

Univerzális programozás

A kódolás dekódolása kezdőknek

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	Schachinger, Zsolt	2019. május 10.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-02-23	Turing feladatok megoldásának megkezdése.	salesz9902
0.0.6	2019-02-27	A Turing nagyjából készen van. Ismerkedés a Chomsky-val.	salesz9902
0.0.7	2019-03-02	Turing befejezése. Chomsky feladatainak elkezdése.	salesz9902

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.8	2019-03-03	Chomsky nagyjából készen van.	salesz9902
0.0.9	2019-03-04	Ismerkedés a Caesar feladatsorral.	salesz9902
0.1.0	2019-03-08	Caesarban lévő programok kipróbálása, elmélkedés. Enyhén elkezdve.	salesz9902
0.1.1	2019-03-08	Caesarban lévő programok kipróbálása, elmélkedés. Enyhén elkezdve.	salesz9902
0.1.2	2019-03-10	Chomsky teljesen befejezve. Caesar nagyjából készen van.	salesz9902
0.1.3	2019-03-11	Ismerkedés a Mandelbrot fejezzettel.	salesz9902
0.1.4	2019-03-16	Belekezdés a Mandelbrotba.	salesz9902
0.1.5	2019-03-17	Mandelbrot nagyjából készen van. Bár még hiányos.	salesz9902
0.1.6	2019-03-18	Ismerkedés a Welch fejezzettel.	salesz9902
0.1.7	2019-03-24	Teljes gőzerővel neki a Welchnek. Nagyjából készen van.	salesz9902
0.1.8	2019-03-25	Ismerkedés a Conway fejezzettel.	salesz9902
0.1.9	2019-03-30	Conway fejezet megkezdése. Nagyjából kész, bár hiányos.	salesz9902

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.2.0	2019-04-01	Ismerkedés a Schwarzenegger fejezettel.	salesz9902
0.2.1	2019-04-05	Mandelbrot teljesen készén van. Conway elmélkedés, utánaolvasás.	salesz9902
0.2.2	2019-04-08	Ismerkedés a Chaitin fejezettel.	salesz9902
0.2.3	2019-04-20	Csiszolatás a feladatokon, olvasónaplók írása folyamatban.	salesz9902
0.2.4	2019-04-28	Utolsó finomítgatások, csiszolatások a feladatokon visszamenőleg.	salesz9902
0.2.5	2019-04-29	Sikertelen védés. Binfa újratanulmányozása, feladatok átnézése.	salesz9902
0.2.6	2019-05-01	Git kezelésének megtanulása linuxon. Innentől kezdve ilyen módon megy a commitolás, pusholás.	salesz9902
0.2.7	2019-05-06	Sikeres védés a Hibavisszaterjesztéses perceptron feladatból.	salesz9902
0.2.8	2019-05-07	További csiszolások a feladatokon. Képek beillesztése pár feladathoz.	salesz9902
0.2.9	2019-05-09	A könyv végleges befejezése, leadható állapotba hozása.	salesz9902

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]

DRAFT

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
1.4. Pár információ a könyv írásával kapcsolatban	2
II. Tematikus feladatok	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	7
2.3. Változók értékének felcserélése	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	11
2.6. Helló, Google!	13
2.7. 100 éves a Brun téTEL	15
2.8. A Monty Hall probléma	16
3. Helló, Chomsky!	19
3.1. Decimálisból unárisba átváltó Turing gép	19
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	19
3.3. Hivatalos nyelv	21
3.4. Saját lexikális elemző	21
3.5. l33t.l	22

3.6. A források olvasása	24
3.7. Logikus	25
3.8. Deklaráció	26
4. Helló, Caesar!	29
4.1. double ** háromszögmátrix	29
4.2. C EXOR titkosító	29
4.3. Java EXOR titkosító	31
4.4. C EXOR törő	32
4.5. Neurális OR, AND és EXOR kapu	35
4.6. Hiba-visszaterjesztéses perceptron	35
5. Helló, Mandelbrot!	37
5.1. A Mandelbrot halmaz	37
5.2. A Mandelbrot halmaz a std::complex osztályval	38
5.3. Biomorfok	39
5.4. A Mandelbrot halmaz CUDA megvalósítása	40
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	47
6.3. Fabejárás	47
6.4. Tag a gyökér	47
6.5. Mutató a gyökér	47
6.6. Mozgató szemantika	48
7. Helló, Conway!	49
7.1. Hangyszimulációk	49
7.2. Java életjáték	49
7.3. Qt C++ életjáték	49
7.4. BrainB Benchmark	50
8. Helló, Schwarzenegger!	51
8.1. Szoftmax Py MNIST	51
8.2. Mély MNIST	51
8.3. Minecraft-MALMÖ	51

9. Helló, Chaitin!	52
9.1. Iteratív és rekurzív faktoriális Lisp-ben	52
9.2. Gimp Scheme Script-fu: króm effekt	52
9.3. Gimp Scheme Script-fu: név mandala	52
10. Helló, Gutenberg!	53
10.1. Juhász István - Magas szintű programozási nyelvek 1	53
10.2. Kernighan Ritchie - A C programozási nyelv	54
10.3. Programozás	54
III. Második felvonás	55
11. Helló, Arroway!	57
11.1. A BPP algoritmus Java megvalósítása	57
11.2. Java osztályok a Pi-ben	57
IV. Irodalomjegyzék	58
11.3. Általános	59
11.4. C	59
11.5. C++	59
11.6. Lisp	59

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 alkalmi igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Minden esetre 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 gyereknek, 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 gyereknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml ←
    --noout
output.xml validates
rm -f output.xml
dblatex bhax-textbook-fdl.xml -p bhax-textbook.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 'dblatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

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 csemegeznek 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.

1.4. Pár információ a könyv írásával kapcsolatban

A könyvet egy Dell Latitude E5570 notebookon írtam már az elejtől kezdve. Oracle VM VirtualBox program segítségével felsetupoltam egy virtuális gépet, mégpedig pontosabban Ubuntu Linux 18.04 disztró, verzió alatt vittem végig. Választásom azért az Ubuntu Linuxra esett, mert eddig ebben a linuxban volt elég tapasztalom ahhoz, hogy gördülékenyen menjenek a dolgok.

A programokat Visual Studio Code-ban írtam, terminálban fordítottam gcc-vel, illetve az Apache Netbeans 11.0-t használtam a könyv szerkesztésére. Az XML olyan szempontból új volt számomra, hogy nem írtam még benne kódöt, mindezek ellenére egész hamar bele lehet tanulni a dolgokba, könnyen tanulható, egyszerű nyelv.

Utólag sajnálom, hogy későn tanultam meg használni a git-et linuxon, hiszen sokkal átláthatóbban lehet vezetni a repositorykat, mintsem grafikus módon összekattintatjuk a böngészőben.

Azt gondolom, hogy eme könyv írása megtanít minket többek között arra, hogy hogyan dokumentáljuk kódjainkat, hogyan kezeljük azokat, illetve erős alapot ad a jövőre nézve ezekkel kapcsolatban.

DRAFT

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás forrása:

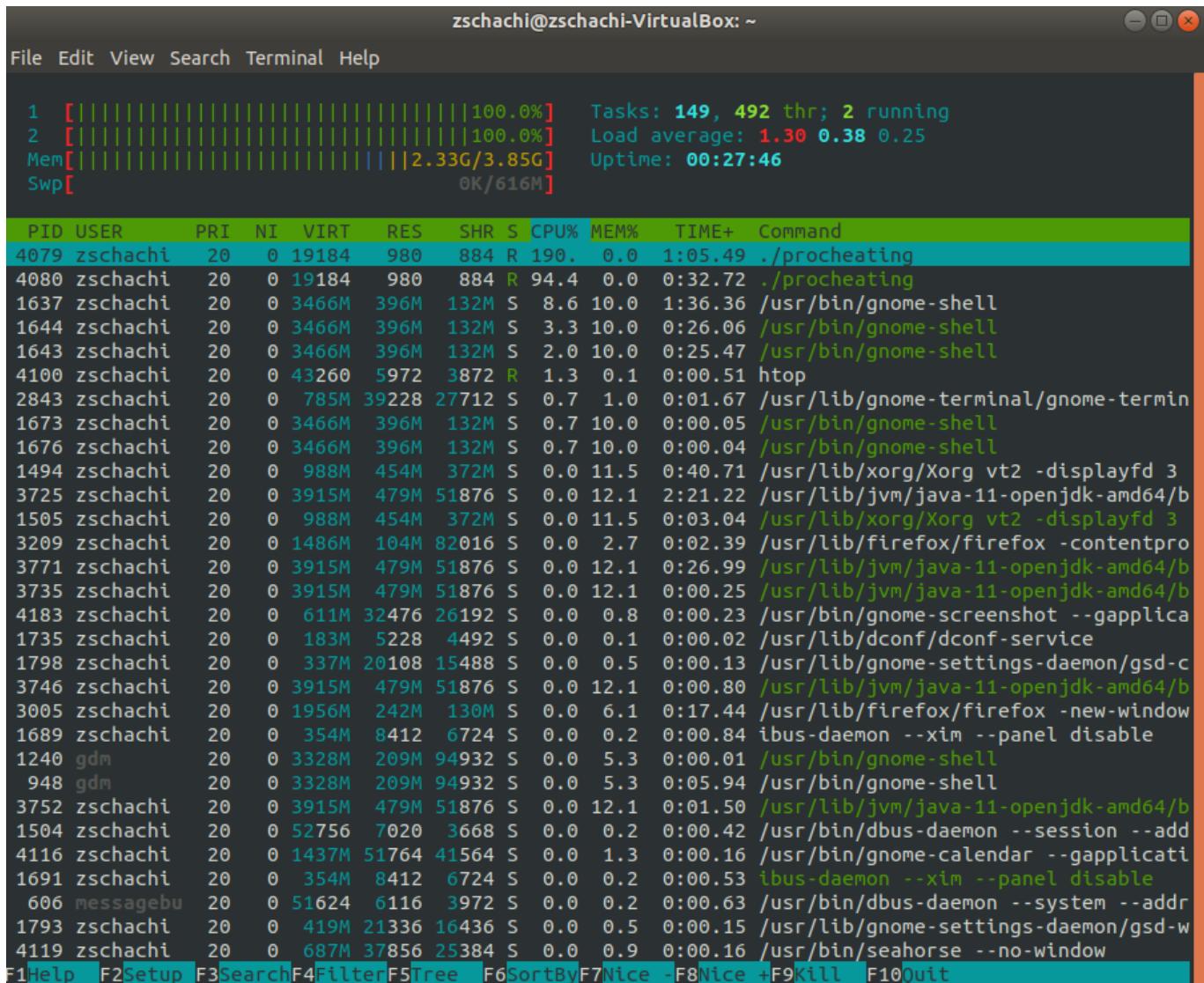
```
Program procheating
{
    #include <stdio.h>
    #include <omp.h>

    int main()
    {
        #pragma omp parallel for
        for (int i=0; i<10; i++)
        {
            i--;
        }

    }
}
```

Az OpenMP (Open Multi-Processing) egy api, ami támogatja a multi többprocesszoros programozást. Ilyen esetben, ezt sokkal egyszerűbb használni, mintsem elkezdenénk a több-szálkezelő (multithread) módszerrel dolgozni..

Azt, hogy a program a processzor összes magját kihasználja, OpenMP segítségével oldottam meg. Ez nagyobb programknál alapszabály, hogy gondoskodjunk a processzor teljes kihasználtságáról.



Ubuntu linux screenshot

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 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 épőlő Lefagy2 már nem tartalmaz feltételezett, csak 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.

Ezt a programot matematikailag lehetetlen megírni számunkra. Ugyanis ilyen program, mint a feladat közben is olvashatjuk nem hozható létre. Ha elkezdjük boncolgatni a problémát, újabb problémába ütközünk, hiszen hamar ellentmondást kapunk a dolgok vizsgálata kapcsán... Elvégre többször is bizonyítva volt már sok nagyobb ember által is, hogy a program nem megírható.

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:

```
Program procheating
{
    #include <stdio.h>

    int main()
    {
        //change the values with an extra variable
        int a, b, c;
        a=2;
        b=5;
        c=0;

        //before the trade
        printf("csere elott:\na=%d, b=%d\n", a,b);
        c=a;
        a=b;
        b=c;
        //after the trade
        printf("csere utan:\na=%d, b=%d\n", a,b);

        //no extra variable used..
        //change values with exor
        a=3;
        b=8;

        //values before the trade
        printf("\ncsere exorral:\n");
    }
}
```

```
printf("csere elott:\na=%d, b=%d\n", a, b);

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

printf("csere után:\na=%d, b=%d\n", a, b);
}

}
```

Mint fent látható, először segédváltozóval oldom meg a cserét, aztán segédváltozó használata nélkül, művelettel(összeadás, kivonás).

Jelen esetben a műveletekkel való csere előtt, az a értéke 3, a b értéke 8. Hogyan is zajlik pontosan itt a csere?

Így:

```
a = 3
b = 8
a = a + b -> a = 11 (3+8)
b = a - b -> b = 3 (11-8)
a = a - b -> a = 8 (11-3)
```

2.4. Labdapattogás

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

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/bouncingball.c>

A programkódok Bátfai Norbert tulajdonában állnak.

Ahhoz, hogy a programot megfelelően tudjuk fordítani, használnunk kell a `-lncurses` kapcsolót a következő módon:

```
gcc "programneve" -o "futtathatoneve" -lncurses
```

Ahhoz, pedig, hogy tudjuk használni a `-lncurses` kapcsolót, telepítenünk kell a `libncurses5-dev`-et:

```
sudo apt-get install libncurses5-dev
```

Az if nélküli módszer

```
Program bouncingball
{
    #include <stdio.h>
    #include <stdlib.h>
```

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

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

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

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

        yj = (yj - 1) % my;
        yk = (yk + 1) % my;

        clear();

        mvprintw(0, 0,
                 " ←
-----");
        mvprintw(24, 0,
                 " ←
-----");
        mvprintw(abs ((yj + (my - yk)) / 2),
                 abs ((xj + (mx - xk)) / 2), "X");

        refresh();
        usleep(150000);
    }
    return 0;
}
```

}

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!

A programkód Bátfai Norbert tulajdonában áll.

A feladat kidolgozásához Racs Tamás könyvét használtam.

Megoldás forrása:

Program BogoMIPS

```
{  
#include <time.h>  
#include <stdio.h>  
  
void  
delay (unsigned long long int loops)  
{  
    unsigned long long int i;  
    for (i = 0; i < loops; i++);  
  
    int  
main (void)  
{  
    unsigned long long int loops_per_sec = 1;  
    unsigned long long int ticks;  
  
    printf ("Calibrating delay loop..");  
    fflush (stdout);  
  
    while ((loops_per_sec <= 1))  
    {  
        ticks = clock ();  
        delay (loops_per_sec);  
        ticks = clock () - ticks;  
  
        printf ("%llu %llu\n", ticks, loops_per_sec);  
  
        if (ticks >= CLOCKS_PER_SEC)  
        {  
            loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;  
  
            printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / ←  
                    500000,  
                    (loops_per_sec / 5000) % 100);  
  
            return 0;  
        }  
    }  
  
    printf ("failed\n");  
    return -1;  
}
```

A BogoMIPS a processzor egy magjának a gyorsaságát méri meg 1 másodperc alatt. Sokan így akarják összehasonlítani számítógépük erősségett, ezt nem erre találták ki. Nyílvánvalóan ez nem lesz annyira pontos, bár elég jó megközelíti a valóságot. A programunk kimenetének második oszlopában a 2 hatványai szerepelnek. Az első oszlop pedig azt mutatja meg, hogy egyes lépésközzel mennyi ideig jutott el a 0-tól az adott hatvány számolásáig, azaz hányszor tickelt a processzorunk.

A program futtatása során az alábbi értékeket kapom:

```
zschachi@zschachi-VirtualBox:~/programming/prog1/turing$ ./bogomips
Calibrating delay loop...3 2
1 4
0 8
1 16
1 32
1 64
1 128
1 256
2 512
3 1024
7 2048
13 4096
24 8192
85 16384
97 32768
255 65536
478 131072
921 262144
1704 524288
2947 1048576
6016 2097152
22113 4194304
23350 8388608
49226 16777216
99516 33554432
201043 67108864
409525 134217728
862637 268435456
1531133 536870912
ok - 700.00 BogoMIPS
```

Ubuntu linux screenshot

2.6. Helló, Google!

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/pagerank.c>

{

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

void
kiir (double tomb[], int db)
{
    int i;

    for (i = 0; i < db; ++i)
        printf ("%f\n", tomb[i]);
}

double
tavolsag (double PR[], double PRv[], int n)
{
    double osszeg = 0.0;
    int i;

    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

    return sqrt(osszeg);
}

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 } ←
    };

    int i, j;

    for (;;)
    {

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

```
    if (tavolsag (PR, PRv, 4) < 0.00000001)
        break;

    for (i = 0; i < 4; ++i)
        PRv[i] = PR[i];

}

kiir (PR, 4);

return 0;
}
}
```

A programkód Bátfai Norbert tulajdonában áll.

A PageRank egy olyan algoritmus, amely linkekhez számokat rendel, majd azokat sorrendbe teszi a hálózatban betöltött szerepük alapján. A Google keresőmotorjának ez az egyik legfontosabb eleme. A PageRank szó egyben a a Google bejegyzett védjegye.

Ez alapján egyértelműen látjuk, hogy melyik weboldal mennyire fontos, és segítségével hasznos listát tudunk felállítani az oldalak fontosságáról. Nyílvánvalóan ez nem jelenti azt, hogy ha a Google keresőnk ötödjére jelenít meg valamit, hogy nem annyira validok rajta az információk, mintsem az első helyen szereplőnek. Közel sem... hiszen ez a kattintásokat veszi figyelembe, ami az oldal népszerűségében nyílvánul meg.

A PageRank-ra a következő a képlet: $\text{PageRank}(i) = (1-d) + d * (\text{PageRank}(j)/L(j))$

2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/brun.r>

Bruntetel

```
{
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))
}
}
```

A programkód Bátfai Norbert tulajdonában áll.

A Brun tételet az ikerprímszámok reciprokaiból képez sorösszegeket, Brun konstans néven ismert véges értékhez konvergál.

A példánkban egy olyan programot írtunk, amely próbálja megközelíteni a Brun konstans értékét. Tehát kiszámolja az ikerprímeket, összegzi a reciprokaikat és részeredményt mutat.

Tisztázzuk az ikerprím fogalmát:

Ikerprímnek két olyan prímszám együttesét nevezzük, amelyek 2-vel térnek el egymástól: például 5 és 7. Mivel a prímszámok (a 2-t kivéve) csak páratlan számok lehetnek, két prímszám között nem lehet kisebb a különbség 2-nél (a (2, 3) pár kivételével). Más megfogalmazás szerint: az ikerprímek két olyan prímszám együttese, amelyek között a prímhézag 2.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/turing/montyhall.r>

Kép forrása: <https://probabilityandstats.wordpress.com/2017/05/11/monty-hall-problem/>

```
Montyhall
{
  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)
```

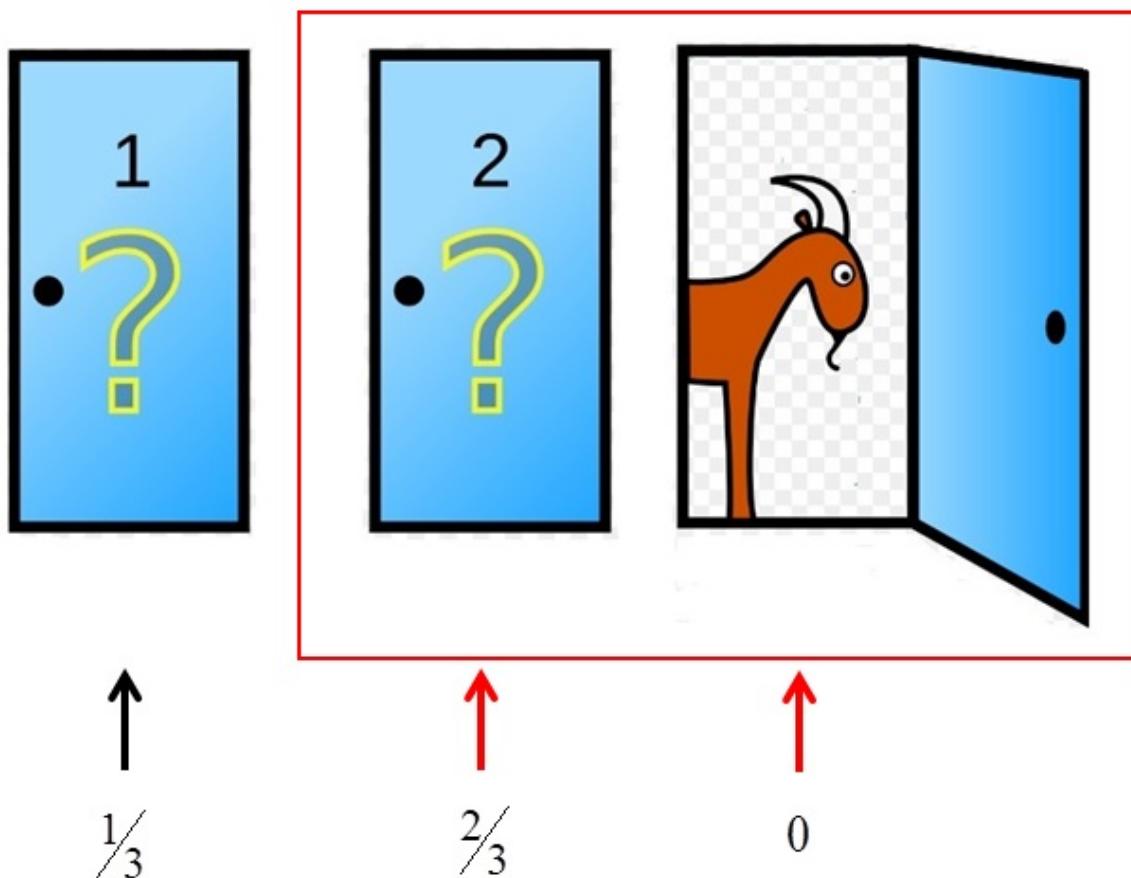
```
for (i in 1:kiserletek_szama) {  
  
    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]  
  
}  
  
valtoztatesnyer = which(kiserlet==valtoztat)  
  
  
sprintf("Kiserletek szama: %i", kiserletek_szama)  
length(nemvaltoztatesnyer)  
length(valtoztatesnyer)  
length(nemvaltoztatesnyer)/length(valtoztatesnyer)  
length(nemvaltoztatesnyer)+length(valtoztatesnyer)  
}
```

A programkód Bátfai Norbert tulajdonában áll.

A Monty Hall egy valószínűségi paradoxon. Az Egyesült Államokban, a Let's Make a Deal televíziós vetélkedő egyik feladatán alapul. Nevét a műsorvezetőről, Monty Hall-ról kapta.

A probléma alap kiindulása az, hogy van 3 csukott ajtónk, amelyek közül 2 mögött van valami számunkra értéktelen dolog, viszont az egyik mögött valami rendkívül értékes lapul. Azt kapjuk meg, amelyik az általunk választott ajtó mögött van. Tehát létezik egy egyszerű valószínűségszámítási eszköz, amely mutatja, hogy melyik ajtót érdemes nekünk választani az adott esetben.

Először tegyük fel, hogy van 1-es 2-es és 3-as ajtónk. A játékosunk először a 3-as ajtót választja. Itt 1/3 eséllyel lesz értékes tárgy. Majd kinyílik a 2-es ajtó. Ez egy értéktelen tárgy lett, ezért ott 0 eséllyel lesz értékes, viszont a mellette lévőben 2/3 az esély.



Az egyik ajtó mögött egy autó, másik kettő mögött kecske található. Az autót keressük. Képen egyértelműen láthatjuk mennyi eséllyel találjuk meg az autót az ajtókban, így, hogy már 1 ajtót kinyitottunk, amiben kecske van.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/chomsky/binarunary.c>

A programkód Molnár Antal Albert tulajdonában van, picit módosítva lett általam.

A Turing-gép Alan Turing angol matematikushoz fűződik, hiszen ő dolgozta ki ennek fogalmát. Ez mindenféle folyamat precízebb megfogalmazására lett kitalálva. Például eljárások, algoritmusok pontosabb leírására.

Írnom kell az unáris számrendserről is. Ez egy nagyon egyszerű számrendszer, amiben vonalakkal ábrázoljuk a számokat. Vegyük példának az 5-öt, ezt öt vonallal ábrázoljuk, a következőképpen:

||||| = 5

A programunk annyit csinál, hogy bekér a felhasználótól egy decimális számot, majd azt kiírja unárisban, 5-ösével elválasztva.

```
zschachi@zschachi-VirtualBox:~/programming/prog1/chomsky$ ./binarunary
Kérlek adj meg egy decimális számot: 16
||||| | | | | | | | |
zschachi@zschachi-VirtualBox:~/programming/prog1/chomsky$
```

Ubuntu linux screenshot

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

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

Legyenek S, X, Y változók. Legyen a, b, c konstansok.

$S \rightarrow abc, S \rightarrow aXbc, Xb \rightarrow bX, Xc \rightarrow Ybcc, bY \rightarrow Yb, Ay \rightarrow aax, Ay \rightarrow aa$

Noam Chomsky szintén egy nyelvész volt, akinek a fenti nyelv grammaticáját is köszönhetjük. Érdekes módon rengeteg kiemelkedő foglalkozása mellett informatikus is volt.

S, X, Y: „változók” (a nemterminálisok)
a, b, c: „konstansok” (a terminálisok)
 $S \rightarrow abc$, $S \rightarrow aXbc$, $Xb \rightarrow bX$, $Xc \rightarrow Ybcc$, $bY \rightarrow Yb$, $aY \rightarrow aaX$, $aY \leftrightarrow S$ (a mondatszimbólum)

S (S → aXbc)
aXbc (Xb → bX)
abXc (Xc → Ybcc)
abYbcc (bY → Yb)
aabbc

S (S → aXbc)
aXbc (Xb → bX)
abXc (Xc → Ybcc)
abYbcc (bY → Yb)
aYbbcc (aY → aaX)
aaXbbcc (Xb → bX)
aabXbcc (Xb → bX)
aabbXcc (Xc → Ybcc)
aabbYbcc (bY → Yb)
aabYbbccc (bY → Yb)
aaYbbbcc (aY → aa)
aaabbccc

A, B, C: „változók” (a nemterminálisok)
a, b, c: „konstansok” (a terminálisok)
 $A \rightarrow aAB$, $A \rightarrow aC$, $CB \rightarrow bCc$, $cB \rightarrow Bc$, $C \rightarrow bc$ (a képzési ↔ szabályok)
S (A kezdőszimbólum)

A (A → aAB)
aAB (A → aC)
aaCB (CB → bCc)
aabCc (C → bc)
aabbcc

A (A → aAB)
aAB (A → aAB)
aaABB (A → aAB)
aaaABBB (A → aC)
aaaaCBBB (CB → bCc)
aaaabCcBB (cB → Bc)
aaaabCBcB (cB → Bc)
aaaabCBBc (CB → bCc)
aaaabbCcBc (cB → Bc)
aaaabbCBcc (CB → bCc)
aaaabbbCccc (C → bc)
aaaabbbbcccc

3.3. Hivatkozási nyelv

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

Megoldás forrása:

```
Program hivatkozasinyelv
{
    #include <complex.h>
    #include <stdbool.h>

    int main()
    {
        long long int asd;
        complex stnum;
    }
}
```

A C nyelvnek is vannak régebbi, illetve újabb változatai. Ilyen például a C89, illetve a C99. Összehasonlítva a C89-hez képest rengeteg változás történt a C99-ben.

Például új header fájlok jöttek be a C99-nél, ilyen a `complex.h`, `stdbool.h` vagy a `tgmath.h`.

Jelentős újítás volt még például az új típusok: `long long int`, vagy a `complex` típus.

A fenti kód például C89-ben nem futna le, mivel még nem ismerné a header fájlokat, illetve a `long long int` típust...

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 használunk, azaz óriások vállán álljunk és ne kispályázzunk!

Megoldás forrása:

```
Program lexikalisi
%
#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 programkód Bátfai Norbert tulajdonában áll.

A programnak megadjuk/definiáljuk a számokat, ezt a [0–9] sorban láthatjuk. Itt azt adjuk meg, hogy bármely szám nullától kilencig, hányszor fordulhat elő. Az ez utáni sorban a . | \n { } után következő utasításnál többet figyelmen kívül hagyjuk.

3.5. l33t.l

Lexelj össze egy l33t cipher!

Megoldás forrása: <https://github.com/Salesz9902/prog1/blob/master/l33t.c>

A programkód Bátfai Norbert tulajdonában áll.

Ebben a feladatban Tóth Balázs volt a tutorom.

```
{  
%{  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <ctype.h>  
  
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))  
  
struct cipher {  
    char c;  
    char *leet[4];  
} l337d1c7 [] = {  
  
{'a', {"4", "4", "@", "/-\\"}},  
{'b', {"b", "8", "|3", "|"}},  
{'c', {"c", "(", "<", "{"}},  
{'d', {"d", "|)", "[", "|"}},  
{'e', {"3", "3", "3", "3"}},  
{'f', {"f", "|=", "ph", "|#"}},  
{'g', {"g", "6", "[", "[+"}},  
{'h', {"h", "4", "|-", "[ - "}},  
{'i', {"1", "1", "|", "!"}},  
{'j', {"j", "7", "_|", "_/"}},  
{'k', {"k", "|<", "1<", "|{"}},  
{'l', {"l", "1", "|", "|_"}},  
{'m', {"m", "44", "(V)", "|\\|/|"}},  
{'n', {"n", "|\\|", "/\\/", "/V"}},
```

```
{'o', {"0", "0", "() ", "[]"}},  
'p', {"p", "/o", "|D", "|o"}},  
'q', {"q", "9", "O_", "(,)"}},  
'r', {"r", "12", "12", "|2"}},  
's', {"s", "5", "$", "$"}},  
't', {"t", "7", "7", "'+'"}},  
'u', {"u", "|_|", "(_) ", "[_]"}},  
'v', {"v", "\\", "\\", "\\"}},  
'w', {"w", "VV", "\\\\"}, "(/\\\")"}},  
'x', {"x", "%", ")(" , ")"}},  
'y', {"y", "", "", ""}}},  
'z', {"z", "2", "7_ ", ">_"}}},  
  
'0', {"D", "0", "D", "0"}},  
'1', {"I", "I", "L", "L"}},  
'2', {"Z", "Z", "Z", "e"}},  
'3', {"E", "E", "E", "E"}},  
'4', {"h", "h", "A", "A"}},  
'5', {"S", "S", "S", "S"}},  
'6', {"b", "b", "G", "G"}},  
'7', {"T", "T", "j", "j"}},  
'8', {"X", "X", "X", "X"}},  
'9', {"g", "g", "j", "j"}}  
  
// https://simple.wikipedia.org/wiki/Leet  
};  
  
%}  
%%  
. {  
  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
    {  
  
        if(l337d1c7[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int) (100.0*rand() / (RAND_MAX+1.0));  
  
            if(r<91)  
                printf("%s", l337d1c7[i].leet[0]);  
            else if(r<95)  
                printf("%s", l337d1c7[i].leet[1]);  
            else if(r<98)  
                printf("%s", l337d1c7[i].leet[2]);  
            else  
                printf("%s", l337d1c7[i].leet[3]);  
  
            found = 1;  
        }  
    }  
}
```

```
        break;
    }

}

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

}

%%

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

A leet nyelv egy internetes nyelv. Bizonyos betű karaktereket számokkal helyettesítünk, amik erősen hasonlítanak a betűkhöz.

Például:

3 = E
4 = A
1 = l
7 = T

A fentiek ismeretében rájöhetünk, hogy a leet szó => 1337 leet nyelven írva. De akár írhatjuk így is: 133t Programunk annyit csinál, hogy beolvas a terminálról karaktereket, aztán ugyanazt visszaadja leet nyelven, feltéve, hogy definiálva van.

3.6. A források olvasása

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

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



Bugok

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

i.

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

ii.

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

Itt egy egyszerű for ciklust láthatunk, amely 0-tól 5-ig megy. Megfigyelhető, hogy az i inkrementálása prefix formában van jelen.

iii.

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

Szintén, mint az előző, annyi különbséggel, hogy már itt postfix formában inkrementál az i.

iv.

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

Szintén egy for ciklus, ami egy tömb i-edik elemét teszi egyenlővé az i++-szal.

v.

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

Itt már találhatunk egy és operátort, ami kettő darab pointert növel eggyel-eggyel.

vi.

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

Itt egy printf függvényt láthatunk, amely kiír két változót, ami egy másik függvény visszatérési értéke lesz.

vii.

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

Itt szintén egy függvény visszatérési, illetve egy változó értékét írja ki.

viii.

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

Itt két számot iratunk ki, ugyanazt a változót, viszont először referenciaiként hivatkozva rá.

Ebben a programban egy jelkezelővel "játszadozhatunk". Ha a program futása során megnyomjuk a Ctrl+C billentyűkombinációt, aminek meg kellene szakítani a folyamatot, először nem fogja. Aztán majd mégegyszeri lenyomás után már másképp veszi figyelembe az általunk kiküldött jelet a programunk.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

```
$ (\forall x \exists y ((x < y) \wedge (y \text{ prim})) \wedge (S y \text{ prim})) \leftrightarrow $
```

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

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

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

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

A LaTeX egy texen alapuló szövegformázó rendszer, amely dokumentumok, szakdolgozatok, akár tudományos cikkek írására is használnak. Matematikusok gyakran használják.

A következőt másoltam be egy .tex kiterjesztésű fájlba:

```
$\forall x \exists y \text{Szeret}(x, y)$  
$\exists y \forall x \text{Szeret}(x, y)$  
$\exists x \forall y \text{Szeret}(x, y)$  
$\exists x \forall y \neg \text{Szeret}(x, y)$
```

A következőképpen tudjuk a .tex kiterjesztésű fájlunkat .pdf-é varázsolni:

```
pandoc -t latex logic.tex -o logic.pdf
```

A fenti tex-re a következő kimenetet kaptam a pdf-be:

$\forall x \exists y \text{Szeret}(x, y) \quad \exists y \forall x \text{Szeret}(x, y) \quad \exists x \forall y \text{Szeret}(x, y) \quad \exists x \forall y \neg \text{Szeret}(x, y)$

Ubuntu linux screenshot

3.8. Deklaráció

Ebben a feladatban György Dóra tutora voltam.

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referencia
- egészek tömbje
- egészek tömbjének referencia (nem az első elemé)
- egészre mutató mutatók tömbje
- egészre mutató mutató 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 forrása: <https://github.com/Salesz9902/prog1/blob/master/deklaracio.c>

Már a legkisebb programokban is találhatunk változó vagy függvénydeklarációt. Ezek kulcsfontosságúak számunkra, hiszen így tudunk tárolni adatokat egyszerűen amiket programunk során felhasználunk. Illetve a függvényekkel saját függvényeket is írhatunk.

Fontos megemlíteni, hogy egy változó deklarálásakor, akár univerzális, akár konkrét típust, de meg kell adnunk, különben hibaüzenetet fogunk kapni fordításkor. Kezdőknél gyakori, hogy a deklarációt összekeverik az értékadással, avagy deklarációinak nevezik az értékadást.

A fenti pontokban több féle deklarációt láthatunk. Az előbb említett értékadás NEM (!) deklaráció. A következőképpen néznek ki:

```
int a; //deklaráció
a = 5; //értékadás
int b = 10; //deklaráció és értékadás egyben
```

Ha egy változót deklarálunk, és nem adunk neki értéket, akkor nagy eséllyel valamilyen "memóriaszemetet" kapunk, ugyanis ilyenkor a programunk az adott változóinknak véletlenszerűen foglal le helyet a memóriában, így ez keletkezik belőle. Ebből az következik, hogy olyan változóknak, amit még az értékének megváltoztatása előtt ki szeretnénk iratni, akkor ne feltétlenül nullára számítsunk, hiszen közel sem biztos, hogy az lesz a kezdőértéke. Ezt az alábbi példában láthatjuk:

```
#include <stdio.h>

int main()
{
    int a, b, c, d, e, f, g;
    printf("%d, %d, %d, %d, %d, %d, %d\n", a,b,c,d,e,f,g);
}
```

A program futtatása, kimenete:

```
zschachi@zschachi-VirtualBox:~/programming/practice$ ./memtrash
22014, 995882304, 22014, 565475760, 32767, 0, 0
zschachi@zschachi-VirtualBox:~/programming/practice$
```

Ubuntu linux screenshot

DRAFT

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás forrása: https://github.com/salesz9902/textbook/blob/master/files/caesar/double_trimatrix.c

A programkód Bátfai Norbert tulajdonában áll.

Elsősorban megadjuk a mátrix sorainak számát. `int nr = 5` Ezután deklarálunk egy valós értékre mutató mutatót.

Érdemes megfigyelnünk a `malloc` függvényt. 5-ször 8 bájtot foglal le, és egy mutatót ad vissza. Ha 0 a méret, akkor a függvény NULL-t vagy egy egyéni mutatót ad vissza. Az egyik for ciklusban `nr` alkalommal (azaz 5) a `malloc` segítségével `tm[i]`-nek lefoglalja a helyet. Az elsőnél 8 bájtot, aztán 16-ot, aztán 32-t és így tovább az 5.-ig.

Futtatásnál a következő kimenetet láthatjuk:

```
./double_trimatrix
0x7ffce6bf4d40
0x55eba3f9a670
0x55eba3f9a6a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: https://github.com/salesz9902/textbook/blob/master/files/caesar/exor_task/xor.c

A programkód Bátfai Norbert tulajdonában áll.

Ebben a feladatban Tóth Balázs tutorált.

```
#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);

}
}
```

Egy olyan program, amely általunk megadott kulcs, illetve karakterhossz által generál nekünk egy titkos szöveget, amit az előbb említett adatok felhasználásával tudunk feltörni. Sok esetben hasznunkra lehet.

A main függvényünknel érdemes megfigyelni a paraméterként adott argumentumokat, mégpedig `argc` és `**argv`. Az `argc` a futtatásnál terminálon keresztül bekért adatok számáért felel, ha lehet így fogalmazni. A `**argv` pedig eltárolja a beolvasott argumentumokat, egyesével.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

A programkód Molnár Antal Albert tuladonában áll.

```
import java.util.*;  
  
class XorEncode {  
    public static void main(String[] args) {  
        String kulcs = "";  
  
        if(args.length > 0) {  
            kulcs = args[0];  
        } else {  
            System.out.println("Kulcs nélkül nem ←  
titkosítok!");  
            System.out.println("Használat: java ←  
XorEncode.java [kulcs]");  
            System.exit(-1);  
        }  
  
        Scanner sc = new Scanner(System.in);  
        String str = "";  
  
        while(sc.hasNext()) {  
            str = sc.next();  
            System.out.println(xor(kulcs, str));  
        }  
    }  
  
    public static String xor(String kulcs, String s) {  
        StringBuilder sb = new StringBuilder();  
  
        for(int i = 0; i < s.length(); i++) {  
            sb.append((char)(s.charAt(i) ^ kulcs.←  
charAt(i % kulcs.length())));  
        }  
  
        return sb.toString();  
    }  
}
```

Ebben a feladatban ugyanazt írjuk meg, mint az előzőben, csak nem C-ben, hanem egy erősen objektum orientált nyelvben, a Java-ban.

A program ugyan úgy működik, mint C elődje, szintén a bekért szöveget titkosítja.

4.4. C EXOR törő

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

Megoldás forrása: https://github.com/salesz9902/textbook/blob/master/files/caesar/exor_task/exor.c

A programkód Bátfai Norbert tulajdonában van.

Ebben a feladatban Butcovan György tutorált.

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE

#include <stdio.h>
#include <unistd.h>
#include <string.h>

void
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{

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

        titkos[i] ^= kulcs[i%kulcs_meret];
    }
}

char *
szo_xor(char * szo,const char kulcs[],int index)
{
    int len=strlen(szo);

    for(int i=0;i<len;i++)
    {
        szo[i]^=kulcs[index%KULCS_MERET];
        index++;
    }
    return szo;
}

int
main (void)
{

    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
```

```
char *p = titkos;
int olvasott_bajtok;
char w[20];

// 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';

//printf("hossz:%d\n",strlen(titkos));

// osszes kulcs eloallitasa
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;

for(int jj=0;jj<KULCS_MERET;jj++)
{
    strcpy(w,"és ");
    if(memmem(titkos,p-titkos,szo_xor(w,kulcs,jj),strlen(w))) //talalt =1, ↵
        vagy nem=0
    {
for(int kk=0;kk<KULCS_MERET;kk++)
{
    strcpy(w,"amelyik ");
    if(memmem(titkos,p-titkos,szo_xor(w,kulcs,kk),strlen(w))) //talalt =1, ↵
```

```
vagy nem=0
{
    for(int ll=0;ll<KULCS_MERET;ll++)
    {
        strcpy(w, " olyan ");

        if(memmem(titkos,p-titkos,szo_xor(w,kulcs,ll),strlen(w))) // ←
            talalt =1, vagy nem=0
        { //printf("%s\n",w);
            for(int mm=0;mm<KULCS_MERET;mm++)
            {
                strcpy(w,"ezért ");
                if(memmem(titkos,p-titkos,szo_xor(w,kulcs,mm), ←
                    strlen(w))) //talalt =1, vagy nem=0
                {
                    exor(kulcs, KULCS_MERET, titkos, p - titkos);
                    printf("Kulcs: [%c%c%c%c%c%c%c]\nTiszta szöveg: ←
                        [%s]\n",
                        ii, ji, ki, li, mi, ni, oi, pi, titkos);
                    // ujra EXOR-ozunk, így nem kell egy masodik ←
                    buffer
                    exor(kulcs, KULCS_MERET, titkos, p - titkos);
                }
            }
        }
    }
}
}

return 0;
}
```

Ha tudjuk a kulcsot, illetve a karakterhosszat, könnyen feltörhetjük az exor titkosított kódot az adatok megadásával. Ezzel a módszerrel akár üzenhetünk is társunknak, illetve olyan szövegeket törhetünk vele, amiről tudjuk mi alapján lett titkosítva.

Ebben a programkódban a Brute force-ot alkalmazzuk, annak segítségével törjük fel a szöveget és a kulcsot, ez addig fut, amíg nem ütközik helyes megoldásba.

Van egy függvényünk, név szerint exor, ami maga az exor művelet végzi el számunkra. A kulcsokat egy

kulcs nevű tömbben tároljuk, mégpedig a main függvényen belül, illetve az olvasott bájtok mérete szintén itt van tárolva. A read segítségével beolvassuk a titkos szöveget, ezzel már visszakapjuk visszatérési értékként a beolvasott szöveget is.

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

A neurális hálózat biológiai neuronok összekapcsolt csoportja. Két koncepcióból jött létre, mégpedig a biológiai és a mesterséges neurális hálózatok ötvözetéből. Az agyunkban is neuronok találhatóak, amik hasonló elven működnek, mint itt, csak magától értetődően természetes módon...

Itt egy neurális hálót készítünk R nyelvben. A neurális háló mesterséges oldala úgy néz ki, hogy megadjuk a programunknak, milyen bemenetre milyen kimenetet adjon, aztán ezt a programunk ennek alapján elkezdi leutánozni egy bizonyos szinten.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/caesar/perceptron/ql.hpp> <https://github.com/salesz9902/textbook/blob/master/files/caesar/perceptron/main.cpp>

A programkódok Bátfai Norbert tulajdonában vannak.

```
#include <iostream>
#include "ql.hpp"
#include <png++/png.hpp>

int main(int argc, char **argv)
{
    png::image<png::rgb_pixel> png_image(argv[1]);

    int size = png_image.get_width() * png_image.get_height();

    Perceptron* p = new Perceptron(3, size, 256, 1);

    double* image = new double[size];

    for(int i{0}; i<png_image.get_width(); ++i)
        for(int j{0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;

    double value = (*p)(image);
```

```
    std::cout << value << std::endl;

    delete p;
    delete [] image;
}
```

Ezt a neuronmodellt a 20. század közepén használták először hatékony képfelismerő algoritmusként.

A perceptron hátránya, hogy kettőnél több réteg esetén a tanítása nehezen kivitelezhető, ugyanis azok a gradiensereszkedések, melyek egy veszteségfüggvényt próbálnak iteratív módon minimalizálni, és ehhez a függvény gradiensével számolnak.

A program fordításához telepítenünk kell a libpng++-t.

```
sudo apt install libpng++
```

A programot a következőképpen kell fordítanunk:

```
g++ ql.hpp main.cpp -o perc -lpng -std=c++11
```

Aztán futtatni a következőképp:

```
./perc (something.png)
```

Használjuk például a mandelbrotnál legenerált png képünket.

A programunk kielemzi a képet, aztán a képünk alapján visszaad egy értéket.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/mandelbrot/mandelbrot.cpp>

A Mandelbrot-halmaz a komplex számsíkon különböző pontok halmaza. Van rá egy rekurzív sorozat, amely abszolút értéken korlátos.

A rekurzív sorozat az alábbi: $x_{n+1} := (x_n)^2 + c$

A fenti C++ programban a Mandelbrot-halmazt fogjuk ábrázolni, mégpedig egy .png kiterjesztésű képen.

Miután lefordítottuk a kódunkat a következőképpen:

```
g++ mandelbrot.cpp -lpng -o mandel
```

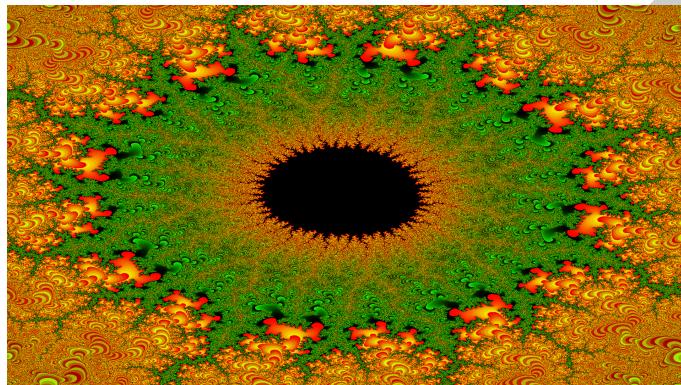
A forráskódban látszik, hogy futtatás után kapunk egy képet kimenet.png néven egész érdekes eredménnyel.

Mandelbrot által generált default image

Ha a másik módon futtatjuk le a programot, ahol már picit manuálisan adhatunk meg több értéket is számára, mint például: szélesség, magasság stb.

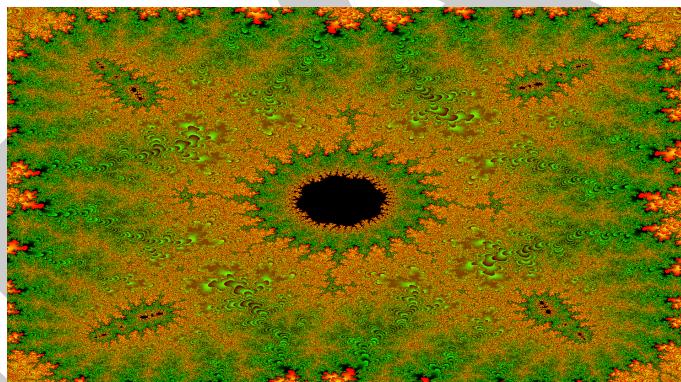
```
./3.1.2 mandel.png 1920 1080 2040 ←  
-0.01947381057309366392260585598705802112818 ←  
-0.0194738105725413418456426484226540196687 ←  
0.798505756933826860155341774655971676111 ←  
0.798505756934379196110285192844457924366
```

Például ezt kipróbálva a következő kimenetet kapjuk szintén kép formájában:



Mandelbrot által generált image

Egy picit másabb generált kép:



Mandelbrot által generált image

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/mandelbrot/mandelbrot2.cpp>

Ebben a feladatban György Dóra tutoráltja voltam.

Ezzel a programmal szintén a Mandelbrot-halmazt ábrázoljuk, viszont itt már `std::complex` osztállyal tesszük meg struktúra alkalmazása helyett. Miután fordítottuk, itt is ugyanazt a képet láthatjuk futtatásnál, mint az előbbinél, csupán a forráskód van másképp kivitelezve.

Itt elhagyjuk a struktúra használatát, helyette osztályt használunk. Talán a struktúra használata talán előnyösebb bizonyos szempontokból, bár egyáltalán nincs köztük olyan nagy különbség, hogy erős okunk legyen rá.

Deklaráljuk a `reC`, `imC`, `reZ` és `imZ` változókat, itt már sejtjük, hogy a komplex számoknak nagy szerepe lesz a programunkban. Ha programunk sikeresen elvégzett minden műveletet, akkor elmentjük a az álláspontot, képet, aztán ezt közöljük is a standard kimeneten.

Fordítás:

```
g++ mandelbrot2.cpp -lpng -O3 -o mandelbrot2
```

Futtatás:

```
./mandelbrot2.cpp mandelcomplex.png 1920 1080 1020  
0.4127655418209589255340574709407519549131  
0.4127655418245818053080142817634623497725  
0.2135387051768746491386963270997512154281  
0.2135387051804975289126531379224616102874
```

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbgrRzY76E>

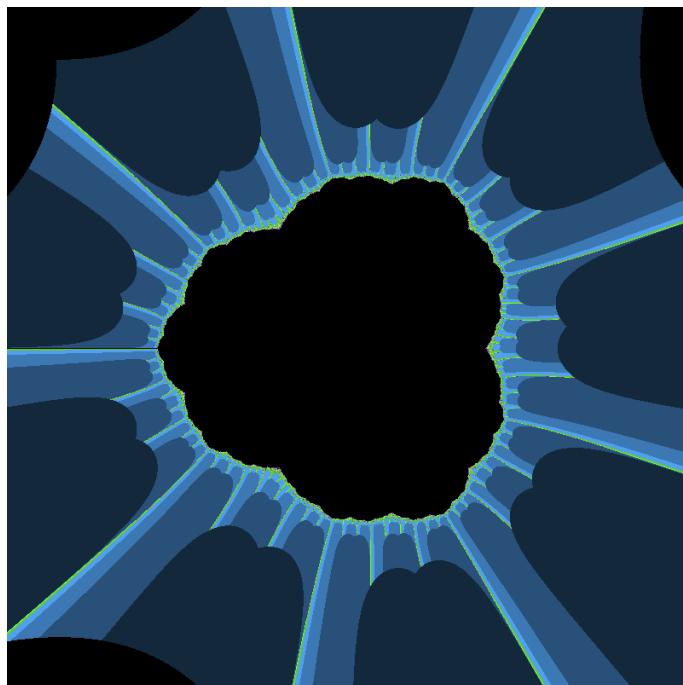
Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/mandelbrot/biomorf.cpp>

A biomorfok jelentősen közel állnak a Mandelbrot-halmazhoz, ugyanis itt szintén a komplex számsíkkal dolgozik a programunk.

Programunk fordítása után, a következőképpen futtassuk:

```
./bmorf bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
```

A fenti esetben ismét egy png kiterjesztésű fájlt fogunk kapni, mégpedig `bmorf.png` néven. Itt már sokkal színgazdagabb formát fogunk kapni, ami picit látványosabb is.



Biomorfok által generált image

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása:

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfai, 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
```

```
//  
// Mandelbrot png  
// Programozó Páternoszter/PARP  
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←  
_01_parhuzamos_prog_linux  
//  
// https://youtu.be/gvaqijHlRUs  
//  
  
#include <png++/image.hpp>  
#include <png++/rgb_pixel.hpp>  
  
#include <sys/times.h>  
#include <iostream>  
  
#define MERET 600  
#define ITER_HAT 32000  
  
__device__ int  
mandel (int k, int j)  
{  
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:  
    // most eppen a j. sor k. oszlopaban vagyunk  
  
    // számítás adatai  
    float a = -2.0, b = .7, c = -1.35, d = 1.35;  
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ←  
        ITER_HAT;  
  
    // a számítás  
    float dx = (b - a) / szelesseg;  
    float dy = (d - c) / magassag;  
    float reC, imC, reZ, imZ, ujreZ, ujimZ;  
    // Hány iterációt csináltunk?  
    int iteracio = 0;  
  
    // c = (reC, imC) a rács csomópontjainak  
    // megfelelő komplex szám  
    reC = a + k * dx;  
    imC = d - j * dy;  
    // z_0 = 0 = (reZ, imZ)  
    reZ = 0.0;  
    imZ = 0.0;  
    iteracio = 0;  
    // z_{n+1} = z_n * z_n + c iterációk  
    // számítása, amíg |z_n| < 2 vagy még  
    // nem értük el a 255 iterációt, ha  
    // viszont elértek, akkor úgy vesszük,  
    // hogy a kiinduláci c komplex számra
```

```
// az iteráció konvergens, azaz a c a
// Mandelbrot halmaz eleme
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteracionsHatar)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;

}
return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{

    int j = blockIdx.x;
    int k = blockIdx.y;

    kepadat[j + k * MERET] = mandel (j, k);

}
 */

__global__ void
mandelkernel (int *kepadat)
{

    int tj = threadIdx.x;
    int tk = threadIdx.y;

    int j = blockIdx.x * 10 + tj;
    int k = blockIdx.y * 10 + tk;

    kepadat[j + k * MERET] = mandel (j, k);

}

void
cudamandel (int kepadat [MERET] [MERET])
{

    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (←
```



```
255 -  
(255 * kepadat[j][k]) / ←  
ITER_HAT));  
}  
}  
kep.write(argv[1]);  
  
std::cout << argv[1] << " mentve" << std::endl;  
  
times(&tmsbuf2);  
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime  
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;  
  
delta = clock() - delta;  
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;  
  
}
```

Szintén egy összetett feladattal állunk szemben. Ahhoz, hogy megfelelően tudjuk fordítani/futtatni a programot, telepítenünk kell az nvidia-cuda-toolkit nevű csomagot.

A programunk konkrétan optimalizálni próbálja a "munkánkat", egy gyorsabb számolást eredményez a hátterben, ami nagyon sok esetben nagy segítségünkre lehet, hiszen ki ne akarná, hogy gyorsabban dolgozzon a gépe.

Az optimalizálásról már korábban is volt szó a könyvben, mégpedig a legelső feladatunkban például, ahol OpenMP segítségével osztottuk fel processzormagokra a végrehajtandó "munkát".

5.5. Mandelbrot nagyító és utazó C++ nyelven

A feladatban Molnár Antal Albert könyve volt segítségemre.

Épít 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 forrása: <https://github.com/salesz9902/prog1/tree/master/mandelzoom>

Ebben a feladatban egy GUI-t fogunk létrehozni a Qt Creator szoftverrel. (Ez egy multiplatformos keretrendszer, amit épp erre (is) találtak ki.)

Tehát itt arról van szó, hogy a Qt Creatorban létre tudunk könnyen hozni egy grafikus felületet az előző C++ kódunkhoz, a Mandelbrot-hoz.

YouTube-on rengeteg oktatóvideót találhatunk a Qt Creator szoftverről. Itt is van egy:

<https://www.youtube.com/watch?v=3SIj6zL6mmA>

Ebből a videóból már tényleg gond nélkül elindulhatunk egy úton a GUI szerkesztés felé.

5.6. Mandelbrot nagyító és utazó Java nyelven

Hasonló szituációban vagyunk, mint az előző feladatnál. Annyi változik, hogy már egy jóval felhasználóközelibb, magasabb szintű programozási nyelven valósítjuk meg, a Javában.

Itt is rengeteg keretrendszerünk van. A programban nagy a testreszabhatóság lehetősége. Mi magunk adhatjuk meg több paramétereit is a programunk elindulásakor felnyíló ablaknak stb.

A konstruktörben beállíthatjuk a Mandelbrot halmaz paramétereit. Például az élességet. Ami még érdekes lehet számunkra, az nem más, mint a BufferedImage típus, amit a Java biztosít számunkra. Ez tulajdonképpen egy osztály, ami lehetőséget ad, hogy külső könyvtár használata nélkül is képesek legyünk egyszerűen képfájlokat létrehozni.



6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás forrása:

```
$ ./java/bin/java polargen.java  
-0.7353431820414118  
-0.33784190028284766  
0.7750031835316805  
0.5524713543467192  
-0.5380423283211784  
1.512849268596637  
2.7148874695500966  
-0.23688836801277952  
-0.3238588036816322  
-0.7963150809415576  
$ ./java/bin/java polargen.java  
-0.6566325405553158  
0.40465899229436114  
0.08634239512228409  
-0.9470321445590416  
0.1926238606249351  
0.7705517022243931  
0.9084531239664848  
-1.4472688950554047  
-1.6250659297425345  
-0.7791586500972545
```

A program 10 darab véletlenszerűen generált normalizált számot köp ki, ahogyan azt várjuk is.

6.2. LZW

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Van egy osztályunk: LZWBInFa, ez építi fel a bináris fájlunkat az általunk beírt bemeneti fájlból. A következőképp kell futtatnunk a kódot:

```
./bin [bemeneti] -o [kimeneti]
```

A kimeneti fájlhoz értelemszerűen megadjuk, hova szeretnénk kiíratni az eredményt.

A programunk sokkal egyszerűbb módon van megírva C-ben. Ugyanis itt már elégé meg van kötve a kezünk a program írásában. Mondhatjuk, hogy ugyanaz a programkód, csak leegyszerűsítve, pár dolgot kivéve az eredeti c++ kódunkból.

6.3. Fabejárás

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Inorder: először a fa bal oldalát járjuk be, a gyökeret, aztán majd a jobb oldalát.

Preorder: gyökérrel indítunk, majd bezárjuk fa bal oldalát, aztán a fa jobb oldalát.

Postorder: gyökérrel indítunk, fa jobb oldalát, aztán a fa bal oldalát járjuk be.

Itt már viszont a programot picit másképpen futtatjuk:

```
./binfa [bemeneti] -o [kimeneti] [o / r]
```

Ha az utolsó argumentum o, postorder, ha pedig r, akkor inorder bejárást fog alkalmazni.

6.4. Tag a gyökér

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Itt az alapvetően C alapú LZWBInFa programkódunkat kellene átírni C++-ba. Ez nem feltétlenül bonyolult feladat. Mivel már el kell mozdulnunk picit az objektumorientált irányba, így egy külön osztályba kell megírnunk a fent említett dolgokat.

6.5. Mutató a gyökér

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

Megoldás forrása: https://github.com/salesz9902/textbook/blob/master/files/welch/mutato_a_gyok.cpp

Ebben az esetben a bináris fa gyökere egy mutató kell, hogy legyen. Ez semmit sem változtat a többi, vagy akár az eredeti LZWBinFa-hoz képest. Ugyanaz a végkimenetele, minden ugyanúgy működik, csak szimplán másiképp van megoldva.

6.6. Mozgató szemantika

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

Megoldás forrása: <https://github.com/salesz9902/textbook/blob/master/files/welch/z3a7.cpp>

Itt használjuk a mozgató-konstruktorkat. A következő változásoknak kell végbemennük a kódunkban:

```
Csomopont * masol ( Csomopont * elem, Csomopont * regifa ) {  
    Csomopont * ujelem = NULL;  
    if ( elem != NULL ) {  
        switch ( elem->getBetu() ) {  
            case '/':  
                ujelem = new Csomopont ( '/' );  
                break;  
            case '0':  
                ujelem = new Csomopont ( '1' );  
                break;  
            case '1':  
                ujelem = new Csomopont ( '0' );  
                break;  
            default:  
                std::cerr<<"HIBA!"<<std::endl;  
                break;  
        }  
        ujelem->ujEgyesGyermek (  
            masol ( elem->egyesGyermek () , regifa )  
        );  
        ujelem->ujNullasGyermek (  
            masol ( elem->>nullasGyermek () , regifa )  
        );  
        if ( regifa == elem )  
            fa = ujelem;  
    }  
    return ujelem;  
}
```

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása:

Nem hiába kapta a feladatunk ezt a nevet. Hiszen programunk a hangyák viselkedését szimulálja le. Figyeljük meg, ahogyan véletlenszerűen változik az apró pontoknak a helyzete, aztán egyre több lesz belőlük. Egyszer itt gyűlnek össze, egyszer ott. Tisztára olyan viselkedést mutatnak mint a hangyák egy élő szituációban.

Az elején elkezdenek egy adott pontból elindulni. Olyan, mintha egy ételmaradékon összegyűlnének, aztán sorban egymás után elkezdenék haza cipelni azokat. A hangyáknak ez egy nagyon jellegzetes tulajdonságuk. A programot sokáig hagyhatjuk futni, hiszen a végtelenségig szimulálja a dolgokat számunkra.

7.2. Java életjáték

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: <https://github.com/salesz9902/textbook/tree/master/files/conway/sejtauto>

Ismét egy eléggé árulkodó nevet kaptunk feladatunknak. Itt különböző pici "sejtautók" mozgásának szemtanúi lehetünk! Az apró mozgalmas sejtek egy-egy élő sejt viselkedését próbálják reprezentálni. Látható, ahogy úgymond legyártódnak a sejtek, aztán elindulnak egymás után egy irányba.

Aztán ahogyan a képernyőnk széléhez érnek, egy újabb részen megjelennek, aztán addig-addig osztódnak, amíg rengeteg nem lesz belőlük.

Alapvetően a sejtek a valóságban is hasonlóan működnek. Kell, hogy legyen egy-egy társuk, különben elpusztulnak.

7.4. BrainB Benchmark

Megoldás forrása:

Itt a Qt keretrendszer segítségével tudjuk megoldani a feladatot. Fontos, hogy amíg eljutunk addig, hogy a programunk megfelelő környezetben megfelelően lefordul és lefut, rengeteg helyet kell annak igénybe vennie. Tehát készüljünk fel rá, hogy legyen 20-30 GB szabad helyünk, hogy kényelmes legyen a program kezelése minden szempontból.

A Qt keretrendszer nagyon nagy eszközökkel rendelkezik, és rengeteg lehetőségünk van kihasználni ezeket. A nagy eszközökészlet a szükséges letöltött dolgok méretében is erősen megnyilvánul.

E program célja, hogy benchmarkot készítsen egy E-sportoló játékosról. Felmére annak tudását, amely segítségével már könnyen besorolható lesz egy adott szintre. A program futtatás után, és akár pár perc játék után részletes leírásokat kapunk kimenetként. Ennek segítségével akár már írányt kaphatunk E-sport karrierünk felépítésében, kérdéssében.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndi8>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

Megoldás forrása:

9.2. Gimp Scheme Script-fu: króm effekt

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szöveghez!

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

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

Tanulságok, tapasztalatok, magyarázat...

9.3. Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelete_a_scheme_programozasi_nyelv

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

Tanulságok, tapasztalatok, magyarázat...

10. fejezet

Helló, Gutenberg!

10.1. Juhász István - Magas szintű programozási nyelvek 1

[?]

Alapfogalmak:

Gépi nyelv: A gépi nyelv, egy olyan nyelv, amely a számítógép számára közvetlenül értelmezhető. Kettes vagy tizenhatos számrendszeren alapul (számokkal ábrázolandó).

Magas szintű nyelv: A magas szintű programozási nyelvek már felhasználó közelibbek. Ezek nem értelmezhetőek közvetlenül a számítógép által. Itt már szükségünk van egy fordítónak, ami lefordítja gépi nyelvre, ahoz, hogy futtatható legyen.

Gépi nyelvezetű vagy alacsony szintű programnyelv például az Assembly, amelyben sokkal nehezebben igazodunk el, hisz egyértelműen látszik, hogy a géphez áll közelebb. Viszont magas szintű programnyelv például a C, amelyben érzékelhetjük is, hogy sokkal jobban érthetőek a C-ben írt kódok, mint akár Assembly-ben. Hisz C-ben mondhatjuk, hogy angol kulcsszavakkal adunk ki "parancsokat" a számítógép számára, ami persze ugyanúgy lefordul majd gépi kódra, aztán futtathatóvá is válik.

Egy programot tudunk szintaktikailag, illetve szemantikailag elemezni. Szintaktikai elemzésnél konkrétan a programkódunk "helyesírására" figyelünk. Tehát, hogy nem-e írtunk el egy adott parancsot például, stb.. Szemantikai elemzésnél már arra figyelünk, hogy miután szintaktikailag helyes a programunk, helyesen fut-e le. Tehát itt azt nézzük, hogy tényleg azt csinálja-e a programunk, ami a célunk volt vele. Helyesen fut-e le.

A programnyelveket két fő csoportba soroljuk: vannak imperatív és dekleratív nyelvek. Az imperatív nyelvek általában az értékadó utasítások megfelelő sorrendben való kiadására koncentrálnak. Ez az a típus, amelyben feltehetően többen programoznak, bár nem feltétlenül, de ha valaki komolyabb programozásra vágyik, ezzel a típussal kezdi el a gyakorlást, majd folytatja a bonyolultabb programokkal. Imperatívak például az eljárásorientált nyelvek, vagy az objektumorientált nyelvek.

A dekleratív csoportba sorolható programkódoknál általában a programíró arra koncentrál, hogy mit szeretne kapni az adott program futása során. Ilyenek például a funkcionális nyelvek, illetve a logikai nyelvek is.

Fontos megjegyezni, hogy elméleti szinten nem fogunk megtanulni programozni. Ezalatt azt értem, hogy ahoz, hogy valaki jó programozóvá váljon, rengeteg programkódot kell átvészelnie mind elméletben, de legfőképpen gyakorlatban.

Utasítások:

Sokféle utasítás létezik. Ilyenek például az értékkadó, üres, ugró, elágaztató, ciklusszervező, hívó, I/O illetve számos egyéb utasítások.

Itt azért pár utasítás eléggé magától értendő. Mint például az értékkadó utasításokkal egy vagy több változó értékkomponensét állítjuk be, vagy éppen módosítjuk.

Az elágaztató utasítások például az if feltétellel kapcsolatos megoldásokat fedi le, azaz valamilyen feltételes, kétirányú logikai utasítás. Ezekből vannak egyirányú, illetve többirányúak is. Ide tartozik a switch utasítás is.

10.2. Kernighan Ritchie - A C programozási nyelv

[KERNIGHANRITCHIE]

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

Ez a könyv egy tematikusan felépülő könyv, amely a C nyelv elsajátításához segíti hozzá olvasóját. Próbálja megismertetni a C nyelvet elég erős szinten, próbál bizonyos keretek között picit mélyebben belemenni a dolgokba.

A könyv alapismeretekkel indít. Hamar lényegre is tör, hiszen már egy "Hello World!" stílusú program megírásával szemléltet az olvasó felé. Erősen ragaszkodik a UNIX-on való fordításra, illetve futtatásra.

Bemutatja a szokásos alap programozási eszközöket. Mint például ciklusok (for, while, do-while), a változó deklarálásától megkezdve a tömbökön át a függvényekig. Kitér külön a változótípusokra, amik működését el is magyarázza, több példán keresztül bemutatja.

Már viszonylag hamar elkezdődnek folyamatos rövid példákkal való szemléltetések, illetve a programkódok kipróbálásra való készítések. A fent leírtak mindegyike érthető módon be van mutatva, le van egyszerű programokba bonyolítva, amelyek értelemszerűen a lehető legjobb megértésre törekednek.

10.3. Programozás

[BMECPP]

A könyvet még nem sikerült beszerezni, így erről nem tudtam elkezdeni megírni az olvasónaplót.

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

DRAFT

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPORG 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 érdeklődésekért és hasznos észrevételeikért.

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