

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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	Tóth, Balázs	2019. március 19.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun tétel	12
2.8. A Monty Hall probléma	13
3. Helló, Chomsky!	15
3.1. Decimálisból unárisba átváltó Turing gép	15
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	15
3.3. Hivatkozási nyelv	15
3.4. Saját lexikális elemző	16
3.5. Leetspeak	17
3.6. A források olvasása	19
3.7. Logikus	20
3.8. Deklaráció	21

4. Helló, Caesar!	25
4.1. double ** háromszögmátrix	25
4.2. C EXOR titkosító	28
4.3. Java EXOR titkosító	29
4.4. C EXOR törő	30
4.5. Neurális OR, AND és EXOR kapu	36
4.6. Hiba-visszaterjesztéses perceptron	36
5. Helló, Mandelbrot!	37
5.1. A Mandelbrot halmaz	37
5.2. A Mandelbrot halmaz a std::complex osztállyal	38
5.3. Biomorfok	40
5.4. A Mandelbrot halmaz CUDA megvalósítása	44
5.5. Mandelbrot nagyító és utazó C++ nyelven	44
5.6. Mandelbrot nagyító és utazó Java nyelven	45
6. Helló, Welch!	46
6.1. Első osztályom	46
6.2. LZW	46
6.3. Fabejárás	46
6.4. Tag a gyökér	46
6.5. Mutató a gyökér	47
6.6. Mozgató szemantika	47
7. Helló, Conway!	48
7.1. Hangyaszimulációk	48
7.2. Java életjáték	48
7.3. Qt C++ életjáték	48
7.4. BrainB Benchmark	49
8. Helló, Schwarzenegger!	50
8.1. Szoftmax Py MNIST	50
8.2. Szoftmax R MNIST	50
8.3. Mély MNIST	50
8.4. Deep dream	50
8.5. Robotpszichológia	51

9. Helló, Chaitin!	52
9.1. Iteratív és rekurzív faktoriális Lisp-ben	52
9.2. Weizenbaum Eliza programja	52
9.3. Gimp Scheme Script-fu: króm effekt	52
9.4. Gimp Scheme Script-fu: név mandala	52
9.5. Lambda	53
9.6. Omega	53
 III. Második felvonás	 54
10. Helló, Arroway!	56
10.1. A BPP algoritmus Java megvalósítása	56
10.2. Java osztályok a Pi-ben	56
 IV. Irodalomjegyzék	 57
10.3. Általános	58
10.4. C	58
10.5. C++	58
10.6. Lisp	58

Ábrák jegyzéke

4.1. A <code>double **</code> háromszögmátrix a memóriában	27
5.1. A Mandelbrot halmaz a komplex síkon	37

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?

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?

1.2. Milyen doksikat olvassak el?

- Olvasgasd 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
- [[KERNIGHANRITCHIE](#)]
- [[BMECPP](#)]
- Az igazi kockák persze csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: <https://github.com/ghjbku/DE/blob/master/video.flv>

Megoldás forrása:

Elsőként a 100%-os végtelen ciklust készítettem el, hiszen ezt volt a legegyszerűbb megírni. Amint láthatjuk elég egyszerűen meg lehet oldani, hogy a cpu 100%-ban dolgozzon a program futása alatt. Itt én a WHILE ciklus-t választottam, de FOR-ral is hasonlóképpen lehet megvalósítani a végtelenítést. Az egész program lényege egyetlen értéken alapszik, amit az `*asd*` változó hordoz. Mivel ez a változó semmiképp sem kap 1-et értékül, a program soha sem fog kilépni a ciklusból.

```
//100%-ban megdolgoztat egy magot
//lefordítás: gcc forrásnév -o késznév
#include <stdio.h>
int main()
{
    int asd =0;
    while (asd=1) {}
    return 0;
}
```

A következő program a 0%-os végtelen ciklus volt. Ha ismerjük az API-t, vagy tapraesettek vagyunk a google-n való keresést illetően, akkor itt is egyszerű dolgunk volt. Amint azt észrevehettük, a programkód nagyon hasonlít az előző kódra, csupán annyi változás történt, hogy a ciklus belsejében megjelent egy függvény, a `*sleep()*`. Ez a függvény annyi milisecond-ig állítja meg a programot, amely számot a két zárójel közé írtunk. Jelen esetben ez `*1*`, de mivel egy végtelen ciklusban vagyunk, ezért végtelen sokszor vár majd 1 milisec-ot a program, így tehát nem használ erőforrást.

```
//0%-os cpu használat
//lefordítás: gcc forrásnév -o késznév
#include <stdio.h>
```

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

Utolsóként pedig jön a "legnehezebb", minden magot 100%-on futtatni. Az igazat megvallva, ez sem valami nagy ördögösség, itt is csak egy pár dolog változott a legelső programhoz képest. A legfontosabb dolog ez a sor `*#include "omp.h"`, ez a header fájl előfeltétele annak, hogy a `*#pragma omp parallel*` kódot értelmezni tudja a fordítóprogram. A `*#pragma...*` sor veszi rá a programunkat, hogy párhuzamos módon, az összes magon futtassa a programot a számítógép.

```
//minden mag 100%-on fut
//lefordítás: gcc -fopenmp forrásnév -o késznév
#include <stdio.h>
#include <unistd.h>
#include "omp.h"
int main () {
int asd=0;

#pragma omp parallel
while (asd=1)
{
}
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Amint láthattuk, elég egyszerű dolgunk volt ezen programok megírását illetően, viszont ez nem azt jelenti, hogy félvállról vehetjük a programozást, hiszen kevés olyan program létezik, aminek valamilyen hétköznapi haszna van, és mégis ilyen egyszerű lenne megírni. Ezen programkódok csak az egyszerűbb megértést segítik elő, gyakorlati hasznuk sajnos nincs.

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...

Amint láthatjuk, lehetetlen olyan programot írni, amely egy másik programról eldöntené, hogy az le fog-e fagyni, vagy sem.

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

Megoldás forrása:

Ezen feladat megoldása igencsak egyszerűnek bizonyult. Na persze nem annyira egyszerű, mint egy végtelen ciklus megírása, de közel azonos szinten mozognak. A lentebb lévő forráskód elég egyszerűen értelmezhető, ezért hát nem megyek bele részletesen, csak a nagyon fontos dolgokat mondom el. A *C* nyelvben a változók értékét egy paranccsal tudjuk hozzáfűzni egy printf függvényhez, attól függően, hogy milyen típusú adatot hordoz a változó. Esetünkben mindkét változó *szám/Digit* típust hordoz, ezért a kód, amivel meghívjuk a behelyettesítő paramétert, ez lesz: *%d*, majd ha végeztünk a kiírni kívánt szöveggel, egy vesszővel jelezzük a fordítóprogramnak, hogy most a behelyettesítendő változók következnek. A kódban megjelenik egy másik kód is, ami ismeretlen lehet az olvasó számára, ez a *\n*, amely annyit tesz, hogy új sorba kezdi az *\n* után beírt szöveget, és a szóközt is értelmezi!

```
#include <stdio.h>
int main()
{
    int a=5,b=3;
    printf("A value = %d\n",a);
    printf("B value = %d\n",b);
    b=b-a;
    a=a+b;
```

```
b=a-b;
printf("A value = %d\n",a);
printf("B value = %d\n",b);
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Ne tévesszen meg bennünket a feladat komplexitása, ha nem gondolunk bele, hogy pontosan hogyan is kellene segédváltozó nélkül elérni céljainkat, elég sokáig el tudunk időzni ezen az egyszerű feladaton. Tehát próbáljunk meg minden feladatot úgy kezdeni, hogy elgondolkozunk azon, hogyan tudnánk megvalósítani a feladatban megírtakat.

2.4. Labdapattogás

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

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

Megoldás forrása:

A feladat nehézségi szintjét tekintve már egy magassabb szinten van, vagyis inkább, gondolkodást igényel. Ebben a forráskódban már megjelenik egy pár új parancs, melyeket eddig még nem láttunk. Kezdve az új Header fájlal, a <math.h> fájlal, amely a matematikai függvényekért felel és minden értéket double típussal kezel(double típust kér, és azt ad vissza), ilyen függvény például az *abs*, amely az abszolút értéket jelöli, de ebben a header fájlban található a *pow* és az *sqrt* is, az előbbi a hatványozást, míg utóbbi a négyzetgyököt kezeli. Aztán ott van az a furcsa sor két sorral alább, az a bizonyos *#define*... ezeket a sorokat úgynevezett "Nevesített konstansok" definiálásánál használjuk. Ezek a konstansok értéket nem változtatnak a program futása során, és bármilyen értéket adhatunk nekik.

```
//Labdapattogás if nélkül (mentorálva Gila Attila által)
#include<stdio.h>
#include<math.h>

#define szel 80
#define mag 24

int putX(x,y)
{
    int ix,iy;

    for(ix=0;ix<x;ix++)
        printf("\n");

    for(iy=0;iy<y;iy++)
        printf(" ");

    printf("O\n");
}
```

```
return 0;
}

int main()
{
    long int x=0,y=0;

    while(1)
    {
        system("clear");
        putX(abs(mag-(x++%(mag*2))),abs(szel-(y++%(szel*2))));
        usleep(15000);
    }

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A labda "pattogása" egyszerű módon van megoldva, miszerint minden egyes "tick" után, amit a program a végtelen ciklusban tölt, a `*system("clear")*` parancs miatt a terminál jelenlegi tartalma törlődik, de mivel az túl gyorsan történik, mi csak úgy érezzük, hogy a labda szépen mozog az ablakban. a "tick" periódust az `*usleep()*` függvény zárójelében megadott szám határozza meg, a mértékegység microsecond. Viszont ha fontos a pontosság, akkor számolnunk kell a számítógép kalkulációs képességeivel, plusz az is időbe telik, hogy a program eljut az `*usleep*` függvényhez, ezután az egész program "alvó" állapotba kerül, kilép a processor ütemezési sorából, és a delay attól is függhet, hogy a processor maga mikor válassza újra a programot, miután a `*usleep*` függvény lefutott. Tehát ne lepődjünk meg, ha néhány ezer microsec-ot téved a program.

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:

Megoldás forrása: <https://github.com/ghjbku/DE/blob/cpp/bitshift.cpp>

A feladat megoldása C++ nyelven történt, viszont C-ben is hasonló módon kell megoldani a problémát.

```
//a bitshift C nyelvben
#include <stdio.h>

int main(){

    unsigned int the_Bit = 1;
    int length = 0;

    do
```

```
length++;  
while((the_Bit <= 1));  
  
printf("A szóhossz mérete: %u\n", length);  
  
return 0;  
}
```

A C megoldás BogoMIPS-el: [itt található](#)

Tanulságok, tapasztalatok, magyarázat...

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása C++-ban: <https://github.com/ghjbku/DE/blob/cpp/bearazas.cpp> és
C-ben: <https://github.com/ghjbku/DE/blob/master/c%20cuccok/bearaz.c>

```
#include <stdio.h>  
#include <math.h>  
  
void  
kiir (double tomb[], int db)  
{  
    int i;  
    for (i=0; i<db; i++)  
        printf("PageRank [%d]: %lf\n", i, tomb[i]);  
}  
  
double tavolsag(double pagerank[], double pagerank_temp[], int db)  
{  
    double tav = 0.0;  
    int i;  
    for (i=0; i<db; i++)  
        tav += abs(pagerank[i] - pagerank_temp[i]);  
    return tav;  
}  
  
int main(void)  
{  
    double L[4][4] = {  
        {0.0, 0.0, 1.0 / 3.0, 0.0},  
        {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},  
        {0.0, 1.0 / 2.0, 0.0, 0.0},  
        {0.0, 0.0, 1.0 / 3.0, 0.0}  
    };  
};
```

```
double PR[4] = {0.0, 0.0, 0.0, 0.0};
double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};

long int i,j,h;
i=0; j=0; h=5;

for (;;)
{
for(i=0;i<4;i++)
PR[i] = PRv[i];
for (i=0;i<4;i++)
{
double temp=0;
for (j=0;j<4;j++)
temp+=L[i][j]*PR[j];
PRv[i]=temp;
}

if ( tavolsag(PR,PRv, 4) < 0.00001)
break;
}
kiir (PR,4);
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A pagerank

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

```
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")
```

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

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

```
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

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

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

  }else{

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

  }

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

}

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

for (i in 1:kiserletek_szama) {

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

}

valtoztatesnyer = which(kiserlet==valtoztat)

sprintf("Kiserletek szama: %i", kiserletek_szama)
```

```
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Tanulságok, tapasztalatok, magyarázat...

DRAFT

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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>

int main()
{
    int inline asdfunc(int a)
```

```
{
    int b = a*a;
    a=b*a;
    return a;
}

asdfunc(5);
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

<https://github.com/ghjbku/DE/blob/master/c%20cuccok/Bildschirmfoto%20von%202019-03-12%2011-35-07.png> Amint láthatjuk, a funkció minden gond nélkül lefordul C99-ben, míg C89-ben hibát észlel. A hiba az `*inline*` parancs miatt van. ez a `cmd` a C99-el jött be, amely általában `*extern inline*` -al párban jelenik meg egy programkódban. Segítségével egy program nagyobb sebességre képes, viszont ezért cserébe nagyobb a helyigénye.

3.4. Saját lexikális elemző

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

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

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

```
//lex fájl
//fordítás c-re : lex -o output.c lexfajl.l (szükséges hozzá a flex ↔)
//jelen esetben: lex -o realnumbers.c realnumbers.l

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

a C-re fordított program ha készen van a fordítás, akkor mondhatjuk a gcc-nek, hogy csináljon nekünk futtatható programot a c forrásból. Ezt a következő sor beírásával tehetjük meg: `* gcc realnumber.c -o realnumber -lfl *`

Tanulságok, tapasztalatok, magyarázat...

Ebben a feladatban kicsit eltértünk a megszokott dolgoktól, ugyanis nem mi írtuk meg a C forráskódot, hanem a lexer. Ez nagyban megkönnyíti a dolgunkat, hiszen ha megnézzük a C forrást, amit a lex elkészített helyettünk, láthatjuk, hogy nem éppen egy rövid kis kódsorozatról van szó. A lexer segítségével nekünk már csak annyi a dolgunk, hogy megmondjuk neki, milyen típust keressen az inputban "digit[0-9]", és hogyan ismerje azt fel " {digit}*(\.{digit}+)? "

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

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

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

```
%{
//a fordítás megegyezik az előző feladatével: lex -o output.c lexfájl.1
//majd gcc output.c -o output
#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', {"1", "1", "|", "|_"}},
    {'m', {"m", "44", "(V", "\\|\\|"}},
    {'n', {"n", "\\|\\|", "/\\|/", "/V"}},
```

```
{'o', {"0", "0", "()", "[]"}},
{'p', {"p", "/o", "|D", "|o"}},
{'q', {"q", "9", "O_", "(,)"}}},
{'r', {"r", "12", "12", "|2"}},
{'s', {"s", "5", "$", "$"}},
{'t', {"t", "7", "7", "'|'"}},
{'u', {"u", "|_|", "(_)", "[_]"}},
{'v', {"v", "\\\/", "\\\/", "\\\/"}},
{'w', {"w", "VV", "\\\/\\\/", "(\/\\\/)"}},
{'x', {"x", "%", ")(", ")(")}},
{'y', {"y", "", "", ""}},
{'z', {"z", "2", "7_", ">_"}},
```

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

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

```
%}
```

```
%%
```

```
. {
```

```
int found = 0;
for(int i=0; i<L337SIZE; ++i)
{
    if(l337d1c7[i].c == tolower(*yytext))
    {

        int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

        if(r<91)
            printf("%s", l337d1c7[i].leet[0]);
        else if(r<95)
            printf("%s", l337d1c7[i].leet[1]);
        else if(r<98)
            printf("%s", l337d1c7[i].leet[2]);
        else
            printf("%s", l337d1c7[i].leet[3]);

        found = 1;
    }
}
```

```
        break;
    }

}

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

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

Tanulságok, tapasztalatok, magyarázat...

Ez a program a terminálról beolvassa a karaktereket, és randomizálva 4 különböző karaktert rak az eredeti helyére. Ha számára ismeretlen karaktert írunk be, akkor visszaadja ugyan azt. A program magja egy l337d1c7 tömb, amely tárolja a helyettesítési értékeket minden karakterhez. Ha nem talál egyetlen karaktert sem az inputban, akkor nem ír ki semmit.

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezeslo)==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 jelkezeslo függvény kezelje. (Miótan a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)



Bugok

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

i.

```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezeslo);
```

Ha a **SIGINT** jelzés nem volt ignorálva, akkor ignorálja.

ii.

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

egy for ciklus, amely 0-tól 5-ig tart, tehát 5x fut le, és minden lefutás után inkrementálja az i értéket 1-el, majd az inkrementálási értéket adja vissza.

iii.

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

ez is egy for ciklus, viszont ebben az esetben az i inkrementálása után az eredeti, növelés előtti értéket adja vissza.

iv.

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

Ebben a for ciklusban nem csak, hogy növeljük az i értékét minden kör után, de ezen értéket behelyezük egy tömbbe is.

v.

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

a ciklus 0-tól indul, és addig megy, amíg i kisebb mint n , továbbá a d pointer növelt értéke megegyezik az s pointer növelt értékével.

vi.

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

kiír két függvényt, melyek számokat adnak vissza, viszont a visszatérési érték precedenciát sugall.

vii.

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

visszaad kettő számot, melyekből az egyik egy függvény visszatérési értéke

viii.

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

ugyanúgy két számot ad vissza, de az f függvényben valami történik.

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

  
egy szám típusú **a** változót
- ```
int *b = &a;
```


egy szám típusú **b** **pointert**, aminek az értéke **a** **memóriában foglalt helye**
- ```
int &r = a;
```

  
integer típusú **r** **értéke a** lesz
- ```
int c[5];
```


létrehoz egy 5 számnak helyet adó **c** tömböt
- ```
int (&tr)[5] = c;
```

  
létrehoz egy **tr** **tömböt**, melynek az értéke **c**
- ```
int *d[5];
```


egy **d** **tömbre mutató pointer**

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

### **h funkcióra mutató pointer**

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

egy pointer, ami az **l** függvényre mutató pointerre mutat

- ```
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](#), [bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr2.c](#).

```
#include <stdio.h>

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

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

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

int
main ()
{
    int (*f) (int, int);
```



```
f = sum;

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

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

g = sumormul;

f = *g (42);

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

return 0;
}
```

```
#include <stdio.h>

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

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

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

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

int
main ()
{

    F f = sum;

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

    G g = sumormul;
```

```
f = *g (42);  
  
printf ("%d\n", f (2, 3));  
  
return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

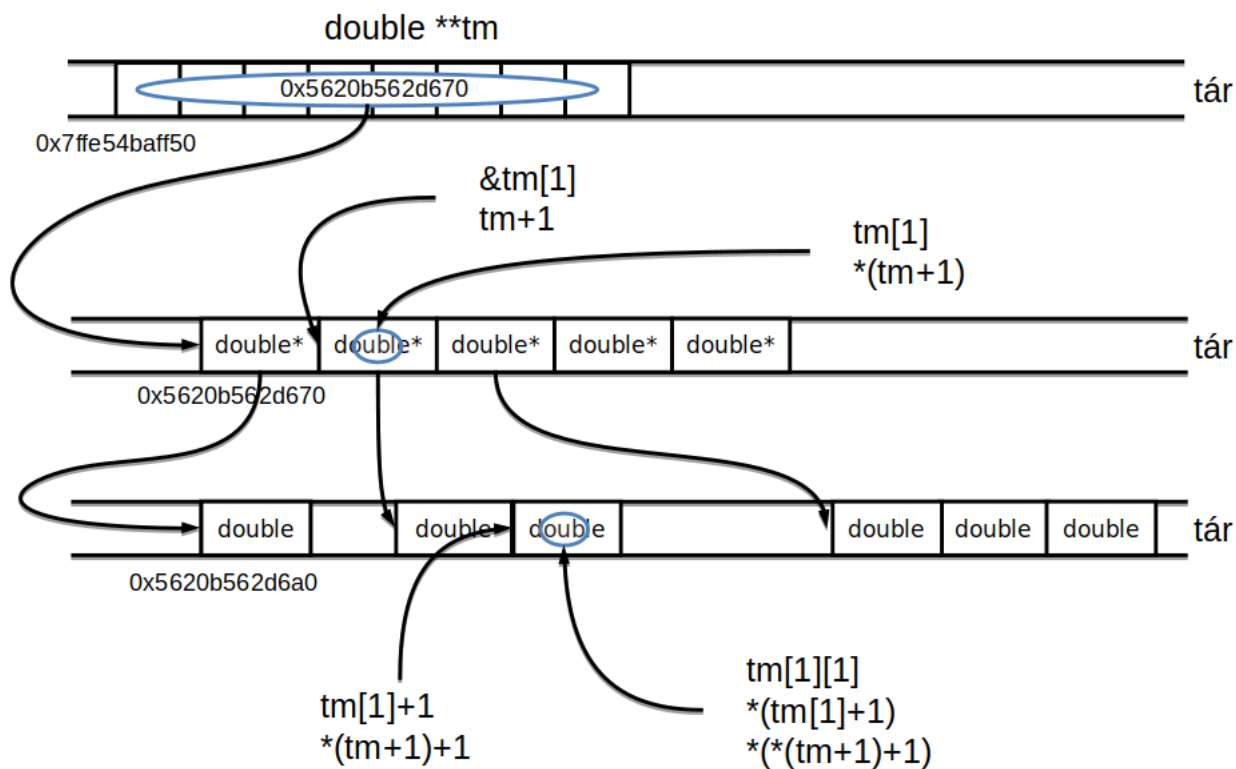
for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
}
//ezek a sorok átírják az előbb feltöltött tm változó értékeit a ←
//követezőkre:
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

//majd újra kiírja a program az egész tömböt, immár a megváltoztatott ←
//értékekkel
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...

Az

```
int nr = 5;
```

sorral létrehozunk egy integer változót, amelynek az értéke 5, ez a változó lesz a háromszög sorainak száma.

A

```
double **tm;
```

sor pedig létrehoz egy double típusú változót, ez lesz később a programunk magja.

Ezek a sorok:

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

pedig megnézik, hogy a programunk le tud e foglalni `nr*8` bájtot, ha nem, akkor kilép a programból a `-1` visszatérési értékkel.

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

```
{  
    return -1;  
}
```

Itt azt láthatjuk, hogy a program a `tm` változót tömbként kezelve bejárja azt, és mindig $(i+1)*8$ bájtot allokál/foglal le az aktuális tömb pozíciójához. Ha ez nem sikerül, akkor megint csak kilép **-1**-es visszatérési értékkel.

```
for (int i = 0; i < nr; ++i)  
for (int j = 0; j < i + 1; ++j)  
    tm[i][j] = i * (i + 1) / 2 + j; 0 1  
  
for (int i = 0; i < nr; ++i)  
{  
    for (int j = 0; j < i + 1; ++j)  
        printf ("%f, ", tm[i][j]);  
    printf ("\n");  
}
```

Ezek a ciklusok a kiíratásért felelnek. **az első egybeágyazott for ciklus pár** a `tm` változót tölti fel számokkal 0-tól 15-ig, majd **a második cikluspár** kiíratja azokat

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

Ezek a sorok a program végén felszabadítják a lefoglalt memóriahelyeket, először a változó tömbjeleimeitől kezdve, majd végül a most már üres változót is letörli.

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...

Amint láthatjuk, a main() függvényben megjelent két ismeretlen paraméter. ezek a program futtatásánál játszanak szerepet: az **argc** jelöli az argumentumok számát, beleértve a **./programnév** sort is. ezzel szemben a ****argv** egy vektor, amely az argumentumokat tárolja. Itt például, ha a terminálba ezt a sort írjuk: **./fájlnév 1234 -o output.txt**, akkor az argc értéke 4 lesz, míg a **argv vektor így néz ki: **<./fájlnév; 1234; -o; output.txt>**

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

Ez a két sor is ismeretlen lehet számunkra, de nem kell tőlük megijedni, elég egyszerű a kezelésük. az első sor az argumentum_vektor 1. elemét(ami az előző példában az 1234 volt) lekéri, és megszámolja annak hosszát(**ez az strlen() függvény dolga**), majd a kulcs_meret nevű változónak ezt a hosszt értékül adja. a második sor pedig egy string másoló függvény, ennek szintaktikája a következő: **strncpy(char cél_változó,char másolni_kívánt_érték,a másolni_kívánt_érték_hossza)** (ha a másolt érték kisebb, mint az utolsó paraméterben megadott szám, akkor a maradékot NULL bájtokkal fogja kipótolni a program)

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

```
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 olvasottBájtok = 0;

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

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

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

    }

    kimenőCsatorna.write(buffer, 0, olvasottBájtok);

}

public static void main(String[] args) {

    try {

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

    } catch(java.io.IOException e) {

        e.printStackTrace();

    }

}
```

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:

```
#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)
{
    // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

void
xor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{
    int kulcs_index = 0;

    for (int i = 0; i < titkos_meret; ++i)
    {
        titkos[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}

int
```



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

és a többi magos változat:

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256  
#define KULCS_MERET 8  
#define _GNU_SOURCE  
  
#include <stdio.h>  
#include <unistd.h>  
#include <string.h>  
#include <stdlib.h>  
  
double  
atlagos_szohossz (const char *titkos, int titkos_meret)  
{  
    int sz = 0;  
    for (int i = 0; i < titkos_meret; ++i)  
        if (titkos[i] == ' ')  
            ++sz;  
  
    return (double) titkos_meret / sz;  
}  
  
int  
tiszta_lehet (const char *titkos, int titkos_meret)  
{  
    // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat  
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a  
    // potenciális töréseket  
  
    double szohossz = atlagos_szohossz (titkos, titkos_meret);  
  
    return szohossz > 6.0 && szohossz < 9.0  
        && strcasestr (titkos, "hogya") && strcasestr (titkos, "nem")
```

```
        && strstr (titkos, "az") && strstr (titkos, "ha");
    }

void
xor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret, ↵
    char *buffer)
{
    int kulcs_index = 0;

    for (int i = 0; i < titkos_meret; ++i)
    {
        buffer[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}

void
xor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{
    char *buffer;

    if ((buffer = (char *)malloc(sizeof(char)*titkos_meret)) == NULL)
    {
        printf("Memoria (buffer) falióra\n");
        exit(-1);
    }

    xor (kulcs, kulcs_meret, titkos, titkos_meret, buffer);

    if (tisza_lehet (buffer, titkos_meret))
    {
        printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTisza szoveg: [%s]\n",
            kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],kulcs ↵
            [6],kulcs[7], buffer);
    }

    free(buffer);
}

int
main (void)
{
```

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

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

// maradek hely nullazasa a titkos bufferben
for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';

// osszes kulcs eloallitasa
#pragma omp parallel for private(kulcs)
for (int ii = '0'; ii <= '9'; ++ii)
    for (int ji = '0'; ji <= '9'; ++ji)
        for (int ki = '0'; ki <= '9'; ++ki)
            for (int li = '0'; li <= '9'; ++li)
                for (int mi = '0'; mi <= '9'; ++mi)
                    for (int ni = '0'; ni <= '9'; ++ni)
                        for (int oi = '0'; oi <= '9'; ++oi)
                            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;

                                    exor_tores (kulcs, KULCS_MERET, titkos, ←
                                                p - titkos);
                                }

return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

az `atlagos_szohossz` függvény bekéri a titkos szöveget és méretét, majd egy `'sz'` változóban megszámolja, hogy hány szóköz található a szövegben. Ezután a szöveg teljes méretét elosztja az `'sz'` változó értékével,

így megkapva az átlagos szóhosszt.

az strcasestr függvény azt keresi, hogy a titkos-ban megtalálható-e a **2. paraméterben megadott szöveg** eg.: "hogya" és "nem"

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

A forráskód hossza miatt nem tenném bele a könyvbe, de kódsnippet-eket fogok használni.

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++](#) 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 [bhaxor/attention-raising/Mandelbrot/3.1.2.cpp](https://github.com/bhaxor/attention-raising-Mandelbrot) nevű állománya.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ↵
-0.01947381057309366392260585598705802112818 ↵
-0.0194738105725413418456426484226540196687 ↵
0.7985057569338268601555341774655971676111 ↵
0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ↵
0.4127655418209589255340574709407519549131 ↵
0.4127655418245818053080142817634623497725 ↵
0.2135387051768746491386963270997512154281 ↵
0.2135387051804975289126531379224616102874
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --line-numbers=1 --left-footer=" ↵
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```



```
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double a = -1.9;
    double b = 0.7;
    double c = -1.3;
    double d = 1.3;

    if ( argc == 9 )
    {
        szelesseg = atoi ( argv[2] );
        magassag =  atoi ( argv[3] );
        iteraciosHatar =  atoi ( argv[4] );
        a = atof ( argv[5] );
        b = atof ( argv[6] );
        c = atof ( argv[7] );
        d = atof ( argv[8] );
    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↵
        " << std::endl;
        return -1;
    }

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

    double dx = ( b - a ) / szelesseg;
    double dy = ( d - c ) / magassag;
    double reC, imC, reZ, imZ;
    int iteracio = 0;

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

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

```
{
    // k megy az oszlopokon

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

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

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

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

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

            ++iteracio;
        }

        kep.set_pixel ( k, j,
                        png::rgb_pixel ( iteracio%255, (iteracio*iteracio <=
                        )%255, 0 ) );
    }

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

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

5.3. Biomorfok

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

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

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

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

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

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

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

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

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

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

            ++iteracio;
        }
    }
}
```

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

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

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

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ←
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ←
color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ←
Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

    int szelesseg = 1920;
    int magassag = 1080;
```

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

if ( argc == 12 )
{
    szelesseg = atoi ( argv[2] );
    magassag = atoi ( argv[3] );
    iteraciosHatar = atoi ( argv[4] );
    xmin = atof ( argv[5] );
    xmax = atof ( argv[6] );
    ymin = atof ( argv[7] );
    ymax = atof ( argv[8] );
    reC = atof ( argv[9] );
    imC = atof ( argv[10] );
    R = atof ( argv[11] );
}
else
{
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↵
        d reC imC R" << std::endl;
    return -1;
}

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

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

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

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

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

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

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

```
int iteracio = 0;
for (int i=0; i < iteraciosHatar; ++i)
{
    z_n = std::pow(z_n, 3) + cc;
    //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
    if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
    {
        iteracio = i;
        break;
    }
}

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

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

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

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: [bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu](https://bhax.attention_raising/CUDA/mandelpngc_60x60_100.cu) nevű állománya.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJssY>, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a

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

DRAFT

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

DRAFT

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

10. fejezet

Helló, Arroway!

10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. 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

10.3. Általános

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

10.4. C

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

10.5. C++

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

10.6. Lisp

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

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

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