

# Univerzális programozás

---

## Így neveld a programozód!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

## COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Borbíró, Szabolcs	2019. május 10.	

## REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna <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

## 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?	7
2.3. Változók értékének felcserélése	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun tétel	11
2.8. A Monty Hall probléma	11
<b>3. Helló, Chomsky!</b>	<b>13</b>
3.1. Decimálisból unárisba átváltó Turing gép	13
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	13
3.3. Hivatkozási nyelv	13
3.4. Saját lexikális elemző	14
3.5. Leetspeak	15
3.6. A források olvasása	17
3.7. Logikus	18
3.8. Deklaráció	19

<b>4. Helló, Caesar!</b>	<b>21</b>
4.1. double ** háromszögmátrix	21
4.2. C EXOR titkosító	23
4.3. Java EXOR titkosító	24
4.4. C EXOR törő	25
4.5. Neurális OR, AND és EXOR kapu	27
4.6. Hiba-visszaterjesztéses perceptron	27
<b>5. Helló, Mandelbrot!</b>	<b>28</b>
5.1. A Mandelbrot halmaz	28
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	28
5.3. Biomorfok	28
5.4. A Mandelbrot halmaz CUDA megvalósítása	28
5.5. Mandelbrot nagyító és utazó C++ nyelven	28
5.6. Mandelbrot nagyító és utazó Java nyelven	29
<b>6. Helló, Welch!</b>	<b>30</b>
6.1. Első osztályom	30
6.2. LZW	31
6.3. Fabejárás	42
6.4. Tag a gyökér	43
6.5. Mutató a gyökér	43
6.6. Mozgató szemantika	43
<b>7. Helló, Conway!</b>	<b>58</b>
7.1. Hangyaszimulációk	58
7.2. Java életjáték	58
7.3. Qt C++ életjáték	58
7.4. BrainB Benchmark	59
<b>8. Helló, Schwarzenegger!</b>	<b>60</b>
8.1. Szoftmax Py MNIST	60
8.2. Mély MNIST	60
8.3. Minecraft-MALMÖ	60

<b>9. Helló, Chaitin!</b>	<b>61</b>
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .	61
9.2. Weizenbaum Eliza programja . . . . .	61
9.3. Gimp Scheme Script-fu: króm effekt . . . . .	61
9.4. Gimp Scheme Script-fu: név mandala . . . . .	62
<b>10. Helló, Gutenberg!</b>	<b>63</b>
10.1. Programozási alapfogalmak . . . . .	63
10.2. Programozás bevezetés . . . . .	63
10.3. Programozás . . . . .	64
<b>III. Második felvonás</b>	<b>65</b>
<b>11. Helló, Arroway!</b>	<b>67</b>
11.1. A BPP algoritmus Java megvalósítása . . . . .	67
11.2. Java osztályok a Pi-ben . . . . .	67
<b>IV. Irodalomjegyzék</b>	<b>68</b>
11.3. Általános . . . . .	69
11.4. C . . . . .	69
11.5. C++ . . . . .	69
11.6. Lisp . . . . .	69

## Ábrák jegyzéke

4.1. A double ** háromszögmátrix a memóriában .....	23
---	----



# Előszó

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

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

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

## Hogyan forgasd

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

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

## Milyen nyelven nyomjuk?

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

## Hogyan nyomjuk?

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

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

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

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



#### A DocBook XML 5.1 új neked?

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

# **I. rész**

## **Bevezetés**

# 1. fejezet

## Vízió

### 1.1. Mi a programozás?

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

### 1.2. Milyen doksikat olvassak el?

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

### 1.3. Milyen filmeket nézzek meg?

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

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

DRAFT

## **II. rész**

### **Tematikus feladatok**

**Bátf41 Haxor Stream**

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

---

DRAFT

## 2. fejezet

# Helló, Turing!

### 2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Végtelen ciklus egy olyan ciklus aminek egy olyan feltételt adunk meg, ami soha nem teljesül, így ameddig le nem lőjük futni fog.

0%-os százalékban dolgoztató végtelen ciklus!

Először is include-oljuk a unistd.h header fájlt, mivel benne található meg a sleep függvény. A main függvényben létrehozunk egy while ciklus, ahol feltételnek megadjuk az 1-et és a while cikluson belül meghívjuk a sleep függvényt, ami a program futását késlelteti, így nem fog CPU-t használni még ha fut is!

```
#include <unistd.h>

int main() {
    while(1) {
        sleep(1);
    }
    return 0;
}
```

100%-ban dolgoztató végtelen ciklus, de csak egy magot!

Most elég nekünk a stdio.h header fájl, mivel nem kell a sleep függvény. Az előzőhöz képest annyi változik, hogy kitöröljük a sleep-et és így a program rendesen futni fog, de csak az egyik magot veszi igénybe.

```
#include <stdio.h>

int main() {
```



```
while (1)
{
    {}
    return 0;
}
```

100%-ban dolgoztató végtelen ciklus és mind a 4 magot!

Itt párhuzamosan fogjuk pörgetni a CPU-t az Openmp segítségével! Először is include-juk a omp.h header fájlt, amibe benne van ami kell nekünk. Az main függvénybe beírjuk a #pragma omp parallel-t, ami a CPU párhuzamos futásáért fog felelni! Utána egy while ciklus és kész is vagyunk.

Ezt a program fordítását a következő képen tudjuk fordítani.

gcc vegtelen.c -o vegtelen -fopenmp

```
#include <omp.h>

int main() {
    #pragma omp parallel
    while (1)
    {
        {}
        return 0;
    }
}
```

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100 (T100)
false
```

ezt a kimenetet adja.

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

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

Nem tudunk olyan programot írni ami megtudja jósolni, hogy egy program le fog-e fagyni. De elméletileg azt megtudjuk nézni ennek a pszeukódnak a segítségével, hogy van-e benne végtelen ciklus.

A T1000-esbe bele ágyazuk a T100-ast. A T1000 program ha saját magát vizsgálja, ha megtalálja a végtelen ciklust le fog fagyni, ha nem akkor meg tovább fut.

## 2.3. Változók értékének felcserélése

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

Megoldás videó: [https://bhaxor.blog.hu/2018/08/28/10\\_begin\\_goto\\_20\\_avagy\\_elindulunk](https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk)

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Egy plusz változóval!

```
#include <stdio.h>

int main(){
    int a=5;
    int b=7;
    int c=a;
    a=b;
    b=c;
    printf("%d\n", a);
    printf("%d\n", b);
    return 0;
}
```

```
#include <stdio.h>

int main(){
    int a=5;
    int b=7;
    a=a*b;
    b=a/b;
    a=a/b;
    printf("%d\n", a);
    printf("%d\n", b);
    return 0;
}
```

```
#include <stdio.h>

int main(){
    int a=5;
```

```
int b=7;
a=a^b;
b=a^b;
a=a^b;
printf("%d\n", a);
printf("%d\n", b);
return 0;
}
```

## 2.4. Labdapattogás

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

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

Megoldás forrása: <https://github.com/Borbiro/Prog1/blob/master/labda.c>

Tanulságok, tapasztalatok, magyarázat...

A labdapattogtatáshoz szükséges include-ni a curses.h és unistd.h header fájlokat! A curses.h, azért kell hogy új ablakot nyithassunk meg és még mellette a pozíciókat meg a mozgást meg tudjuk adni. Az unistd.h meg a programm milyen gyorsan fusson ("A labda milyen gyors legyen").

Először is deklarálunk kell egy ablakot, ami a jelen pillanatba win nevet viseli. Utána deklarálunk hat változót amiből 2 lesz a kiindulási pont, 2 a mozgásnak a nagysága, 2 meg a szélsőérték.

Lértehozunk egy for ciklust aminek nem adunk feltételeket, hogy addig fusson ameddig ki nem lőjük!(végtelen ciklus)

A getmaxyx-el megcsináljuk az asztalt meg a szélsőértékeket, utána pedig a labda pozícióit és amit mozgunk (jelen pillanatba egy "0") a mvprintw-vel.

A refresh-el megnyitjuk az asztalt, az usleep-el meg a sebességet határozzuk meg, minél nagyobb az érték annál lassabb.

A clear-el az előzőleg generált "labdákat" tüntetjük el így mindig egy lesz.

A if függvényel pedig ha hozzá ér a szélekhez akkor reciprokot von és "visszapattan".

## 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Írd 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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az a változónak megadunk egy értéket és a while ciklus segítségével, addig lépked ballra ameddig a nem lesz egyenlő 0-val. Ez a bitenként lépkedés.

```
#include "stdio.h"

int main()
{
    int a = 5;
    int c = 0;
    while (a!=0) {
        a = a << 1;
        c = c + 1;
    }
    printf("%d\n", c);
}
```

## 2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: <https://github.com/Borbiro/Prog1/blob/master/pagerank.c>

Tanulságok, tapasztalatok, magyarázat...

A pagerank sok böngészőnél használt keresőmotor, ami hiperlinkeket használ, ami számokat rendel az adott oldalhoz. Mindenki a saját lapján csak neki tetsző oldalakat oszt meg, így ez a pagerank-nál bizonyos szavazatot jelent.

## 2.7. 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)

Tanulságok, tapasztalatok, magyarázat...

A brun tétel ikerprímekről szól, az ikerprímek azok amiknek a különbsége 2. Az ikerprímek reciprók összege egy bizonyos összeghez konvergál, amit a tudosok kb. 1.8 és 2.2 közé tesznek, ezeket a számokat Brun-konstansnak nevezik.

## 2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/03/erdos\\_pal\\_mit\\_keresett\\_a\\_nagykonyvben\\_a\\_monty\\_hall-paradoxon\\_kapcsan](https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/MontyHall\\_R](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R)

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall probléma egy valószínűségszámításhoz kapcsolódó paradoxon. Arról szól, hogy 2 ajtó mögött rossz nyeremény van az egyik mögött meg egy jó! A játékos választ egy ajtót és a játék vezető kinyit neki egy ajtót ahol egy rossz nyeremény van és esélyt ad a játékosnak, hogy változtathason és itt a probléma, hogy változtasson vagy ne?

Monty csak azt az ajtót választhatja, hogy ne legyen az autó, szóval ha a játékos az autóra mutat akkor két választási lehetősége van, de  $2/3$  az esélye hogy kecskére mutatsz, így mivel az autó ajtaját nem nyithatja ki, ezért  $2/3$ -a valószínűsége ha váltasz autót találsz.

## 3. fejezet

# Helló, Chomsky!

### 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: <https://slideplayer.hu/slide/2108567/8/images/27/Decim%C3%A1lisb%C3%B3l+un%C3%A1risba+átváltó+Turing+gép>

Nincs egyszerűbb számrendszer mint az unáris számrendszer. Az ábrázolása egy karakterből áll például itt jelent pillanatba "I". A 3-as decimális számot például, úgy írjuk le hogy "III". Ez nagyobb számoknál nagy problémát jelenthet. Behozhatunk új karaktereket amivel jelöljük a 10-est /, a 100-ast \* vagy éppen az 1000-est !. Pl.: 1234=!\*///III. De mivel ezek nem a legszebb karakterek bevezeték az 1-es római számot, amit rengetegszer lehet ismételni. A turing gép ciklusa addig vonja a számból az 1-et míg 0 nem lesz.

### 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:

```
#include <stdio.h>

void main() {

    int long long a=998989889898898; //nem fordul le 89-es szabvánnyal
    printf("%llu", a);
}
```

```
borbiro@Borbiro-Lenovo-Y520-15IKBA:~/Documents$ gcc -std=c89 hivat.c -o hivat
hivat.c: In function 'main':
hivat.c:5:35: error: C++ style comments are not allowed in ISO C90
    int long long a=998989889898898;
hivat.c:5:35: error: (this will be reported only once per input file)
```

A 89-es szabvánnyal nem működik, de a 99-es el igen.

```
borbiro@Borbiro-Lenovo-Y520-15IKBA:~/Documents$ gcc -std=c99 hivat.c -o hivat
borbiro@Borbiro-Lenovo-Y520-15IKBA:~/Documents$
```

### 3.4. Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetén megjelenő valós számokat! Nem elfogadható olyan megoldás, amely maga olvassa betűként 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).

```
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
%%
```

ELőször is a dupla százalék jell között includoljuk a stdio.h header fájlt, utána létrehozunk egy változót melynek 0 kezdőértéket adunk és azután megadjuk hogy milyen karaktereket olvason be.

```
{digit}* (\. {digit}+)? {++realnumbers;
printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
```

Ezek után a digit csupán definiáljuk és megadjuk mely karakterekre mit adjon vissza. Ha találunk olyan karaktert a realnumbers-t növeljük 1-el.

```
int
main ()
```



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

Létrehozunk egy main függvényt amiben beírjuk a yylex() függvényt ami nagyon fontos mert ennek a segítségével indítjuk be a lexikálást. Utána meg csupán egy kiíratás.

```
borbiro@Borbiro-Lenovo-Y520-15IKBA:~/Documents$ lex -o leex.c leex.l
borbiro@Borbiro-Lenovo-Y520-15IKBA:~/Documents$ gcc leex.c -o leex
/tmp/cce5Pqc9.o: In function `yylex':
leex.c:(.text+0x4bc): undefined reference to `yywrap'
/tmp/cce5Pqc9.o: In function `input':
leex.c:(.text+0x10c9): undefined reference to `yywrap'
collect2: error: ld returned 1 exit status
borbiro@Borbiro-Lenovo-Y520-15IKBA:~/Documents$ gcc leex.c -o leex -lfl
borbiro@Borbiro-Lenovo-Y520-15IKBA:~/Documents$ ./leex
asdash4jj6
asdash[realnum=4 4.000000]jj[realnum=6 6.000000]
```

### 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/1337d1c7.1](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/IgyNeveldaProgramozod/Chomsky/1337d1c7.1)

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

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

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

    {'a', {"4", "4", "@", "/-\\\"}},
    {'b', {"b", "8", "|3", "|\"}},
    {'c', {"c", "(", "<", "{"}},
    {'d', {"d", "|)", "|]", "|\"}},
    {'e', {"3", "3", "3", "3\"}},
    {'f', {"f", "|=", "ph", "|#\"}},
```

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

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

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

```
%}
```

```
%%
```

```
. {
```

```
int found = 0;
for(int i=0; i<L337SIZE; ++i)
{
    if(l337d1c7[i].c == tolower(*yytext))
    {
        int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

        if(r<91)
            printf("%s", l337d1c7[i].leet[0]);
    }
}
```

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

        found = 1;
        break;
    }

}

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

}

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

Ez egy hasonló program mint az előző. Először is include-uk a kellő header fájlokat és létrehozunk egy konstanst. Az első részben létrehozunk egy struktúrát ami 4 karaktert fog kiválasztani és ahhoz generál véletlen számot. A szám és az if-ek segítségével a programm eldönti, hogy a négy lehetőség közül meiket választ. Az utolsó részben elindítjuk a lexikális elemzést.

### 3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezo) == 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 jelkezo 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 meggyránézésre, 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)`

Ez egy for ciklus ami 5-ször fut le.

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

Ez egy for ciklus ami 5-ször fut le.

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

Ez a for ciklus 5-ször fut le és az i++, elemeit hozzárendeli a tomb elemeihez.

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?

$\$ (\backslash \text{forall } x \backslash \text{exists } y ((x < y) \backslash \text{wedge} (y \text{ \textit{prím}}))) \$$   
Minden  $x$ , létezik olyan  $y$ , ahol  $x$  kisebb  $y$ -nél és  $y$  prímszám.

$\$ (\backslash \text{forall } x \backslash \text{exists } y ((x < y) \backslash \text{wedge} (y \text{ \textit{prím}})) \backslash \text{wedge} (Ssy \text{ \textit{prím}})) \leftrightarrow$   
 $) \$$   
Minden  $x$ , létezik olyan  $y$ , ahol  $x$  kisebb  $y$ -nél és  $y$  ikerprímszám

$\$ (\backslash \text{exists } y \backslash \text{forall } x (x \text{ \textit{prím}}) \backslash \text{supset } (x < y)) \$$   
Létezik  $y$ , minden olyan  $x$ -re, amikor  $x$  prímszám, ami abból következik, hogy  $\leftrightarrow$   
 $x$  kisebb  $y$ -nél

$\$ (\backslash \text{exists } y \backslash \text{forall } x (y < x) \backslash \text{supset } \backslash \text{neg } (x \text{ \textit{prím}})) \$$   
Létezik  $y$ , minden olyan  $x$ -re amikor  $y$  kisebb  $x$ -nél és ebből leszűrhető,  $\leftrightarrow$   
hogy  $x$  nem prímszám

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 típusú változót hozunk létre`
- `int *b = &a; //a pointer a-nak memóriacímét tárolja`
- `int &r = a; //r megkapja az a változó értékét`
- `int c[5]; //5 elemű tömb`
- `int (&tr)[5] = c; //a tömbben lévő elemeket átadjuk a tr-nek`
- `int *d[5]; //Tömbre mutató mutató`
- `int *h (); //egészre pointer pointer visszaadó függvény`

- ```
int *(*l) (); //Pointer pointer függvény pointer
```
- ```
int (*v (int c)) (int a, int b) //egy 2 egészet kapó függvényre pointer ↵  
pointer
```
- ```
int ((*z) (int)) (int, int); //EGy egészet visszaadó 2-őt kapó függvény ↵  
pointer pointer
```

Megoldás videó:

Megoldás forrása:

## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\* háromszögmátrix

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

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

Megoldás forrása: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/Caesar/tm.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c)

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

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

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

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

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

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

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

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

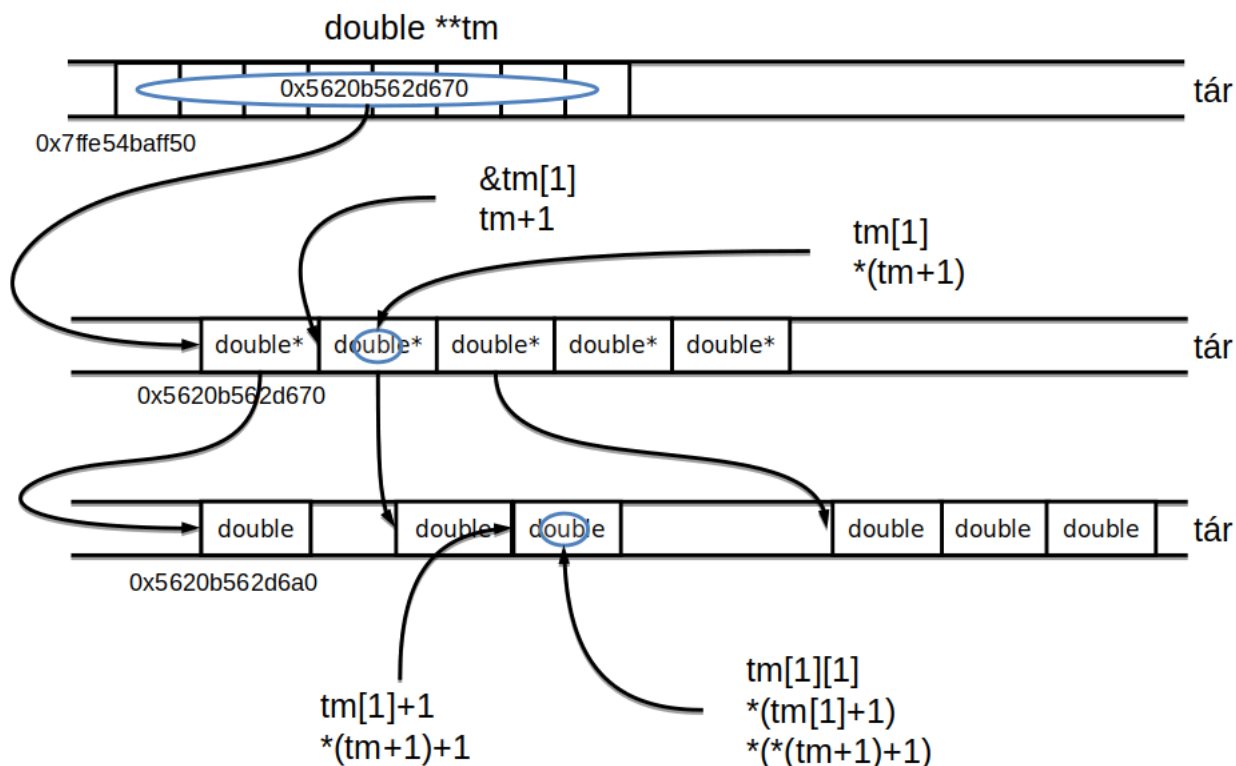
for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
}

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

free (tm);

return 0;
}
```





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

Létrehozunk egy változót amely a mátrixnak a sorrát adja meg ami jelen pillanatban 5. A `**tm` -el lefoglaljuk a memóriát, ami 8 bájt. Uátna a `nr` változónkhoz hozzá adjuk a `sizeof` függvénnnyel. Ha sikeres a memória foglalás akkor amennyi sort foglaltunk le annyi oszlopot fogunk kapni.

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define MAX_KULCS 100
#define BUFFER_MERET 256
int
main (int argc, char **argv)
{
    char kulcs[MAX_KULCS];
    char buffer[BUFFER_MERET];
    int kulcs_index = 0;
```

```
int olvasott_bajtok = 0;
int kulcs_meret = strlen (argv[1]);
strncpy (kulcs, argv[1], MAX_KULCS);
while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
{
    for (int i = 0; i < olvasott_bajtok; ++i)
    {
        buffer[i] = buffer[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
    write (1, buffer, olvasott_bajtok);
}
}
```

Tanulságok, tapasztalatok, magyarázat...

Használata: ./titkosító 56789012 tiszta.txt titkos.szoveg (A program neve, a kulcs, a titkosítani való szöveg és a titkosított szöveg. cat titkos.szoveg (Megnézük a titkos szöveget.

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

```
//Batfai kód
public class ExorTitkosító {

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

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBajtok = 0;

        while((olvasottBajtok =
            bejövőCsatorna.read(buffer)) != -1) {

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

                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;

            }

            kimenőCsatorna.write(buffer, 0, olvasottBajtok);
        }
    }
}
```

```
    }

}

public static void main(String[] args) {

    try {

        new ExorTitkosító(args[0], System.in, System.out);

    } catch (java.io.IOException e) {

        e.printStackTrace();

    }

}

}
```

Ugyan úgy működik mint a C-ben megírt program, adni kell neki egy kódot ami szerinte titkosít.

## 4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <string.h>
double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
    int sz = 0;
    for (int i = 0; i < titkos_meret; ++i)
        if (titkos[i] == ' ')
            ++sz;
    return (double) titkos_meret / sz;
}
int
tisztalehet (const char *titkos, int titkos_meret)
{

```

[illegible]

```
for (int pi = '0'; pi <= '9'; ++pi)
{
    kulcs[0] = ii;
    kulcs[1] = ji;
    kulcs[2] = ki;
    kulcs[3] = li;
    kulcs[4] = mi;
    kulcs[5] = ni;
    kulcs[6] = oi;
    kulcs[7] = pi;
    if (exor_tores (kulcs, KULCS_MERET, ←
        titkos, p - titkos))
        printf
            ("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta ←
             szoveg: [%s]\n",
             ii, ji, ki, li, mi, ni, oi, pi, ←
             titkos);
    // ujra EXOR-ozunk, így nem kell egy ←
    // masodik buffer
    exor (kulcs, KULCS_MERET, titkos, p - ←
        titkos);
}

return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

gcc t.c -o t -std=c99-vel fordítjuk és a program futtatásánál grep-el odaírjuk a kódot.

## 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)

A OR logikai műveletnél, mindig 1-et kapsz kivéve, ha mindkettő 0.

Az AND logikai műveletnél ha megegyezik a két szám, akkor megkapod ugyanazt a számot.

Az EXOR logikai műveletnél, ha megegyezik a két szám akkor 0-át kapsz, de ha az egyik 1-es akkor 1-est.

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

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

Ez a program 3 részre van bontva. Ahoz hogy futatni tudjuk szükségünk lesz a libpng++-dev, mert képekkel fogunk dolgozni.

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

### 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)

Tanulságok, tapasztalatok, magyarázat...

### 5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

### 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 forrása:

Megoldás videó:

Megoldás forrása:

## 5.6. Mandelbrot nagyító és utazó Java nyelven

DRAFT

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

Megoldás videó:

Megoldás forrása:

```
public class PolarGenerator {
    boolean nincsTarolt = true;
    double tarolt;
    public PolarGenerator() {
        nincsTarolt = true;
    }
    public double kovetkezo() {
        if(nincsTarolt) {
            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2*u1 - 1;
                v2 = 2*u2 - 1;
                w = v1*v1 + v2*v2;
            } while(w > 1);
            double r = Math.sqrt((-2*Math.log(w))/w);
            tarolt = r*v2;
            nincsTarolt = !nincsTarolt;
            return r*v1;
        } else {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }
    public static void main(String[] args) {
        PolarGenerator g = new PolarGenerator();
        for(int i=0; i<10; ++i)
```



```
        System.out.println(g.kovetkezo());  
    }  
}
```

A polárgenerátor két véletlenszerű érték generálására alkalmazható.

```
$ ubuntu@ubuntu:~/Desktop/Polargen$ Java Polargen  
2.40276  
-0.892715  
0.533079  
-0.0335743  
1.03269  
-0.0934786  
0.425988  
-2.02593  
0.291546  
0.375556
```

## 6.2. LZW

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

```
// z.c  
  
//  
  
// LZW fa építő  
  
// Programozó Páternoszter  
  
//  
  
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail ←  
// .com  
  
//  
  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
  
//  
  
// This program is distributed in the hope that it will be useful,
```

```
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1, http://progpater.blog.hu/2011/02/19/gyonyor\_a\_tomor
```

```
// 0.0.2, csomópontok mutatóinak NULLázása (nem fejtette meg senki :)

// 0.0.3, http://progpater.blog.hu/2011/03/05/ ↔
    labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat

//

    //>gcc z.c -lm -o z          fordítása

    //>

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

typedef struct binfa
{
    int ertek;

    struct binfa *bal_nulla;

    struct binfa *jobb_egy;

    //>itt definiáljuk a binfa típust
} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
```

```
BINFA_PTR p;

if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
{
    perror ("memoria");
    exit (EXIT_FAILURE);
}

return p;
}

extern void kiir (BINFA_PTR elem);
extern void ratlag (BINFA_PTR elem);
extern void rszoras (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);

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

    int egy_e;

    int i;

    unsigned char c;

    //>BinfaPTR== user által definiált típus

    BINFA_PTR gyoker = uj_elem ();

    gyoker->ertek = '/';
```

```
gyoker->bal_nulla = gyoker->jobb_egy = NULL;

BINFA_PTR fa = gyoker;

long max=0;

while (read (0, (void *) &b, sizeof(unsigned char)))

{

    for(i=0;i<8; ++i)

    {

        egy_e= b& 0x80;

        if ((egy_e >>7)==0)

            c='1';

        else

            c='0';

    }

//    write (1, &b, 1);

    if (c == '0')

    {

        if (fa->bal_nulla == NULL)

        {

            fa->bal_nulla = uj_elem ();

            fa->bal_nulla->ertek = 0;

            fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;

            fa = gyoker;

        }

        else

        {
```

```
        fa = fa->bal_nulla;

    }

}

else

{

    if (fa->jobb_egy == NULL)

    {

        fa->jobb_egy = uj_elem ();

        fa->jobb_egy->ertek = 1;

        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;

        fa = gyoker;

    }

    else

    {

        fa = fa->jobb_egy;

    }

}

}

printf ("\n");

kiir (gyoker);

extern int max_melyseg, atlagosszeg, melyseg, atlagdb;

extern double szorasosszeg, atlag;
```

```
printf ("melyseg=%d\n", max_melyseg - 1);

/* Átlagos ághossz kiszámítása */

atlagosszeg = 0;

melyseg = 0;

atlagdb = 0;

ratlag (gyoker);

// atlag = atlagosszeg / atlagdb;

// (int) / (int) "elromlik", ezért casoljuk

// K&R tudatlansági védelem miatt a sok () :)

atlag = ((double) atlagosszeg) / atlagdb;

/* Ághosszak szórásának kiszámítása */

atlagosszeg = 0;

melyseg = 0;

atlagdb = 0;

szorasosszeg = 0.0;

rszoras (gyoker);

double szoras = 0.0;

if (atlagdb - 1 > 0)

    szoras = sqrt (szorasosszeg / (atlagdb - 1));

else
```

```
    szoras = sqrt (szorasosszeg);

    printf ("atlag=%f\nszoras=%f\n", atlag, szoras);

    szabadit (gyoker);
}

// a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
// http://sourceforge.net/projects/javacska/
// az atlag() hivasakor is inicializalni kell oket, a
// a rekurziv bejaras hasznalja
int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

void
ratlag (BINFA_PTR fa)
{

    if (fa != NULL)
    {
        ++melyseg;

        ratlag (fa->jobb_egy);

        ratlag (fa->bal_nulla);

        --melyseg;
```



```
        if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)

        {

            ++atlagdb;

            atlagosszeg += melyseg;

        }

    }

}

// a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
// http://sourceforge.net/projects/javacska/
// az atlag() hivasakor is inicializalni kell oket, a
// a rekurziv bejaras hasznalja
double szorasosszeg = 0.0, atlag = 0.0;

void
rszoras (BINFA_PTR fa)
{

    if (fa != NULL)

    {

        ++melyseg;
```

```
        rszoras (fa->jobb_egy);

        rszoras (fa->bal_nulla);

        --melyseg;

        if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
        {

            ++atlagdb;

            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));

        }

    }

}

//static int melyseg = 0;

int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
```

```
    _melyseif (melyseg > maxg);

max_melyseg = melyseg;

    kiir (elem->jobb_egy);

    // ez a postorder bejáráshoz képest

    // 1-el nagyobb mélység, ezért -1

    for (int i = 0; i < melyseg; ++i)

printf ("---");

    printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↔
        ,

        melyseg - 1);

    kiir (elem->bal_nulla);

    --melyseg;

}

}

void

szabadit (BINFA_PTR elem)

{

    if (elem != NULL)

    {

        szabadit (elem->jobb_egy);

        szabadit (elem->bal_nulla);

        free (elem);

    }

}
```

Program futatása: ./z3a7 [bemenő fájl] -o [kimenő fájl]

## 6.3. Fabejárás

Preorder bejárás: 1. gyökér feldolgozás 2. bal oldal bejárása 3. jobb oldal bejárása

```
void kiir (Csomopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;

        for (int i = 0; i < melyseg; ++i)
            os << "---";
        kiir (elem->nullasGyermek (), os);
        kiir (elem->egyenesGyermek (), os);
        --melyseg;
    }
}
```

Inorder bejárás: 1. Baloldal bejárása 2. Gyökér feldolgozás 3. Jobb oldal bejárása

```
void kiir (Csomopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;

        kiir (elem->nullasGyermek (), os);
        for (int i = 0; i < melyseg; ++i)
            os << "---";

        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->egyenesGyermek (), os);
        --melyseg;
    }
}
```

Postorder bejárás: 1. Bal oldal bejárása 2. Jobb oldal bejárása 3. gyöklér feldolgozás

```
void kiir (Csomopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyenesGyermek (), os);
        kiir (elem->nullasGyermek (), os);
    }
}
```

```
        for (int i = 0; i < melyseg; ++i)
            os << "----";

    os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
    --melyseg;
}
}
```

## 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása: <https://github.com/Borbiro/Prog1/blob/master/z3a7.cpp>

## 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: <https://github.com/Borbiro/Prog1/blob/master/z3a7.cpp>

## 6.6. Mozgató szemantika

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

Megoldás videó:

```
// z3a7.cpp
//
// Együtt támadjuk meg: http://progpater.blog.hu/2011/04/14/ ↵
// együtt_tamadjuk_meg
// LZW fa építő 3. C++ átírata a C változatból (+mélység, átlag és szórás)
// Programozó Páternosztér
//
// Copyright (C) 2011, 2012, Bátfa Norbert, nbatfai@inf.unideb.hu, ↵
// nbatfai@gmail.com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
```

```
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1,      http://progpater.blog.hu/2011/02/19/gyonyor\_a\_tomor
// 0.0.2,      csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3,      http://progpater.blog.hu/2011/03/05/ ←
//              labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// 0.0.4,      z.cpp: a C verzióból svn: bevezetes/C/ziv/z.c átírjuk C++- ←
//              ra
//              http://progpater.blog.hu/2011/03/31/ ←
//              imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol
// 0.0.5,      z2.cpp: az fgv(*mut)-ok helyett fgv(&ref)
// 0.0.6,      z3.cpp: Csomopont beágyazva
//              http://progpater.blog.hu/2011/04/01/ ←
//              imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2
// 0.0.6.1     z3a2.c: LZWBinFa már nem barátja a Csomopont-nak, mert ←
//              annak tagjait nem használja direktben
// 0.0.6.2     Kis kommentezést teszünk bele 1. lépésként (hogy a kicsit ←
//              lemaradt hallgatóknak is
//              könnyebb legyen, jól megtűzdeljük további olvasmányokkal)
//              http://progpater.blog.hu/2011/04/14/egyutt\_tamadjuk\_meg
//              (majd a 2. lépésben "beletesszük a d.c-t", majd s 3. ←
//              lépésben a parancssor sor argok feldolgozását)
// 0.0.6.3     z3a2.c: Fejlesztgetjük a forrást: http://progpater.blog.hu ←
//              /2011/04/17/a_tizedik_tizenegyedik_labor
```

```
// 0.0.6.4      SVN-beli, http://www.inf.unideb.hu/~nbatfai/pl/forrasok-SVN ↵  
//             /bevezetes/vedes/  
// 0.0.6.5      2012.03.20, z3a4.cpp: N betűk (hiányok), sorvégek, vezető ↵  
//             komment figyelmen kívül: http://progpater.blog.hu/2012/03/20/ ↵  
//             a_vedes_elokeszítése  
// 0.0.6.6      z3a5.cpp: mamenyaka kolléga észrevételére a több komment ↵  
//             sor figyelmen kívül hagyása  
//             http://progpater.blog.hu/2012/03/20/a\_vedes\_elokeszítése/ ↵  
//             fullcommentlist/1#c16150365  
// 0.0.6.7      Javaslom ezt a verziót választani védendő programnak  
// 0.0.6.8      z3a7.cpp: pár kisebb javítás, illetve a védések támogatásához ↵  
//             további komment a <<  
//             eltoló operátort tagfüggvényként, illetve globális függvényként ↵  
//             túlterhelő részekhez.  
//             http://progpater.blog.hu/2012/04/10/ ↵  
//             imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_4/fullcommentlist/1# ↵  
//             c16341099  
//  
  
#include <iostream>    // mert olvassuk a std::cin, írjuk a std::cout ↵  
//             csatornákat  
#include <cmath>       // mert vonunk gyököt a szóráshoz: std::sqrt  
#include <fstream>     // fájlból olvasunk, írunk majd  
#include <vector>  
  
/* Az LZWBinFa osztályban absztraháljuk az LZW algoritmus bináris fa ↵  
//             építését. Az osztály  
//             definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ↵  
//             ez lesz a  
//             beágyazott Csomopont osztály. Miért ágyazzuk be? Mert külön nem szánunk ↵  
//             neki szerepet, ezzel  
//             is jelezzük, hogy csak a fa részeként számolunk vele.*/  
  
class LZWBinFa  
{  
public:  
    /* Szemben a bináris keresőfánkkal (BinFa osztály)  
    http://progpater.blog.hu/2011/04/12/ ↵  
    imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3  
    itt (LZWBinFa osztály) a fa gyökere nem pointer, hanem a '/' betűt ↵  
    tartalmazó objektum,  
    lásd majd a védett tagok között lent: Csomopont gyoker;  
    A fa viszont már pointer, mindig az épülő LZW-fánk azon csomópontjára ↵  
    mutat, amit az  
    input feldolgozása során az LZW algoritmus logikája diktál:  
    http://progpater.blog.hu/2011/02/19/gyonyor\_a\_tomor  
    Ez a konstruktor annyit csinál, hogy a fa mutatót ráállítja a gyökérre ↵  
    . (Mert ugye  
    laboron, blogon, előadásban tisztáztuk, hogy a tartalmazott tagok, ↵  
    most "Csomopont gyoker"
```

```
konstruktora előbb lefut, mint a tagot tartalmazó LZWBinFa osztály ←
konstruktora, éppen a
következő, azaz a fa=&gyoker OK.)
*/
LZWBinFa ():fa ()
{
    gyoker=new Csomopont('/');
    fa=gyoker;
}
~LZWBinFa ()
{
    szabadit(gyoker);
}

LZWBinFa (LZWBinFa&& masik){
    gyoker=nullptr;
    *this= std::move(masik);
}

LZWBinFa& operator= (LZWBinFa&& masik){
    std::swap(gyoker,masik.gyoker);
    return *this;
}

/* Tagfüggvényként túlterheljük a << operátort, ezzel a célunk, hogy ←
felkeltsük a
hallgató érdeklődését, mert ekkor így nyomhatjuk a fába az inputot: ←
binFa << b; ahol a b
egy '0' vagy '1'-es betű.
Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett ←
paraméterként"
kapott ) példány, azaz annak a fának amibe éppen be akarjuk nyomni a b ←
betűt a tagjai
(pl.: "fa", "gyoker") használhatóak a függvényben.

A függvénybe programoztuk az LZW fa építésének algoritmusát tk.:
http://progpater.blog.hu/2011/02/19/gyonyor\_a\_tomor

a b formális param az a betű, amit éppen be kell nyomni a fába.

a binFa << b (ahol a b majd a végén látszik, hogy már az '1' vagy a ←
'0') azt jelenti
tagfüggvényként, hogy binFa.operator<<(b) (globálisként így festene: ←
operator<<(binFa, b) )

*/
void operator<< (char b)
{
```



```
// Mit kell betenni éppen, '0'-t?
if (b == '0')
{
    /* Van '0'-s gyermeke az aktuális csomópontnak?
    megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
    if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
    {
        // elkészítjük, azaz példányosítunk a '0' betű akt. ←
        parammal
        Csomopont *uj = new Csomopont ('0');
        // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
        // jegyezze már be magának, hogy nullás gyereke mostantól ←
        van
        // küldjük is Neki a gyerek címét:
        fa->ujNullasGyermek (uj);
        // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
        fa = gyoker;
    }
    else // ha van, arra rálépünk
    {
        // azaz a "fa" pointer már majd a szóban forgó gyermekre ←
        mutat:
        fa = fa->nullasGyermek ();
    }
}
// Mit kell betenni éppen, vagy '1'-et?
else
{
    if (!fa->egyenesGyermek ())
    {
        Csomopont *uj = new Csomopont ('1');
        fa->ujEgyenesGyermek (uj);
        fa = gyoker;
    }
    else
    {
        fa = fa->egyenesGyermek ();
    }
}
}
/* A bejárással kapcsolatos függvényeink (túlterhelt kiir-ók, atlag, ←
    ratlag stb.) rekurzívak,
    tk. a rekurzív fabejarást valósítják meg (lásd a 3. előadás "Fabejárás ←
    " c. fóliáját és társait)

    (Ha a rekurzív függvénnyel általában gondod van => K&R könyv megfelel ←
    ő része: a 3. ea. izometrikus
    részében ezt "letáncoltuk" :) és külön idéztük a K&R álláspontját :)
    */
void kiir (void)
```

```
{
    // Sokkal elegánsabb lenne (és más, a bevezetésben nem kibontandó ←
    // reentráns kérdések miatt is, mert
    // ugye ha most két helyről hívják meg az objektum ilyen ←
    // függvényeit, tehát ha kétszer kezd futni az
    // objektum kiir() fgv.-e pl., az komoly hiba, mert elromlana a ←
    // mélység... tehát a mostani megoldásunk
    // nem reentráns) ha nem használnánk a C verzióban globális ←
    // változókat, a C++ változatban példánytagot a
    // mélység kezelésére: http://progpater.blog.hu/2011/03/05/ ←
    // there_is_no_spoon
    melyseg = 0;
    // ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát, ←
    // akkor a
    // sztenderd out-ra nyomjuk
    kiir (gyoker, std::cout);
}
/* már nem használjuk, tartalmát a dtor hívja
void szabadit (void)
{
    szabadit (gyoker.egyGyermek ());
    szabadit (gyoker.nullasGyermek ());
    // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a ←
    // szabad tárban (halmon).
}
*/

/* A változatosság kedvéért ezeket az osztálydefiníció (class LZWBinFa ←
{...};) után definiáljuk,
hogy kénytelen légy az LZWBinFa és a :: hatókör operátorral minősítve ←
definiálni :) l. lentebb */
int getMelyseg (void);
double getAtlag (void);
double getSzoras (void);

/* Vágyunk, hogy a felépített LZW fát ki tudjuk nyomni ilyenformán: std ←
::cout << binFa;
de mivel a << operátor is a sztenderd névtérben van, de a using ←
namespace std-t elvből
nem használjuk bevezető kurzusban, így ez a konstrukció csak az ←
argfüggő névfeloldás miatt
fordul le (B&L könyv 185. o. teteje) ám itt nem az a lényeg, hanem, ←
hogy a cout ostream
osztálybeli, így abban az osztályban kéne módosítani, hogy tudjon ←
kiírni LZWBinFa osztálybelieket...
e helyett a globális << operátort terheljük túl,

a kiFile << binFa azt jelenti, hogy

- tagfüggvényként: kiFile.operator<<(binFa) de ehhez a kiFile ←
```

valamilyen  
std::ostream stream osztály forrásába kellene beleírni ezt a ↵  
tagfüggvényt,  
amely ismeri a mi LZW binfánkat...

- globális függvényként: operator<<(kiFile, binFa) és pont ez látszik ↵  
a következő sorban:

```
*/  
friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)  
{  
    bf.kiir (os);  
    return os;  
}  
void kiir (std::ostream & os)  
{  
    melyseg = 0;  
    kiir (gyoker, os);  
}
```

private:

```
class Csomopont  
{  
public:  
    /* A paraméter nélküli konstruktor az elepértelmezett '/' "gyökér- ↵  
       betűvel" hozza  
       létre a csomópontot, illet hívunk a fából, aki tagként tartalmazza a ↵  
       gyökeret.  
       Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu" ↵  
       tagba, a két  
       gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is ↵  
       megteszi. */  
    Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0)  
    {  
    };  
    ~Csomopont ()  
    {  
    };  
    // Aktuális csomópont, mondd meg nékem, ki a bal oldali gyermeked  
    // (a C verzió logikájával műxik ez is: ha nincs, akkor a null megy ↵  
    vissza)  
    Csomopont *nullasGyermekek () const  
    {  
        return balNulla;  
    }  
    // Aktuális csomópont, mondd meg nékem, ki a jobb oldali gyermeked?  
    Csomopont *egyGyermekek () const  
    {  
        return jobbEgy;  
    }  
}
```

```
// Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a bal ←
    oldali gyereked!
void ujNullasGyermek (Csomopont * gy)
{
    balNulla = gy;
}
// Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a jobb ←
    oldali gyereked!
void ujEgyesGyermek (Csomopont * gy)
{
    jobbEgy = gy;
}
// Aktuális csomópont: Te milyen betűt hordozol?
// (a const kulcsszóval jelezzük, hogy nem bántjuk a példányt)
char getBetu () const
{
    return betu;
}

private:
    // friend class LZWBinFa; /* mert ebben a változatban az LZWBinFa ←
        metódusai nem közvetlenül
    // a Csomopont tagjaival dolgoznak, hanem beállító/lekérdező ←
        üzenetekkel érik el azokat */

    // Milyen betűt hordoz a csomópont
    char betu;
    // Melyik másik csomópont a bal oldali gyermeke? (a C változatból " ←
        örökölt" logika:
    // ha hincs ilyen csermek, akkor balNulla == null) igaz
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    // nem másolható a csomópont (ökörszabály: ha van valamilye a ←
        szabad tárban,
    // letiltjuk a másoló konstruktort, meg a másoló értékadást)
    Csomopont (const Csomopont &); //másoló konstruktor
    Csomopont & operator= (const Csomopont &);
};

/* Mindig a fa "LZW algoritmus logikája szerinti aktuális" ←
    csomópontjára mutat */
Csomopont *fa;
// technikai
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
// szokásosan: nocopyable
LZWBinFa (const LZWBinFa &);
LZWBinFa & operator= (const LZWBinFa &);

/* Kiírja a csomópontot az os csatornára. A rekurzió kapcsán lásd a ←
```

```
korábbi K&R-es utalást... */
void kiir (Csomopont * elem, std::ostream & os)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    // leállítása
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyenesGyermek (), os);
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl; ←
        kiir (elem->nullasGyermek (), os);
        --melyseg;
    }
}

void szabadit (Csomopont * elem)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    // leállítása
    if (elem != NULL)
    {
        szabadit (elem->egyenesGyermek ());
        szabadit (elem->nullasGyermek ());
        // ha a csomópont mindkét gyermekét felszabadítottuk
        // azután szabadítjuk magát a csomópontot:
        delete elem;
    }
}

protected:    // ha esetleg egyszer majd kiterjesztjük az osztályt, mert
    // akarunk benne valami újdonságot csinálni, vagy meglévő tevékenységet ←
    // máshogy... stb.
    // akkor ezek látszanak majd a gyerek osztályban is

/* A fában tagként benne van egy csomópont, ez erősen ki van tüntetve, ←
   Ő a gyökér: */
Csomopont* gyoker;
int maxMelyseg;
double atlag, szoras;

void rmelyseg (Csomopont * elem);
void ratlag (Csomopont * elem);
void rszoras (Csomopont * elem);

};
```

```
// Néhány függvényt az osztálydefiníció után definiálunk, hogy lássunk ←
//   ilyet is ... :)
// Nem erőltetjük viszont a külön fájlba szedést, mert a ←
//   sablonosztályosított tovább
// fejlesztésben az linkelési gondot okozna, de ez a téma már kivezet a ←
//   laborteljesítés
// szükséges feladatából: http://progpater.blog.hu/2011/04/12/ ←
//   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3

// Egyébként a melyseg, atlag és szoras fgv.-ek a kiir fgv.-el teljesen egy ←
//   kaptafa.

int
LZWBinFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (gyoker);
    return maxMelyseg - 1;
}

double
LZWBinFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
    return atlag;
}

double
LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (gyoker);

    if (atlagdb - 1 > 0)
        szoras = std::sqrt (szorasosszeg / (atlagdb - 1));
    else
        szoras = std::sqrt (szorasosszeg);

    return szoras;
}

void
LZWBinFa::rmelyseg (Csomopont * elem)
{
    if (elem != NULL)
```

```
{
    ++melyseg;
    if (melyseg > maxMelyseg)
        maxMelyseg = melyseg;
    rmelyseg (elem->egyenesGyermekek ());
    // ez a postorder bejáráshoz képest
    // 1-el nagyobb mélység, ezért -1
    rmelyseg (elem->nullasGyermekek ());
    --melyseg;
}
}

void
LZWBinFa::ratlag (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyenesGyermekek ());
        ratlag (elem->nullasGyermekek ());
        --melyseg;
        if (elem->egyenesGyermekek () == NULL && elem->nullasGyermekek () == NULL ↔
            )
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

void
LZWBinFa::rszoras (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyenesGyermekek ());
        rszoras (elem->nullasGyermekek ());
        --melyseg;
        if (elem->egyenesGyermekek () == NULL && elem->nullasGyermekek () == NULL ↔
            )
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

// teszt pl.: http://progpater.blog.hu/2011/03/05/ ↔
labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
```

```
// [norbi@sgu ~]$ echo "01111001001001000111" | ./z3a2
// -----1(3)
// -----1(2)
// -----1(1)
// -----0(2)
// -----0(3)
// -----0(4)
// ---/(0)
// -----1(2)
// -----0(1)
// -----0(2)
// depth = 4
// mean = 2.75
// var = 0.957427
// a laborvédelemhez majd ezt a tesztelést használjuk:
// http://

/* Ez volt eddig a main, de most komplexebb kell, mert explicite bejövő, ←
   kimenő fájlokkal kell dolgozni
int
main ()
{
    char b;
    LZWBinFa binFa;

    while (std::cin >> b)
    {
        binFa << b;
    }

    //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ←
    verziókban de, hogy izgalmasabb legyen
    // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

    std::cout << binFa; // ehhez kell a globális operator<< túlterhelése, ←
    lásd fentebb

    std::cout << "depth = " << binFa.getMelyseg () << std::endl;
    std::cout << "mean = " << binFa.getAtlag () << std::endl;
    std::cout << "var = " << binFa.getSzoras () << std::endl;

    binFa.szabadit ();

    return 0;
}
*/

/* A parancssor arg. kezelést egyszerűen bedolgozzuk a 2. hullám kapcsolódó ←
   feladatából:
http://progpater.blog.hu/2011/03/12/hey\_mikey\_he\_likes\_it\_ready\_for\_more\_3
```



```
de mivel nekünk sokkal egyszerűbb is elég, alig hagyunk meg belőle valamit ↵
    ...
*/

void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    // http://progpater.blog.hu/2011/03/12/ ↵
    // hey_mikey_he_likes_it_ready_for_more_3
    // alapján a parancssor argok ottani elegáns feldolgozásából kb. ennyi ↵
    // marad:
    // "*(++argv)+1"...

    // a kiírás szerint ./lzwtree in_file -o out_file alakra kell mennie, ↵
    // ez 4 db arg:
    if (argc != 4)
    {
        // ha nem annyit kapott a program, akkor felhomályosítjuk erről a ↵
        // jüzettr:
        usage ();
        // és jelezzük az operációs rendszer felé, hogy valami gáz volt...
        return -1;
    }

    // "Megjegyezzük" a bemenő fájl nevét
    char *inFile = ++argv;

    // a -o kapcsoló jön?
    if (*(++argv) + 1) != 'o')
    {
        usage ();
        return -2;
    }

    // ha igen, akkor az 5. előadásból kimásoljuk a fájlkezelés C++ ↵
    // változatát:
    std::fstream beFile (inFile, std::ios_base::in);

    // fejlesztgetjük a forrást: http://progpater.blog.hu/2011/04/17/ ↵
    // a_tizedik_tizenegyedik_labor
    if (!beFile)
    {
        std::cout << inFile << " nem letezik..." << std::endl;
        usage ();
    }
}
```

```
        return -3;
    }

    std::fstream kiFile (*++argv, std::ios_base::out);

    unsigned char b;    // ide olvassik majd a bejövő fájl bájtjait
    LZWBinFa binFa;    // s nyomjuk majd be az LZW fa objektumunkba

    // a bemenetet binárisan olvassuk, de a kimenő fájl már karakteresen ←
    // írjuk, hogy meg tudjuk
    // majd nézni... :) l. az említett 5. ea. C -> C++ gyökkeltes átírási ←
    // példáit

    while (beFile.read ((char *) &b, sizeof (unsigned char)))
        if (b == 0x0a)
            break;

    bool kommentben = false;

    while (beFile.read ((char *) &b, sizeof (unsigned char)))
    {

        if (b == 0x3e)
        {
            // > karakter
            kommentben = true;
            continue;
        }

        if (b == 0x0a)
        {
            // újsor
            kommentben = false;
            continue;
        }

        if (kommentben)
            continue;

        if (b == 0x4e)    // N betű
            continue;

        // egyszerűen a korábbi d.c kódját bemásoljuk
        // laboron többször lerajzoltuk ezt a bit-tologatást:
        // a b-ben lévő bájt bitjeit egyenként megnézzük
        for (int i = 0; i < 8; ++i)
        {
            // maszkolunk eddig..., most már simán írjuk az if fejébe a ←
            // legmagasabb helyiértékű bit vizsgálatát
            // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ←
            // megmondja melyik:
            if (b & 0x80)
```

```
        // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW ↵
        fa objektumunkba
        binFa << '1';
    else
        // különben meg a '0' betűt:
        binFa << '0';
    b <<= 1;
}

}

//std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ↵
// verziókban de, hogy izgalmasabb legyen
// a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:


kiFile << binFa;    // ehhez kell a globális operator<< túlterhelése, ↵
// lásd fentebb
// (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, ↵
// mégis megy, hurrá)


kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;


kiFile.close ();
beFile.close ();


return 0;
}
```

Létrehozunk egy új csomópontot, amit egyenlővé teszünk a gyökérel és lenullázuk azt. Felszabadítjuk a gyökért. Az `std::move` függvénnyel mozgatjuk a fát és `*this`-el vissza adja a fát.

## 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: <https://github.com/Borbiro/Prog1/tree/master/Hangya>

Tanulságok, tapasztalatok, magyarázat: 3 osztályt hozunk létre: Ant, Antwin, Anthread. Az Antba a hangyának a tulajdonságait adjuk meg, például hogy merre tartson. Azt Antwin-be az ablak szélességét, magasságát határozzuk meg, a hangyák színét. Az Anthread nagyból ugyan olyan tulajdonságokkal rendelkezik mint a Antwin. A program futtatásához a qmake-et kell használnunk, amivel tudjuk szüneteltetni a program futtását a "P" lenyomásával, illetve a "Q" lenyomásával kiléphetünk belőle.

### 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: <https://github.com/Borbiro/Prog1/blob/master/etjatek.java>

A John Horton Conway-féle életjáték egy nagyon egyszerű játék, van egy élettér ami négyzetrácsos és minden négyzetbe egy sejt élhet, születhet vagy halhat meg. Ha 3 vagy kevesebb sejt van körülötte akkor a sejt életben marad, ha 3-nál több akkor meghal és ha az üres mezőnél a szomszédjai kötött pontosan 3 él akkor ott születni fog egy sejt. A szimuláció úgy indul, hogy megadjuk a kiindulási pontját. Fontos a java fordító használata.

### 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/Borbiro/Prog1/tree/master/elejtatek>

A program szinte ugyanugy mukodik mint a java esetében lényeges eltérés csak a fordításnálfedezhető fel.  
~ A qmake - procejt parancsal a .pro faljt generalja le . Ezt futtatva generaljuk le a makefile-t es a make parancsal fordítjuk le a programot .qmake - project | qmake x.y.pro | make | ./xy.

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: <https://github.com/Borbiro/Prog1/tree/master/BrainB>

A program az esportolok tesztelés miatt jött létre.

## 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://progater.blog.hu/2016/11/13/hello\\_samu\\_a\\_tensorflow-bol](https://progater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol)

Ezt a programot Python-al írták meg. Ehez a program működéséhez kell nekünk telepíteni a tesnorflow-ot. A tensorflow egy arc felismerő program. Amint feltelepítettük tele lesz az adatbázis képpel amivel elemezni fogja a többi képet és különböző egységekbe rakja őket.

### 8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása: [https://github.com/Borbiro/Prog1/blob/master/minst\\_deep.py](https://github.com/Borbiro/Prog1/blob/master/minst_deep.py)

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

Iteratív faktoriális

```
(define (fact n)
  (do ((i 1 (+ 1 i))
      (number 1 (* i number)))
    ((> i n) number)))
```

Inkrementálni kell i-t 1-gyel, utána a number értékét 1-re állítjuk, majd megszorozzuk i-vel. Ha az  $i > n$ , akkor visszakapjuk a number-t.

A rekurzív-nál először definiálni kell a faktoriális és utána vizsgálni meg a n kisebb 1-et és összeszorozom az n-1 faktoriálisát n-nel.

Megoldás videó:

Megoldás forrása:

### 9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása: [http://www.universelle-automation.de/1966\\_Boston.pdf](http://www.universelle-automation.de/1966_Boston.pdf)

Adatbázisban több szó és kifejezés található és ezekhez válasz-sablonok vannak. Ha a beszélgető válaszában megtalálható, az adatbázisban szereplő kulcsszó akkor az algoritmus egyszerűen kiválasztja random az adatbázisból egy elemet.

### 9.3. 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)

## 9.4. 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

[?]

A programozásnak 3 féle szintjét különböztetjük meg: 1, gépi nyelv 2, Assembly nyelv 3, magas szintű nyelv. A magasszintű nyelven megírt programmot forrásprogramnak vagy forrásszövegnek hívjuk. A magas szintű programozási nyelvekbe tartozik például a C,C++,Java,Python stb.. A program írás közben figyelembe kell venni a szintaktikai és szemantikai szabályokat. Ezeket a programokat gépire kell írni, így erre használjuk a fordítóprogramokat és az interpreteseket. A fordítóprogramok a következőket veszik figyelembe: lexikális elemzés,szintaktikai elemzés,szemantikai elemzés,kódgenerálás. A futó programot az operációs rendszer felügyeli. Az interpretes nem készít tárgykódot, hanem lefordítja és egyből le is futattja a programmot. Minden programnyelvnek meg van a saját szabványa, amit hivatkozási nyelvnek hívunk. A implemetációk egymással és a hivatkozási nyelvekkel sem kompatibilisek a mai napig. Napjainban a programozáshoz fejlesztői környezetek állnak rendelkezésünkre. A program nyelveket osztályoz, imperatív, deklaratív és más nyelvek között. Az adattípusokat 3 dolog határoz meg: 1, tartomány 2, műveletek 3, reprezentáció. Az adattípusok tartománya azokat az elemeknek a konkrét értékeit határozza meg. Az adattípushoz hozzátartoznak azok a műveletek, amelyeket a tartomány elemein végre tudunk hajtani. A reprezentáció a tartományok értékének megjelenítésért felel. Az adattípusoknak van két nagy csoportja: 1, A skalár vagy az egyszerű 2, A strukturált vagy az összetett A nevesített konstansoknak 3 komponense van név,típus, érték. A változó olyan programozási eszköz, amelynek négy komponense van név, attribútumok, cím, érték. A C-ben sok típus rendszer van például egész:int,long int, short int vagy betű:char.

### 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

A kifejezések szintaktikai eszközök. Arra valók, hogy a program egy adott pontján ott már ismert értékekből új értéket határozzunk meg. A kifejezések formálisan 3 összetevőből áll: operandusok, operátorok, kerek zárójelek. A kifejezésnek három alakja lehet attól függően, hogy kétoperandusú operátorok esetén az operandusok és az operátor sorrendje milyen. A lehetséges esetek: -prefix, az operátor az operandusok előtt áll. -infix, az operátor az operandusok között áll. -postfix, az operátor az operandusok mögött áll. Azt a kifejezést, amelynek értéke fordítási időben eldől, amelynek kiértékelését a fordító végzi konstans kifejezésnek hívjuk. Operandusai literálok és nevesített konstansok lehetnek.

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

## 10.3. Programozás

[BMECPP]

Az egyik legnépszerűbb és legnehezebb magas szintű programozási nyelv a C++. Emellett még ott a C ami szinte ugyan az, csak nem lehet változókat túlterhelni vagy cout helyett printf-el kell kiírni vagy C-ben nincsenek még osztályok. Elég sok minden van a C++-ban már ami a C-ben állom például használhatunk operátorokat amit C-ben függvények helyettesítenek.

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

## 11.5. C++

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

## 11.6. Lisp

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

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

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