

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	Bátfai, Norbert	2019. március 10.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	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?	7
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	13
2.8. A Monty Hall probléma	13
3. Helló, Chomsky!	16
3.1. Decimálisból unárisba átváltó Turing gép	16
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	16
3.3. Hivatkozási nyelv	16
3.4. Saját lexikális elemző	18
3.5. l33t.1	19
3.6. A források olvasása	19
3.7. Logikus	20
3.8. Deklaráció	20

4. Helló, Caesar!	22
4.1. int *** háromszögmátrix	22
4.2. C EXOR titkosító	22
4.3. Java EXOR titkosító	22
4.4. C EXOR törő	22
4.5. Neurális OR, AND és EXOR kapu	23
4.6. Hiba-visszaterjesztéses perceptron	23
5. Helló, Mandelbrot!	24
5.1. A Mandelbrot halmaz	24
5.2. A Mandelbrot halmaz a std::complex osztállyal	24
5.3. Biomorfok	24
5.4. A Mandelbrot halmaz CUDA megvalósítása	24
5.5. Mandelbrot nagyító és utazó C++ nyelven	24
5.6. Mandelbrot nagyító és utazó Java nyelven	25
6. Helló, Welch!	26
6.1. Első osztályom	26
6.2. LZW	26
6.3. Fabejárás	26
6.4. Tag a gyökér	26
6.5. Mutató a gyökér	27
6.6. Mozgató szemantika	27
7. Helló, Conway!	28
7.1. Hangyaszimulációk	28
7.2. Java életjáték	28
7.3. Qt C++ életjáték	28
7.4. BrainB Benchmark	29
8. Helló, Schwarzenegger!	30
8.1. Szoftmax Py MNIST	30
8.2. Szoftmax R MNIST	30
8.3. Mély MNIST	30
8.4. Deep dream	30
8.5. Robotpszichológia	31

9. Helló, Chaitin!	32
9.1. Iteratív és rekurzív faktoriális Lisp-ben	32
9.2. Weizenbaum Eliza programja	32
9.3. Gimp Scheme Script-fu: króm effekt	32
9.4. Gimp Scheme Script-fu: név mandala	32
9.5. Lambda	33
9.6. Omega	33
 III. Második felvonás	 34
10. Helló, Arroway!	36
10.1. A BPP algoritmus Java megvalósítása	36
10.2. Java osztályok a Pi-ben	36
 IV. Irodalomjegyzék	 37
10.3. Általános	38
10.4. C	38
10.5. C++	38
10.6. Lisp	38

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:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/turing/inf_sleep.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/turing/inf_sleep.c)

[bhax/thematic_tutorials/bhax_textbook/turing/infinite_single.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/turing/infinite_single.c)

[bhax/thematic_tutorials/bhax_textbook/turing/infinite_multi.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook/turing/infinite_multi.c)

A végtelen ciklusok nagyon hasznosak tudnak lenni például ha egy menüt akarunk írni ahol semmi mást nem csinálunk, csak várunk valamilyen bemenetre és az alapján végrehajtunk valamit, majd újra várakozunk. C-ben 2 féle módon írhatunk végtelen ciklust:

- for ciklussal

```
for(;;) {  
    //infinite loop  
}
```

- while ciklussal

```
while(1) {  
    //infinite loop  
}
```

Először nézzük meg a processzort 100%-on futtató programot. A könnyebb olvashatóság érdekében a while ciklust használtam és az "1"-es értékkel (amit a fordító logikai igaz értékként értelmez ebben az esetben) a ciklus soha nem ér véget. Mivel a ciklusba nem írtunk semmit, a program a while ciklus feltételének vizsgálásával tölti minden idejét, ami a processzor 100% leterheltségét eredményezi.



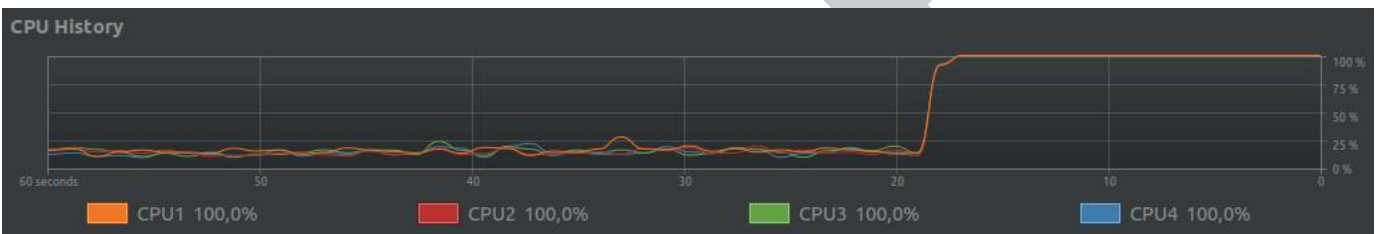
Ahhoz hogy az összes magot 100%-on pörgessük, csak 1 sorral kell kibővítenünk a programot, de ehhez szükségünk lesz az `omp.h` könyvtár implementálására és fordításkor a `"-fopenmp"` kapcsolóra.

```
#include <omp.h>

int main() {

    #pragma omp parallel
    while(1) {}

    return 0;
}
```



Ha 0%-os terhelést akarunk, akkor csökkenteni kell a feltételvizsgálat gyakoriságát. Vagyis a `sleep(seconds)` paranccsal várakozásra utasítjuk a processzort.

```
#include <unistd.h>

int main() {

    while(1) sleep(1);

    return 0;
}
```

1

[|||||]

10.0%

Tasks: 169, 690 thr; 1 running

2

[|||||]

14.1%

Load average: 0.49 0.68 0.72

3

[|||||]

10.6%

Uptime: 07:15:23

4

[|||||]

9.9%

Mem

[|||||]

3.77G/7.70G

Swp

[|||||]

0K/3.81G

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1070	gdm	20	0	111M	2724	12	S	0.0	0.0	0:00.00	(sd-pam)
1408	my	20	0	111M	2728	12	S	0.0	0.0	0:00.00	(sd-pam)
25849	my	20	0	4376	740	680	S	0.0	0.0	0:00.00	./inf sleep
11075	my	20	0	31388	5068	3568	S	0.0	0.1	0:00.02	/bin/bash
1069	gdm	20	0	77136	8316	6780	S	0.0	0.1	0:00.08	/lib/systemd/systemd --user

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 épülő `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.

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: [bhax/thematic_tutorials/bhax_textbook/turing/valtcserere.c](https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk)

Változók értékének felcserélésére leggyakrabban a rendezési algoritmusoknál van szükség. Ilyenkor a legegyszerűbb módszer jut az eszembe, miszerint veszek egy harmadik, úgymond segédváltozót, amiben ideiglenesen eltárolom az egyik változó értékét, hogy azt majd a másik változó kaphassa meg.

```
int c = a;
a = b;
b = c;
```

Ha segédváltozó nélkül szeretnénk megoldani a problémát, akkor kénytelenek vagyunk valamilyen műveleteket alkalmazni. Nem kell sokáig keresgélni, mivel az összeadás és kivonás tökéletesen megfelel erre a feladatra. Ehhez először összeadjuk őket az a változóba, majd abból kivonva a b-t megkapjuk először a b, majd az a értékét, ahogy azt a programkód szemlélteti:

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

2.4. Labdapattogás

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

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

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/turing/labdaIf.c](https://bhaxor.blog.hu/2018/08/28/labdapattogas) [bhax/thematic_tutorials/bhax_textbook/turing/labdaIfNelkul.c](https://bhaxor.blog.hu/2018/08/28/labdapattogas)

Ez a feladat sokkal egyszerűbb, mint az ember gondolná, legalábbis ha használhatunk if-eket. Csak annyit kell csinálni, hogy ciklusonként növeljük a labda pozícióját és ha elérjük az ablak kereteit, megfordítjuk a labda irányát. Ezután már csak meg kell jeleníteni a labdát.

```
#include <stdio.h>  
  
#define width 80  
#define height 24  
  
void draw(int x, int y) {  
    for(int i = 0; i < y; i++) printf("\n");  
    for(int k = 0; k < x; k++) printf(" ");  
    printf("@\n");  
}  
  
int main() {  
    int x = 1;  
    int y = 1;  
    int a = 1;  
    int b = 1;  
  
    while(1) {  
        system("clear");  
  
        if(x >= width || x <= 0) a *= -1;  
        if(y >= height || y <= 0) b *= -1;  
  
        draw(x, y);  
        x += a;  
        y += b;
```

```
    usleep(15000);  
  
}  
  
return 0;  
}
```

If-ek nélkül azonban már kicsit nehezebb a dolgunk, ugyanis egy kicsit gondolkodni kell. De ekkor se kell kétségbe esni, mert ugyebár a matematika a "barátunk" és mindig nyújt számunkra egy bonyolultnak látszó képletet, hogy azt egyszer bepötyögve örökre megszabaduljunk tőle. Most mi is ezt tesszük és még egy magyarázatot is csatolok hozzá.

```
draw(abs(width-(x++%(width*2))), abs(height-(y++%(height*2))));
```

Tegyük fel hogy az `x` értéke 1 és a `width` értéke pedig 5. Ha az 1-et elosztjuk maradékos osztással a `width` kétszeresével, akkor 1-et kapunk, amit kivonunk a szélességből. Ennek eredményeként a labda pattogása az ablak jobb alsó sarkából indul.

```
x = 1; width = 5;  
|5 - ( 1 % ( 5 * 2 ) )| =  
|5 - ( 1 % 10 )| =  
|5 - 1| = 4
```

Ha az `x` eléri a 6-ot, az abszolútérték miatt a végeredmény növekedni kezd, egészen amíg el nem éri a `width` kétszeresét, amikor is újra csökkenni kezd és ez így megy körbe-körbe. A `++` az `x` után a számolások után megnöveli 1-el az `x`-et.

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: [bhax/thematic-tutorials/bhax-textbook/turing/bogomips.c](https://bhax.thematic-tutorials.com/bhax-textbook/turing/bogomips.c)

Mi is ez a Bogomips valójában? Megméri, hogy 1 másodperc alatt hányszor fut le a programunk egy ciklusa. Sokan abba a hibába esnek, hogy ez alapján hasonlítják össze a gépüket, hogy kié gyorsabb. Egyedül a linux bootolásakor érdemes odafigyelni rá, hogy a processzor és a cache rendben van-e.

A `ticks` változóba elmentjük a jelenlegi processzoridőt, majd a `for` ciklus egy bizonyos számú lefutása után a jelenlegi processzoridőből kivonjuk a `ticks` értékét, így megkapjuk hogy mennyi ideig tartott a ciklus `x`-szeri lefutása.

```
while ((loops_per_sec <= 1))  
{  
    ticks = clock ();  
    delay (loops_per_sec);  
    ticks = clock () - ticks;
```

```
...  
}
```

Ha ez az idő kisebb mint 1 másodperc, akkor legközelebb többször futtatjuk le a for ciklust, ellenben viszont némi átalakításokkal kiírjuk a gépünk BogoMips értékét.

```
Calibrating delay loop..ok - 690.00 BogoMIPS
```

A BogoMips while ciklus fejét használva mostmár könnyen megkapjuk szóhosszt. Konkrétan itt az történik, hogy az `int` típusú változó elején levő bitet ciklusonként eltoljuk és hozzáadunk egyet a szóhosszt tároló `length` változóhoz.

```
int x = 1;  
int length = 0;  
  
do  
    length++;  
while((x <= 1));
```

Az én gépemen a szóhossz mérete 32.

```
$ ./szohossz  
A szóhossz mérete: 32
```

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: [bhax/thematic_tutorials/bhax_textbook/turing/helloGoogle.c](https://bhax.thematic_tutorials.com/bhax_textbook/turing/helloGoogle.c)

Most megvizsgáljuk a PageRank algoritmust, ami hatalmas fellendülést hozott a Google-nek.

Egy oldal PageRankját az határozza meg, hogy mennyi oldal hivatkozik rá. Mivel ez így magába nem lesz elég, mert az algoritmus azt is figyelembe veszi, hogy mekkora a PageRankjuk a rá hivatkozó oldalaknak. Például ha a Facebook hivatkozik a te oldaladra, akkor az sokkal többet jelent, mintha csak valami ismeretlen oldal hivatkozna rá, még akkor is ha több van belőlük.

A program elején létrehozunk egy kétdimenziós tömböt, ami meghatározza, hogy melyik mire hivatkozik. Ezután egy végtelen ciklusba bizonyos számításokkal addig finomítgatjuk az oldalak PR értékét, amíg már nincs értelme tovább számolni.

```
#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 += (pagerank_temp[i] - pagerank[i]) * (pagerank_temp[i] - pagerank[i] ←
    );
return sqrt(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;
}
```

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó: <https://youtu.be/xbYhp9G6VqQ>

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/turing/brun.r

```
# Copyright (C) 2019 Dr. Norbert Bاتفai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>

library(matlab)

stp <- function(x){

  primes = primes(x)
  diff = primes[2:length(primes)]-primes[1:length(primes)-1]
  idx = which(diff==2)
  t1primes = primes[idx]
  t2primes = primes[idx]+2
  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: bhax/thematic_tutorials/bhax_textbook/turing/monty.r

Tanulságok, tapasztalatok, magyarázat...

```
# An illustration written in R for the Monty Hall Problem
# Copyright (C) 2019 Dr. Norbert Bاتفai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
# https://bhaxor.blog.hu/2019/01/03/ ←
# erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan
#

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)
```


3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

Mutass be legalább két környezetfüggő generatív grammatikát, amely ezt a nyelvet generálja!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

A [KERNIGHANRITCHIE] könyv C referencia-kézikönyv/Utasítások melléklete alapján definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

Megoldás videó:

Megoldás forrása: [bhax/thematic_tutorials/bhax_textbook/chomsky/hivat_k.c](https://bham.ac.uk/~bham/bhax/thematic_tutorials/bhax_textbook/chomsky/hivat_k.c)

Először nézzük meg mi is az a BNF. A BNF (Backus-Naur Form) egy olyan nyelv, ami más nyelvek helyesírását írja le. Nézzünk meg ráegy egyszerű példát:

```
<egész szám> ::= <előjel><szám>
<előjel> ::= [-|+]
<szám> ::= <számjegy>{<számjegy>}
<számjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

Ez alapján egy c utasítás BNF-ben így fog kinézni:

```
<statement> ::= <labeled_statement>
               | <compound_statement>
               | <expression_statement>
               | <selection_statement>
               | <iteration_statement>
               | <jump_statement>

<labeled_statement> ::= <identifier> : <statement>
                       | case <constant_expression> : <statement>
                       | default : <statement>

<compound_statement> ::= { <declaration_list> }
                       | { <statement_list> }
                       | { <declaration_list> <statement_list> }

<declaration_list> ::= <declaration>
                      | <declaration_list> <declaration>

<statement_list> ::= <statement>
                   | <statement_list> <statement>

<expression_statement> ::= <expression>

<selection_statement> ::= if ( <expression> ) <statement>
                        | if ( <expression> ) <statement> else <statement>
                        | switch ( <expression> ) <statement>

<iteration_statement> ::= while ( <expression> ) <statement>
                        | do <statement> while ( <expression> ) ;
                        | for ( <expression_statement> <↵
                          expression_statement> ) <statement>
                        | for ( <expression_statement> <↵
                          expression_statement> <expression> ) <statement>

<jump_statement> ::= goto <identifier> ;
                  | continue ;
                  | break ;
                  | return <expression> ;
```

A következő példává szemlélteti a c nyelv fejlődését:

```
int main(){

    // can't compile me with: -std=c89

    for(int i = 0; i < 10; i++)
        printf("Succes!\n");

    return 0;
```

```
}
```

Ha a "-std=c89" kapcsolóval próbáljuk meg lefordíttatni ezt a programot, akkor hibába ütközünk.

```
hivatk.c: In function 'main':
hivatk.c:5:2: error: C++ style comments are not allowed in ISO C90
  // can't compile me with: -std=c89
  ^
hivatk.c:5:2: error: (this will be reported only once per input file)
hivatk.c:7:2: error: 'for' loop initial declarations are only allowed in C99 or C11 mode
  for(int i = 0; i < 10; i++)
  ^~~
hivatk.c:7:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11 to compile your code
```

Tehát a c89-es szabvánnyal még nem lehetett dupla "/"-jeles kommentet írni és a for ciklus ciklusváltozóját a ciklus előtt kellett deklarálni.

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:

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook/chomsky/realnum.1

Ez a lexer úgy működik, hogy az általunk megadott minták alapján átnézi az általunk megadott szöveget. Először megírjuk a mintákat ami alapján a lexer legenerálja a c programunkat és az fogja elvégezni a szövegelemzést.

A "%{" és "%}" jelek közé berakjuk a generálandó program elejére szánt sorokat, mint pl.: könyvtár importálás vagy deklaráció:

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

Ezután jönnek a definíciók. Jelen esetben egyedül a számjegy fogalmát kell definiálnunk, mivel ez már elegendő egy valós szám leírására.

```
digit [0-9]
```

Az első "%" jel után megadjuk, hogy mit keresünk és hogy mit csináljunk, ha megtaláltuk a keresett szövegrészt.

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

Az utolsó rész - a második "%" jel után - pedig a programunk, ahol elindítjuk a lexikális elemzést a `yylex()` függvény hívásával.

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

A flex és a bison csomagok telepítése után már hozzá is láthatunk a fordításhoz. Először a lexerrel legeneráltatjuk a `c` fájlt, amit majd lefordítunk és futtatunk.

lex -o realnum.c realnum.l

gcc realnum.c -o realnum

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelő)==SIG_IGN)  
    signal(SIGINT, SIG_IGN);
```

Ha a `SIGINT` jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a `jelkezelő` függvény kezelje. (Miután a **man 7 signal** lapon megismertem a `SIGINT` jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)



Bugok

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

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

$$)\$$$
$$\$(\backslash\text{exists } y \ \backslash\text{forall } x \ (x \ \text{prím})) \supset (x < y) \$$$
$$\$(\backslash\text{exists } y \ \backslash\text{forall } x \ (y < x) \supset \neg (x \ \text{prím})))\$$$
Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeXMegoldás 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;`
- `int *b = &a;`
- `int &r = a;`
- `int c[5];`
- `int (&tr)[5] = c;`
- `int *d[5];`
- `int *h ();`
- `int *(*l) ();`
- `int (*v (int c)) (int a, int b)`
- `int (*(z) (int)) (int, int);`

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. `int ***` háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

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

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

**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.