-ünk ahol használjuk. Külön külön fordítjuk, és ha minden rendben akkor végül együtt kell működniük majd, tehát linkelésnél valamilyen módon a FooMain belüli használathoz társítani kell a Foo belüli implementációt. A szerződés az együttműködésre Foo.hpp.

```
1 #ifndef FOO_H
2 #define FOO_H
3 class Foo{
4 public:
5     Foo(int v);
6     int getA() const;
7 private:
8     int a;
9 };
10 #endif
```

```
1 #include "Foo.hpp"
2
3 Foo::Foo(int v){
4     this->a = v;
5 }
6
7 int Foo::getA() const
8 {
9     return this->a;
10 }
```

```
1 #include "Foo.hpp"
2 #include <iostream>
3 int main(){
4     Foo foo(1);
5     std::cout << foo.getA() << std::endl;
6     return 0;
7 }
```

```
UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ g++ -c Foo.cpp

UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ g++ -c FooMain.cpp

UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ g++ -o FooMain.exe Foo.o FooMain.o

UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ ./FooMain.exe
1
```

10.1. ábra. Foo és Foo Main

Na jó... de mi történik, ha megváltozik az a field? Mi van ha kifele int-et mutatok, de valójában másképp akarom tárolni?

Amíg a hpp változatlan addig azt csinállok implementációban amit akarok!

De...szóval mi van ha esetleg az a implementációjához akarok hozzányúlni. Bad luck! Hpp-t módosítani kell és akkor már nem tudnak ellened linke... VÁRJUNK CSAK!

Egy kis trükközéssel encapsulating kivitelezhető ezen kívánságra is, csak kompozíciót kell alkalmazni és egy struct-ba wrappelni amit rejtetni kívánunk.

```
1 #ifndef FOOABI_H
2 #define FOOABI_H
3
4 #include <memory>
5
6 class FooABI{
7 public:
8     ~FooABI();
9
10    FooABI(const FooABI&) = delete;
11
12    FooABI& operator=(const FooABI&) = delete;
13
14    FooABI(FooABI&&) = delete;
15
16    FooABI& operator=(FooABI&&) = delete;
17 public:
18    FooABI(int v);
19
20    int getA() const;
21 private:
22    struct Impl;
23    std::unique_ptr<struct Impl> impl_;
24 };
```

```
25 #endif
```

```
1 #include "FooABI.hpp"
2
3 struct FooABI::Impl
4 {
5     Impl(int v) : a(v) {};
6     int a;
7 };
8
9 FooABI::FooABI(int v) : impl_(new Impl(v)) {
10
11 }
12
13 FooABI::~FooABI() = default;
14
15 int FooABI::getA() const
16 {
17     return impl_>a;
18 }
```

```
1 #include "FooABI.hpp"
2 #include <iostream>
3 int main(){
4     FooABI fooabi(1);
5     std::cout << fooabi.getA() << std::endl;
6     return 0;
7 }
```

```
UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ g++ -c FooABI.cpp

UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ g++ -c FooABIMain.cpp

UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ g++ -o FooABIMain.exe FooABI.o FooABIMain.o

UBMCGU@UBMCGUPC MINGW64 ~/ws_school/bhax-derived/thematic_tutorials/bhax_textboo
k_IgyNeveldaProgramozod/cbook (master)
$ ./FooABIMain.exe
1
```

10.2. ábra. FooABI és FooABI Main

A tankönyvet régen írták, de igen, továbbra is nagyon fontos hogy ki hogyan és mihez kapcsolódik, viszont mivel telt múlt azóta az idő, vannak új trükkök!

A betöltés egy nagyon fontos dolog. Miért is? A gépekben az adat, az eljárás és a macskás képek nem különülnek el. Minden adat. Vicces túlzással élve a számítógép valójában egy ipari lyukasztó gép amit, nos lényegében lyukasztott kártyákkal programozunk, innentől kezdve ő kilyukasztja az utasítás kártyát ha kell, és ha kell lefuttatja a kilyukasztott végterméket egyaránt ha beadjuk neki. Számára lyuk-lyuk egyre megy.

### Betöltés

C++, C, Java oldalról nehéz megérteni a betöltést. Nézzük assemblyvel egy egyszerű példán:

```
@0
D=M
@INFINITE_LOOP
D;JLE
@counter
M=D
@SCREEN
D=A
@address
M=D
(LOOP)
@address
A=M
M=-1
@address
D=M
@32
D=D+A
@address
M=D
@counter
MD=M-1
@LOOP
D;JGT
(INFINITE_LOOP)
@INFINITE_LOOP
0;JMP
```

A lényeg, hogy van egy LOOP label-em. Ez egy hely a kódban ahova ugorhatok. Hogy ugrom oda? Long story short @LOOP-al betárolom LOOP helyét majd JGT-vel ugrom.

Ok. Pszeudokódban megy, de...Mennyi is a LOOP label címe? Mármint konkrétan nekem kellene, hogy az most akkor 0111 0111 0111 1111? Honnan tudom?

Naív válasz: Oké, 0 memória címre lesz betöltve a programom, szóval simán kiszámolom hogy az @address(hisz oda fogok ugrani, mert a LOOP az csak egy sajtos papír "tag", tag alatt az angol tag-et értem)

Ez egy tökéletes megoldás lehet Nintendo-n, vagy nem tudom... valami ROM-on!

De akkor mi van, ha én nem oda kerülök, hanem mondjuk már előttem vannak dolgok, mondjuk egy macskakép sokszorosító?

Egyszerű megoldás: Akkor derüljön ki LOOP értéke, mikor én elhelyezésre kerülök! Zseniális!

Viszont...nos, innentől kezdve én elmozdíthatatlan vagyok! Pontosabban elmozdíthatnak, de mivel a LOOP egy konkrét érték, ezért ha arébb raknak, akkor rossz területre fog hivatkozni.

És akkor például itt jöhetnek trükkös megoldások a cím újra számításra, vagy esetleg arra, hogy én ne direktbe hivatkozzak egyenesen a fizikai címre, hanem magamhoz képest relatív.

Java esetben ez másképp van hiszen egy stack machine-be pakolunk dolgokat, ami az Oracle szerint write once run...

Persze a VM egyébként egy [nagyon jó dolog](#). Vagy például az eve online [Stackless Python](#)-t használ ami egyébként ugyanúgy a unmutabilityt választotta, hasonlóan az Erlang-hoz

Interpreternél ugye nincs szükség ekkora hercehurcára, kivéve ha van szükség. Mármint például egyes interpretált nyelveknél direkt egy előfordított formába rkhajjuk a szkriptet és akkor kicsit gyorsítani tudunk a dolgokon.

Interpreteres esetekben persze mindig ott a lehetőség, hogy a CPU intenzív dolgokat natívba rakjuk. Pl. a [dönt starve](#) esetén Lua intézi az üzleti logikát, ami az állatok párzási időszakban erősödő agressziójáért felelős, de a grafika, fizika, collision C/C++ oldalon van tartva. Azért nem mondok tiszta Cpp-t, mert Lua raw c ptr-eket fogad, illetve C-s callbackek szolgálnak hook-ként a lifecycle eventekre (magyarul ha a Lua gc elakar takarítani valamit, és az egy küldő kódból származó raw ptr, akkor egy user defined c callback-et hív ezen ptr-el. Mi például itt tudjuk az átküldött címre hívni a destruktort explicit, utána visszakerül luanak az irányítás. Azért nincs free vagy delete, mert az is customizeable, azaz lehet például, hogy mi írunk alá memory managementet, mert folyamatosan az OS-től kérni apró chunkokat elég lassú.).

Ezzel az egész résszel az volt a célom, hogy kifejezzem, hogy a könyv nagyon jó, de már régóta eltűntek azok az éles határok, illetve mivel nem láttam pontos definíciót ezért nehéz egyáltalán megtámadni is.

Természetesen a lexikális elemzés során megtörténik a forrás szöveg lexikális egységekre történő bontása. Ez ma is így van.

Egyébként viszont az is egy érdekes kérdés, hogy a Cpp type system az imperatív nyelvbe hogy kerül bele. Mármint arra a vicces dologra akarom felhívni a figyelmet, hogy mondjuk én egy extends-el egy abszolút nem imperatív dolgot csinállok, a type inference pedig...nos ennél kevésbé imperatív dolog nincs. Persze, igen, C-style cast.

Most bele lehetne menni szárazon a BNF-be, de ennél aranyosabb a  $Q=\{"I","3","t",a++b \mid a,b \text{ eleme } Q\}$  Szóval [133t](#), de Várterész Tanárnő egyébként szó szerint ilyen "elemzést végzett", amikor felírta az ábécét és a szabályokat.

A szintaktikai szabályok kicsit hajlékonyak, például ha Tanárnő hiányzik, akkor emlékezzünk arra, hogy a `-Wpedantic`(pl: field initialization sorrend csak a deklarációs sorrendben megengedett)

Imperatív nyelveknél a programozó mondja meg hogy hogyan, ezért tele van bugokkal. Mellette szól viszont, hogy [gyorsabb kódot lehet így írni az elméletileg lehetségesnél](#)

Deklaratív nyelveknél a programozó nem mondja meg hogyan. Emiatt nincs hiba. Technikailag. De természetesen abszolút nem az fog történni amire az ember gondol és sok szerencsét a Prolog debuggolással.

Imperatív nyelvekhez még talán annyit, hogy...nos az OOP nagyon jó dolog. Bizonyos feladatokra. Folyamatos vessző paripám az [ECS](#). Nem ez nem egy nyelv, hanem egy megközelítési forma. Az egész arról szól hogy passzív adatstruktúráim vannak és a viselkedést megvalósító részeket megpróbálom (bár általában nehéz) állapot mentesíteni. Szerintem az OOP az emberi intuíciót és a problémákról történő gondolkodást elősegíti, de semmilyen bizonyíték nincs arra, hogy karbantartható codebase-hez vezet. (Például a [Tony Hawk](#)) Vannak olyan nyelvek melyek az interface-t (vagy teljesen absztrakt class-t) preferálják, és kigyomlálták a hétköznapi java-ban burjánzó inheritance fákat, erdőket.

- numerikus - pl.: (C)int, (C)float
- karakteres - pl.: (C)char
- karakterlánc - pl.: ezt általában nem igazi primitív, hanem a nyelv mellé adott alap library része, például Erlang-ban egyébként egy lista
- logikai - pl.: (C) bool
- felsorolós - pl.: (C) enum
- sorszámozott - pl.: (Pascal) byte, word, int, ...

A nyelv által definiált egyszerű típusokból van lehetőség új strukturált típusok összerakására ezek az összetett típusok.

A típusnak pedig végül kell hogy legyen valami azonosítója, hogy tudjunk rá a későbbiekben hivatkozni.

A mutató típusról kicsit külön érdemes beszélni. A mutató egy tárbeli címre mutat vagy **NULL**-ra. Érdemes tudni, hogy a mutatott cím egyáltalán milyen típus. Azaz a `char* foo` ptr típusú, viszont amire mutat azt char-ként fogja "interpretálni". Persze egy int-et tároló mem területre rámehetünk egy char ptr-rel gond nélkül.

### 10.1.2. A nevesített konstans

(34) név, típus, érték

Szerintem a könyvbeli preprocesszor-os példa technikailag nem igaz. Az hogy a preproceszor mit csinál már a Turing-os fejezetben bemutattam, plusz mutattam olyat is amikor makró alkalmaz makrót ami kódot injektál. Na egy ilyen esetben látszik hogy a preprocesszor csak egy "szövegszerkesztő" eszköz, nem a nyelv része, plusz úgy ütöm felül a a define-okat ahogy akarom.

Ellenben a `const int =6;` egy konstans. típus, const qualifier, és értékadás, just like God intended. Ha már itt tartunk akkor itt a világ legmegbízhatóbb Java kódja, ami viszont olykor mégis hibát okozhat (assuming that it wont get optimized away): `private static void foo(){};` He. He.

### 10.1.3. A változó

(35) név, attribútumok, cím, érték

A változóknak négy komponense van:

- név - pl.: user defined, scope-on belül egyedi
- attribútumok - pl.: típus, vagy qualifiers
- cím - stack, heap vagy manual
- érték - értékadás, itt annyit érdemes megjegyezni, hogy attól hogy létrehozom és kap címet, azt nem lehet várni hogy a tár tiszta legyen, úgyhogy érdemes lehet initelni.
- felsorolós - pl.: (C) enum
- sorszámozott - pl.: (Pascal) byte, word, int, ...

### 10.1.4. Alapelemek az egyes nyelvekben

(39) innen csak a C nyelvész rész persze

Aritmetikai típusok

- integrális - egész (int, short[int], long[int]) : signed unsigned-ról már turingban írtam példával együtt
- integrális - karakter (char)
- integrális - felsorolásos (enum)
- valós - (float, double, long double)

Származtatott típusok

- tömb
- függvény
- mutató
- struktúra
- union
- void

## 10.2. Programozás bevezetés

[KERNIGHANRITCHIE] (2nd edition)

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

## 10.3. Programozás

[BMECPP]

## **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**

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](http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf) , 2004.

[SICP] Abelson, Harold, *Structure and Interpretation of Computer Programs*, MIT , 1996.

## 11.7. My Little Ponys

[CHISOMORPH] Sorensen, Morten Heine B, *Lectures on the Curry-Howard Isomorphism*, Pdf , 1998.

[DENOTATIONALSEMANTICS] Allison, Lloyd, *A practical introduction to denotational semantics*, Cambridge University Press , 1986.

[NANDTOTETRIS] Nisan, Noam, *The Elements of Computing Systems*, Homepage of the book , 2005.

[PROGLANGS] Harper, Robert, *Practical Foundations for Programming Language*, Carnegie Mellon University , 2016.

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.