

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	Schachinger, Zsolt	2019. május 9.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun tétel	12
2.8. A Monty Hall probléma	12
3. Helló, Chomsky!	14
3.1. Decimálisból unárisba átváltó Turing gép	14
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	14
3.3. Hivatkozási nyelv	16
3.4. Saját lexikális elemző	16
3.5. l33t.1	17
3.6. A források olvasása	17
3.7. Logikus	18
3.8. Deklaráció	18

4. Helló, Caesar!	21
4.1. double ** háromszögmátrix	21
4.2. C EXOR titkosító	21
4.3. Java EXOR titkosító	22
4.4. C EXOR törő	22
4.5. Neurális OR, AND és EXOR kapu	22
4.6. Hiba-visszaterjesztéses perceptron	22
5. Helló, Mandelbrot!	24
5.1. A Mandelbrot halmaz	24
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	24
5.3. Biomorfok	24
5.4. A Mandelbrot halmaz CUDA megvalósítása	25
5.5. Mandelbrot nagyító és utazó C++ nyelven	25
5.6. Mandelbrot nagyító és utazó Java nyelven	25
6. Helló, Welch!	27
6.1. Első osztályom	27
6.2. LZW	28
6.3. Fabejárás	28
6.4. Tag a gyökér	28
6.5. Mutató a gyökér	28
6.6. Mozgató szemantika	29
7. Helló, Conway!	30
7.1. Hangyaszimulációk	30
7.2. Java életjáték	30
7.3. Qt C++ életjáték	30
7.4. BrainB Benchmark	31
8. Helló, Schwarzenegger!	32
8.1. Szoftmax Py MNIST	32
8.2. Mély MNIST	32
8.3. Minecraft-MALMÖ	32

9. Helló, Chaitin!	33
9.1. Iteratív és rekurzív faktoriális Lisp-ben	33
9.2. Gimp Scheme Script-fu: króm effekt	33
9.3. Gimp Scheme Script-fu: név mandala	33
10. Helló, Gutenberg!	34
10.1. Juhász István - Magas szintű programozási nyelvek 1	34
10.2. Kernighan Ritchie - A C programozási nyelv	35
10.3. Programozás	35
III. Második felvonás	36
11. Helló, Arroway!	38
11.1. A BPP algoritmus Java megvalósítása	38
11.2. Java osztályok a Pi-ben	38
IV. Irodalomjegyzék	39
11.3. Általános	40
11.4. C	40
11.5. C++	40
11.6. Lisp	40

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 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 programoknál alapszabály, hogy gondoskodjunk a processzor teljes kihasználtságáról.

```

zschachi@zschachi-VirtualBox: ~
File Edit View Search Terminal Help

1 [|||||||||||||||||||||||||||||100.0%] Tasks: 149, 492 thr; 2 running
2 [|||||||||||||||||||||||||||||100.0%] Load average: 1.30 0.38 0.25
Mem[|||||||||||||||||||||||||2.33G/3.85G] Uptime: 00:27:46
Swp[|||||||||||||||||||||0K/616M]

  PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
 4079 zschachi   20    0 19184    980   884  R 190.   0.0   1:05.49 ./procheating
 4080 zschachi   20    0 19184    980   884  R  94.4   0.0   0:32.72 ./procheating
 1637 zschachi   20    0 3466M   396M  132M  S   8.6  10.0   1:36.36 /usr/bin/gnome-shell
 1644 zschachi   20    0 3466M   396M  132M  S   3.3  10.0   0:26.06 /usr/bin/gnome-shell
 1643 zschachi   20    0 3466M   396M  132M  S   2.0  10.0   0:25.47 /usr/bin/gnome-shell
 4100 zschachi   20    0 43260   5972  3872  R   1.3   0.1   0:00.51 htop
 2843 zschachi   20    0  785M 39228 27712  S   0.7   1.0   0:01.67 /usr/lib/gnome-terminal/gnome-termin
 1673 zschachi   20    0 3466M   396M  132M  S   0.7  10.0   0:00.05 /usr/bin/gnome-shell
 1676 zschachi   20    0 3466M   396M  132M  S   0.7  10.0   0:00.04 /usr/bin/gnome-shell
 1494 zschachi   20    0  988M  454M  372M  S   0.0  11.5   0:40.71 /usr/lib/xorg/Xorg vt2 -displayfd 3
 3725 zschachi   20    0 3915M  479M  51876  S   0.0  12.1   2:21.22 /usr/lib/jvm/java-11-openjdk-amd64/b
 1505 zschachi   20    0  988M  454M  372M  S   0.0  11.5   0:03.04 /usr/lib/xorg/Xorg vt2 -displayfd 3
 3209 zschachi   20    0 1486M  104M  82016  S   0.0   2.7   0:02.39 /usr/lib/firefox/firefox -contentpro
 3771 zschachi   20    0 3915M  479M  51876  S   0.0  12.1   0:26.99 /usr/lib/jvm/java-11-openjdk-amd64/b
 3735 zschachi   20    0 3915M  479M  51876  S   0.0  12.1   0:00.25 /usr/lib/jvm/java-11-openjdk-amd64/b
 4183 zschachi   20    0  611M 32476 26192  S   0.0   0.8   0:00.23 /usr/bin/gnome-screenshot --gapplika
 1735 zschachi   20    0  183M  5228  4492  S   0.0   0.1   0:00.02 /usr/lib/dconf/dconf-service
 1798 zschachi   20    0  337M 20108 15488  S   0.0   0.5   0:00.13 /usr/lib/gnome-settings-daemon/gsd-c
 3746 zschachi   20    0 3915M  479M  51876  S   0.0  12.1   0:00.80 /usr/lib/jvm/java-11-openjdk-amd64/b
 3005 zschachi   20    0 1956M  242M  130M  S   0.0   6.1   0:17.44 /usr/lib/firefox/firefox -new-window
 1689 zschachi   20    0  354M  8412  6724  S   0.0   0.2   0:00.84 ibus-daemon --xim --panel disable
 1240 gdm        20    0 3328M  209M  94932  S   0.0   5.3   0:00.01 /usr/bin/gnome-shell
  948 gdm        20    0 3328M  209M  94932  S   0.0   5.3   0:05.94 /usr/bin/gnome-shell
 3752 zschachi   20    0 3915M  479M  51876  S   0.0  12.1   0:01.50 /usr/lib/jvm/java-11-openjdk-amd64/b
 1504 zschachi   20    0 52756  7020  3668  S   0.0   0.2   0:00.42 /usr/bin/dbus-daemon --session --add
 4116 zschachi   20    0 1437M 51764 41564  S   0.0   1.3   0:00.16 /usr/bin/gnome-calendar --gapplika
 1691 zschachi   20    0  354M  8412  6724  S   0.0   0.2   0:00.53 ibus-daemon --xim --panel disable
  606 messagebu 20    0 51624  6116  3972  S   0.0   0.2   0:00.63 /usr/bin/dbus-daemon --system --addr
 1793 zschachi   20    0  419M 21336 16436  S   0.0   0.5   0:00.15 /usr/lib/gnome-settings-daemon/gsd-w
 4119 zschachi   20    0  687M 37856 25384  S   0.0   0.9   0:00.16 /usr/bin/seahorse --no-window
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```

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 építő Lefagy2 már nem tartalmaz feltételezett, csak konkrét kódot:

```
Program T1000
{

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

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

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


Mit for kiírni erre a T1000 (T1000) futtatásra?

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

akkor most hogy fog működni? Sehoggy, 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 előtt:\na=%d, b=%d\n", a, b);
        c=a;
        a=b;
        b=c;
        //after the trade
        printf("csere után:\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 előtt:\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 írné egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videókon.)

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

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

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!

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

A BogomIPS a processzor egy magjának a gyorsaságát méri meg 1 másodperc alatt. Sokan így akarják összehasonlítani számítógépük erősségét, ezt nem erre találták ki.

```
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/prog1/blob/master/pagerank.c>

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 Google bejegyzett védjegye.

Ez alapján egyértelműen látjuk, hogy melyik weboldal mennyire fontos, és segítségével hasznos listát tudunk felállítani az oldalak fontosságáról.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás forrása: <https://github.com/salesz9902/prog1/blob/master/brun.r>

A Brun tétel 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!

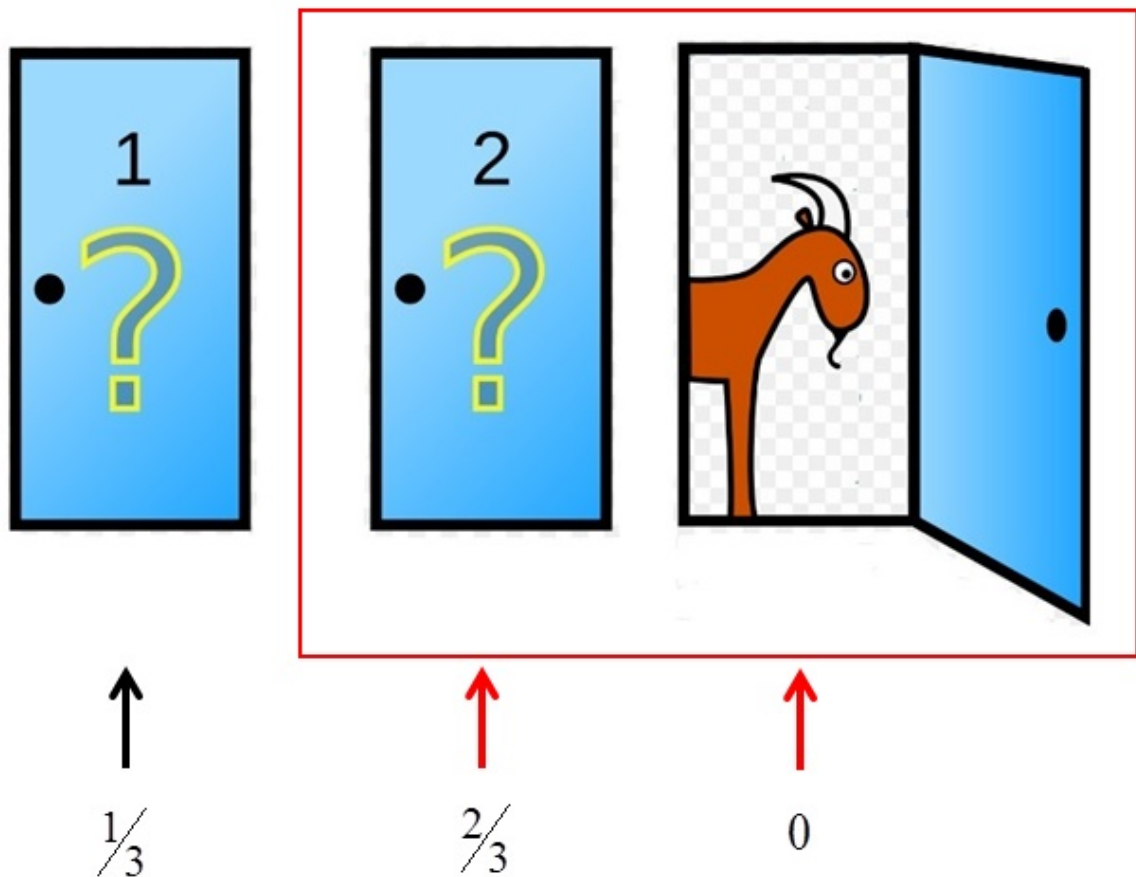
Megoldás forrása: <https://github.com/salesz9902/prog1/blob/master/montyhall.r>

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

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 megmutatja, 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 ajtóknban, így, hogy már 1 ajtót kinyitottunk, amiben kecske van.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

Megoldás forrása: <https://github.com/salesz9902/prog1/blob/master/binarunary.c>

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 számrendszerről 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 grammatikát, 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 grammatikáját is köszönhetjük. Érdekes módon rengeteg kiemelkedő foglalkozása mellett informatikus is volt.

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

$S (S \rightarrow aXbc)$
 $aXbc (Xb \rightarrow bX)$
 $abXc (Xc \rightarrow Ybcc)$
 $abYbcc (bY \rightarrow Yb)$
 $aabbcc$

$S (S \rightarrow aXbc)$
 $aXbc (Xb \rightarrow bX)$
 $abXc (Xc \rightarrow Ybcc)$
 $abYbcc (bY \rightarrow Yb)$
 $aYbbcc (aY \rightarrow aaX)$
 $aaXbbcc (Xb \rightarrow bX)$
 $aabXbcc (Xb \rightarrow bX)$
 $aabbXcc (Xc \rightarrow Ybcc)$
 $aabbYbcc (bY \rightarrow Yb)$
 $aabYbbcc (bY \rightarrow Yb)$
 $aaYbbbcc (aY \rightarrow aa)$
 $aaabbbcc$

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)

$A (A \rightarrow aAB)$
 $aAB (A \rightarrow aC)$
 $aaCB (CB \rightarrow bCc)$
 $aabCc (C \rightarrow bc)$
 $aabbcc$

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

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 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 használjunk, azaz óriások vállán álljunk és ne kispályázzunk!

Megoldás forrása:

```
Program lexikalis
%{
    #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 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 ciphert!

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

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 rájöhethetünk, hogy a leet szó => l337 leet nyelven írva. De akár írhatjuk így is: l33t

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. (Mióután 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 megváránzésre, 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);
```

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

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\texttt{forall } x \texttt{ exists } y ((x < y) \texttt{wedge} (y \texttt{ text{ prím}})))$  
  
$(\texttt{forall } x \texttt{ exists } y ((x < y) \texttt{wedge} (y \texttt{ text{ prím}})) \texttt{wedge} (SSy \texttt{ text{ prím}})) \leftrightarrow  
 )$  
  
$(\texttt{exists } y \texttt{ forall } x (x \texttt{ text{ prím}}) \texttt{ supset } (x < y))$  
  
$(\texttt{exists } y \texttt{ forall } x (y < x) \texttt{ supset } \texttt{ neg } (x \texttt{ text{ prím}}))$
```

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

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

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

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 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 deklarációnak nevezik az értékadást.

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

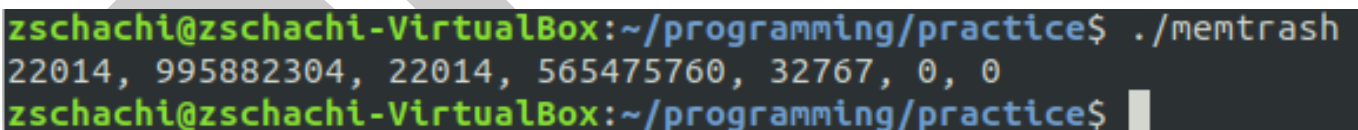
```
int a; //deklaráció
a = 5; //értékadás
int b = 10; //deklaráció és értékadás egyben
```

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 írni, 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

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás forrása: https://github.com/salesz9902/prog1/blob/master/double_trimatrix.c

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/prog1/blob/master/exorencrypt.c>

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.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

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/prog1/blob/master/exor.c>

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.

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

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 elkezd leutánozni egy bizonyos szinten.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: <https://github.com/salesz9902/prog1/blob/master/nlp.hpp> <https://github.com/salesz9902/-prog1/blob/master/main.cpp>

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 programot a következőképpen kell fordítanunk:

```
g++ ql.hpp main.cpp -o perc -lpng
```

Aztán futtatni a következőképp:

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

Használjuk például a mandelbrotnál legenerált png képünket.

A program a képünk alapján visszaad egy értéket.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás forrása: <https://github.com/salesz9902/prog1/blob/master/mandelbrot.cpp>

A Mandelbrot-halmaz a komplex számsíkon különböző pontok halmaza. Van rá egy rekurzív sorozat, amely abszolút értékben 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 kódukunkat a következőképpen:

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

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

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

Megoldás forrása: <https://github.com/salesz9902/prog1/blob/master/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.

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbgRzY76E>

Megoldás forrása: <https://github.com/salesz9902/prog1/blob/master/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 biomorf.png 800 800 10 -2 2 -2 2 .285 0 10
```

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

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása:

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 háttérben, ami nagyon sok esetben nagy segítségünkre lehet, hiszen ki ne akarná, hogy gyorsabban dolgozzon a gépe.

Az optimalizálásáról 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

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 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 konstruktorban 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.

DRAFT

6. fejezet

Helló, Welch!

6.1. Első osztályom

Valósítsd meg C++-ban és Java-ban az módosított polártranszformációs algoritmust! A matek háttér teljesen irreleváns, csak annyiban érdekes, hogy az algoritmus egy számítása során két normálist számol ki, az egyiket elspájzolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs 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/prog1/blob/master/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 kiírni az eredményt.

A programunk sokkal egyszerűbb módon van megírva C-ben. Ugyanis itt már eléggé 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/prog1/blob/master/welch/z3a7.cpp>

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

Preorder: gyökérrel indítunk, majd bejárjuk 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 ültess 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/prog1/blob/master/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 megírunk 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/prog1/blob/master/welch/z3a7.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 másképp 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/prog1/blob/master/welch/z3a7.cpp>

Itt használjuk a mozgató-konstruktorokat. A következő változásoknak kell végbemenniük 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->egyGyermek(), regifa)
        );
        ujelem->ujNullasGyermek(
            masol(elem->nullasGyermek(), regifa)
        );
        if ( regifa == elem )
            fa = ujelem;
    }
    return ujelem;
}
```

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:

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:

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 eszközkészlettel rendelkezik, és rengeteg lehetőségünk van kihasználni ezeket. A nagy eszközkészlet 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 irányt kaphatunk E-sport karrierünk felépítésében, kérdésében.

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 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...

9.3. Gimp Scheme Script-fu: név mandala

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

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

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 közelebbiek. Ezek nem értelmezhetők közvetlenül a számítógép által. Itt már szükségünk van egy fordítóra, ami lefordítja gépi nyelvre, ahhoz, hogy futtatható legyen.

Gépi nyelvezetű avagy 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 deklaratív 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. Imperatívok például az eljárásorientált nyelvek, vagy az objektumorientált nyelvek.

A deklaratív csoportba sorolható programkódoknál általában a programíró arra koncentrálnak, 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 ahhoz, hogy valaki jó programozóvá váljon, rengeteg programkódot kell átvészelnie mind elméletben, de legfőképpen gyakorlatban.

Utasítások:

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

Itt azért pár utasítás eléggé magától értendő. Mint például az értékadó 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 deklarációjától 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 készítése. A fent leírtak mindegyike érthető módon be van mutatva, le van egyszerű programokba bonyolítva, amelyek értelemszerűen a lehető legjobb megértetésre törekednek.

10.3. Programozás

[BMECPP]

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

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

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

DRAFT

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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