

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>

KÖZREMÜKÖDTEK

	<i>CÍM :</i> Univerzális programozás		
<i>HOZZÁJÁRULÁS</i>	<i>NÉV</i>	<i>DÁTUM</i>	<i>ALÁÍRÁS</i>
ÍRTA	Bátfai Norbert és Ignéczi Tibor	2019. május 7.	

VERZIÓTÖRTÉNET

VERZIÓ	DÁTUM	LEÍRÁS	NÉV
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
0.1	2019-03-03	Helló, Turing	sidragosam
0.2	2019-03-10	Helló, Chomsky	sidragosam
0.3	2019-03-16	Helló, Caesar	sidragosam
0.4	2019-03-27	Helló, Mandelbrot	sidragosam

VERZIÓTÖRTÉNET

VERZIÓ	DÁTUM	LEÍRÁS	NÉV
0.5	2019-04-04	Helló, Welch!	sidragosam
0.6	2019-04-13	Helló, Conway, Helló, Gutenberg	sidragosam
0.7	2019-04-22	Helló, Schwarzenegger, Helló, Chaitin	sidragosam
0.8	2019-05-01	Simítások, javítgatások, ellenőrzések	sidragosam
1.0	2019-05-06	Véglegesítés az 1.0-ás verzióra	sidragosam
1.1	2019-05-07	Hiányosságok pótlása.	sidragosam

DRAFT

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
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	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	11
2.6. Helló, Google!	12
2.7. 100 éves a Brun téTEL	13
2.8. A Monty Hall probléma	14
3. Helló, Chomsky!	16
3.1. Decimálisból unárisba átváltó Turing gép	16
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	16
3.3. Hivatalos nyelv	17
3.4. Saját lexikális elemző	17
3.5. l33t.l	18
3.6. A források olvasása	18
3.7. Logikus	21
3.8. Deklaráció	21

4. Helló, Caesar!	23
4.1. double ** háromszögmátrix	23
4.2. C EXOR titkosító	25
4.3. Java EXOR titkosító	27
4.4. C EXOR törő	28
4.5. Neurális OR, AND és EXOR kapu	31
4.6. Hiba-visszaterjesztéses perceptron	32
5. Helló, Mandelbrot!	33
5.1. A Mandelbrot halmaz	33
5.2. A Mandelbrot halmaz a std::complex osztállyal	37
5.3. Biomorfok	41
5.4. A Mandelbrot halmaz CUDA megvalósítása	44
5.5. Mandelbrot nagyító és utazó C++ nyelven	48
5.6. Mandelbrot nagyító és utazó Java nyelven	51
6. Helló, Welch!	55
6.1. Első osztályom	55
6.2. LZW	58
6.3. Fabejárás	63
6.4. Tag a gyökér	69
6.5. Mutató a gyökér	70
6.6. Mozgató szemantika	70
7. Helló, Conway!	77
7.1. Hangyszimulációk	77
7.2. Java életjáték	79
7.3. Qt C++ életjáték	80
7.4. BrainB Benchmark	81
8. Helló, Schwarzenegger!	83
8.1. Szoftmax Py MNIST	83
8.2. Mély MNIST	85
8.3. Minecraft-MALMÖ	85

9. Helló, Chaitin!	87
9.1. Iteratív és rekurzív faktoriális Lisp-ben	87
9.2. Gimp Scheme Script-fu: króm effekt	88
9.3. Gimp Scheme Script-fu: név mandala	93
10. Helló, Gutenberg!	100
10.1. Olvasónapló BRIAN W. KERNIGHAN és DENNIS M. RITCHIE A C programozási nyelv című könyvből	100
10.2. Olvasónapló Juhász István Magas Szintű Programozási Nyelvek 1 című könyvből	102
10.3. Olvasónapló a BME C++ könyvből	102
III. Második felvonás	105
11. Helló, Arroway!	107
11.1. A BPP algoritmus Java megvalósítása	107
11.2. Java osztályok a Pi-ben	107
IV. Irodalomjegyzék	108
11.3. Általános	109
11.4. C	109
11.5. C++	109
11.6. Lisp	109

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?

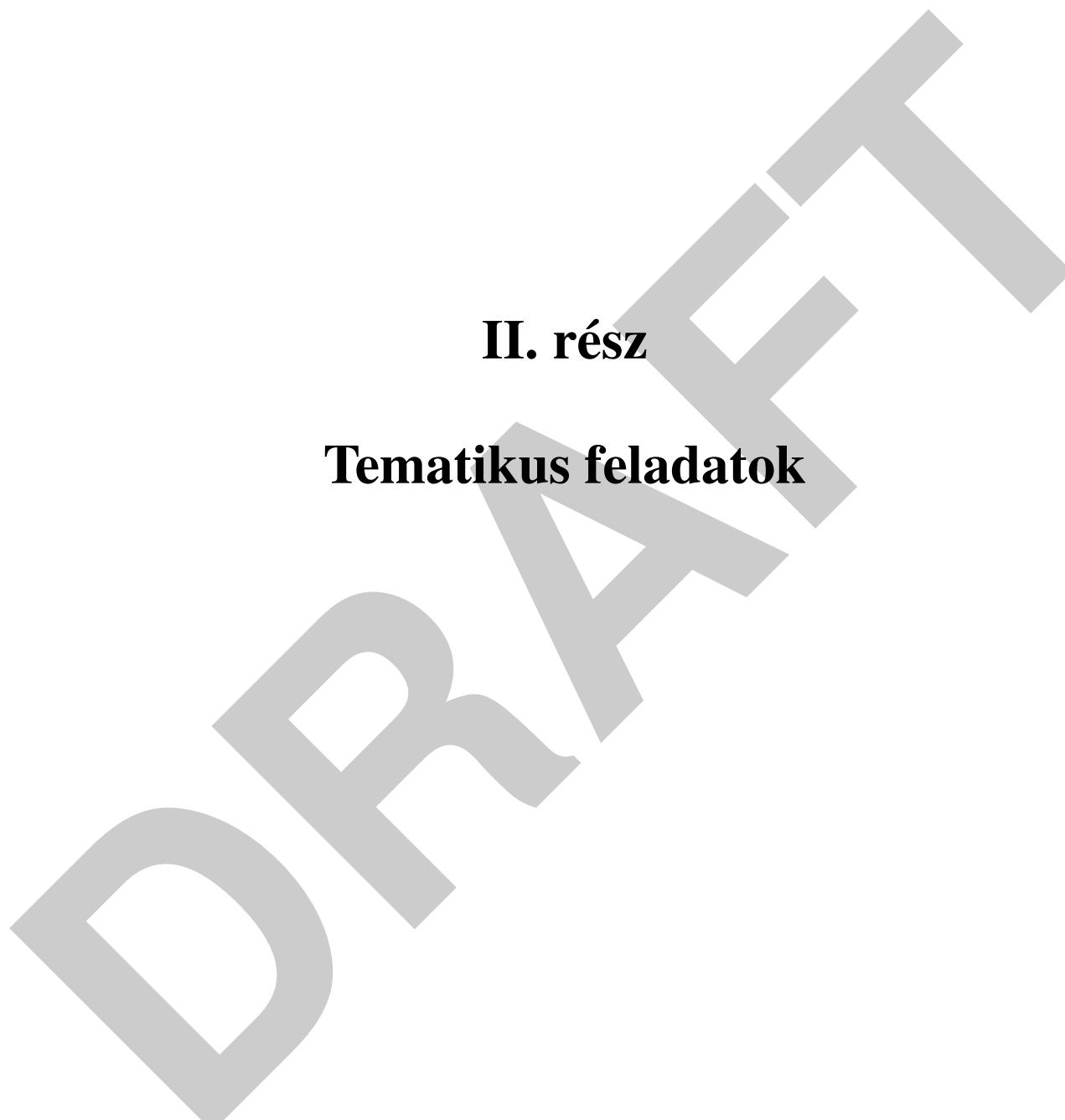
- Olvasgasd a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- [KERNIGHANRITCHIE]
- [BMECPP]
- Az igazi kockák persze csemegeznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok



**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: <https://youtube.com/watch?v=HMRHMBkbyeQ>

Megoldás forrása: A videóban látható a megoldás. 2. labor órán a feladatot sikeresen megcsináltuk így könnyű dolgom volt már.

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

int main()
{
/*0*/
/*  int i = 1;
while(i)
{
    printf("%d", i);
    sleep(i);
    i++;
}
/*100*/
/*  for(; ;){ } */

/*100 több szálon*/
/* #pragma omp parallel
for(; ;){ }*/
}
```

A forrásban jelöltem, hogy az adott ciklus hány százalékon terheli le a processzort. A legfelső eset 0%-on terheli a processzort, a második egy szálon 100%-ig, míg az utolsó több szálon 100%-on. Az utolsónál lefordítás esetén figyelni kell, hogy használjuk a -fopenmp kapcsolót. A nyelvünkben a // segítségével

kommenteket írhatunk ki egy sorban, `/**/` segítségével pedig több sorban, vagy akár egy sorban. Ahhoz, hogy egy egy kódot használni tudjunk a fenti forrásból ki kell törölünk a `/**/` elő és utótagokat. A futó programokat a CTRL+C lenyomásával tudjuk leállítani.

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

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

Megoldás videó:

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

```
Program T100
{
    boolean Lefagy(Program P)
    {
        if (P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a `Lefagy`-ra 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.

Nem tudunk ilyen programot írni. Tegyük fel, hogy meg tudnánk írni a programot, de akkor abba a hibába ütközénk, hogy minden programról el kellene tudnia döntenie (magáról is) a programnak, hogy megáll-e, vagy sem. Könnyen véiggondolhatjuk a két lehetséges inputtal a kísérlet kimenetelét. Ha a programunk megálló, akkor a futása során gyakorlatilag önmaga ismétlését csinálná újra és újra, így végtelen ciklusba kerülne a program, ha pedig feltesszük, hogy a program nem megálló, akkor eldönthetné magáról, hogy nem megálló, ezt követően megállna, tehát ha megálló akkor nem megálló, és ha nem megálló akkor megálló, ami ugye ellentmondás. Ez az ellentmondás pedig már elvben is lehetetlenné teszi, hogy ilyen programot írunk.

2.3. Változók értékének felcserélése

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

Megoldás videó: <https://youtu.be/Z-49Iuz-T-s>

Megoldás forrása:

```
#include <stdio.h>

int main()
{
    int a = 5;
    int b = 2;
    printf("A: %d, B: %d\n", a,b);
    a = a - b;
    b = b + a;
    a = b - a;
    printf("A: %d, B: %d\n", a,b);
    a ^= b;
    b ^= a;
    a ^= b;
    printf("A: %d, B: %d\n", a,b);
    a = a*b;
    b = a/b;
    a = a/b;
    printf("A: %d, B: %d\n", a,b);
}
```

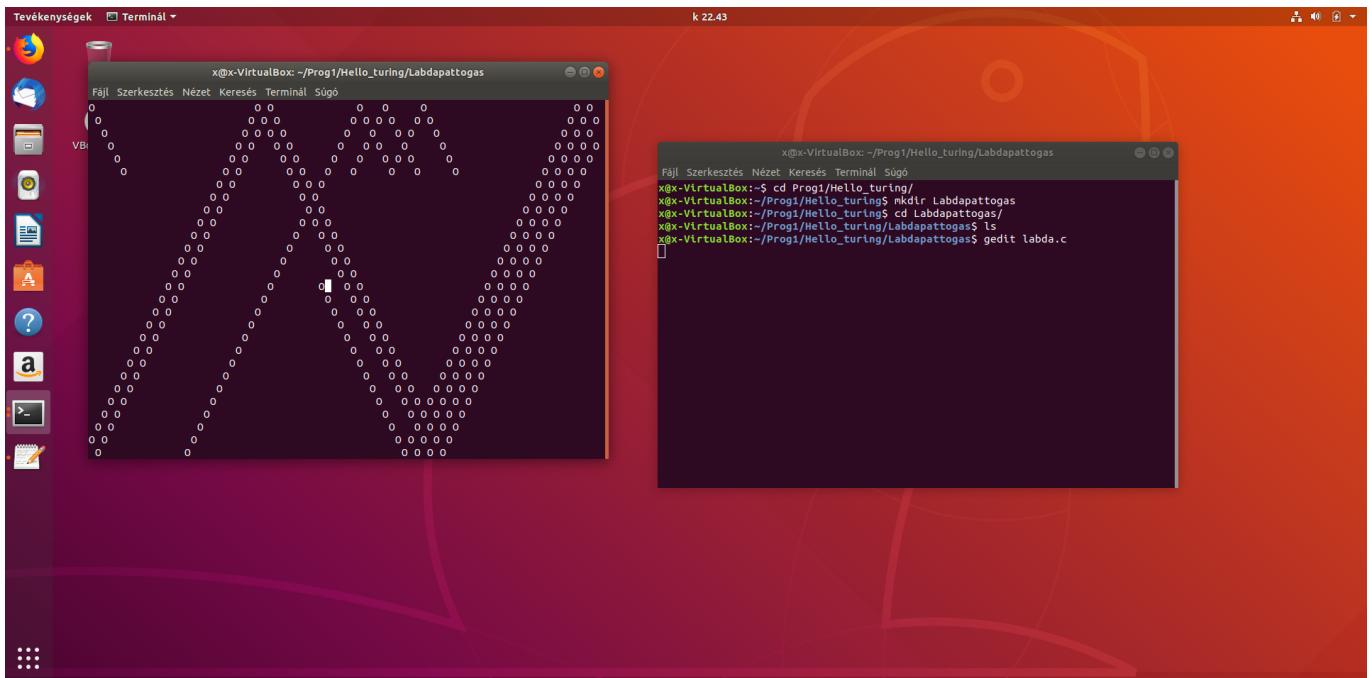
Tanulságok, tapasztalatok, magyarázat... Papíron is le lehet vezetni, hogy hogyan tudjuk a két változó értékét felcserélni. Próbáljuk ki papíron 5 és 2 értékekkel, majd addig számolgassunk míg fel nem tudjuk őket cserélni.

Ha papíron sikerült megoldani, akkor nézzük számítógépen a gyakorlatban. Ha folyamatosan ellenőrizni szeretnénk magunkat, akkor a printf-et minden értékváltozás után helyezzük be. $a = a - b$, ilyenkor az történik, hogy az a felveszi az $a(5) - b(2)$ értékét, vagyis $a = 3$, következő lépésnél b vesz fel értéket és megkapja a $b(2)+a(3)$ összeget, ekkor a b értéke már 5 lesz. Utolsó lépésként a jelenlegi b-ből vagyis az 5-ből kivonjuk az a-t, ami ugye 3, így megkapjuk a értékének a 2-t. Sikerült felcserélni? Igen sikerült.

2.4. Labdapattogás

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

Megoldás videó: <https://youtu.be/vF6P26WHeiw>



Megoldás forrása: if-el:

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

//-lncurses
int main ()
{
    WINDOW *ablak;
    ablak = initscr ();
    int x = 0;
    int y = 0;
    int xnov = 1;
    int ynov = 1;
    int mx;
    int my;

    for ( ; ) {
        getmaxyx ( ablak, my , mx );
        mvprintw ( y, x, "O" );
        refresh ();
        usleep ( 100000 );
        clear();
        x = x + xnov;
        y = y + ynov;
        if ( x>=mx-1 ) { // elérte a jobb oldalt?
            xnov = xnov * -1;
        }
        if ( x<=0 ) { // elérte a bal oldalt?
            xnov = xnov * -1;
        }
    }
}
```

```
    if ( y<=0 ) { // elétre a tetejét?
        ynov = ynov * -1;
    }
    if ( y>=my-1 ) { // elérte az alját?
        ynov = ynov * -1;
    }

}
return 0;
}
```

Az if-es változatban láthatjuk, hogy ifek segítségével ellenőrizzük, hogy a labda mikor éri el az adott széleket, majd ha eléri valamelyik, határt akkor megváltoztatjuk az előjelet és így a labda a másik irányba kezd majd el haladni.

if nélkül:

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

int main ()
{
    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);
    int mx;
    int my;
    for (;;) {
        getmaxyx ( ablak, my , mx );
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;
        yj = (yj - 1) % my;
        yk = (yk + 1) % my;
        clear ();
        mvprintw (abs ((yj + (my - yk)) / 2),
                  abs ((xj + (mx - xk)) / 2), "O");
        refresh ();
        usleep (150000);
    }
    return 0;
}
```

Az if nélküli változatban a maradékos osztás segítségével tudjuk eldönteneni, hogy a labda mikor tartózkodik az adott határokon belül, vagy épp mikor éri el a határt. Fordításnál ügyelni kell arra, hogy az -lncurses

kapcsolót használjuk.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

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

Megoldás forrása:

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

void
delay (unsigned long long loops)
{
    for (unsigned long long i = 0; i < loops; i++);
}

int
main (void)
{
    unsigned long long loops_per_sec = 1;
    unsigned long long ticks;

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

        if (ticks <=1)
        {
            printf("Gépi szó hossza: %llu \n",loops_per_sec);
            return 0;
        }
    }

    printf ("failed\n");
    return -1;
}
```

Linus Torvald eredeti BogoMips rutinja kis módosításokkal alkalmas a feladat megoldására. Meg kell néznünk, hogy a delay() futási ideje mikor lépi át a CLOCKS_PER_SEC értékét 1,000,000-t, míg a módosításuk után 1 tick idő alatt futást akar az if.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

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

void kiir(double tomb[], int db)
{
    int i;
    for(i=0;i<db;i++)
        printf("Pagerank [%d]:\n %.2f\n", i, tomb[i]);
}

double tavolsag(double pagerank[], double pagerank_temp[], int db)
{
    int i;
    double sum = 0;
    for(i=0;i<db;i++)
        sum+=(pagerank_temp[i] - pagerank[i]) * (pagerank_temp[i] - pagerank[i]);
    return sqrt(sum);
}

int main()
{
    double L[4][4] =
    {
        {0.0, 0.0, 1.0 / 3.0, 0.0},
        {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
        {0.0, 1.0 / 2.0, 0.0 , 0.0},
        {0.0, 0.0, 1.0 / 3.0 , 0.0},
    };

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

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

    for(;;)
    {
        for(i=0;i<4;i++)
            PR[i] = PRv[i];
        for(i=0;i<4;i++)
        {
            for(j=0;j<4;j++)
                for(h=0;h<5;h++)
                    if(j==h)
                        PR[i] += L[i][j];
                    else
                        PR[i] += L[i][j] / 4.0;
        }
        for(i=0;i<4;i++)
            PRv[i] = PR[i];
    }
}
```

```
    double t = 0;
    for(j=0; j<4; j++)
        t+=L[i][j]*PR[j];
        PRv[i]=t;
    }
    if(tavolsag(PR, PRv, 4) < 0.00001)
        break;
}
kiir(PR, 4);
return 0;
}
```

A pagerank kiszámolja, hogy melyik oldalra hány oldal mutat és amelyikre a legtöbb mutat, tehát a legnagyobb pagerankkal rendelkezik az kerül a legelső helyre. Maga a programunk a következőképp épül fel: Tartalmaz egy kiír függvényt, amivel majd kiratjuk a pagerank értékeitet, tartalmaz egy távolság függvényt, ahol az oldalak közötti "távolságot" számolhatjuk ki, valamint tartalmazza a main függvényt ugye, amiben meghívjuk ezeket a későbbiekben. Nos mielőtt enniyre előrehaladnánk nézzük meg a main felépítését. Létrehozunk egy 4x4-es tömböt, amiben tároljuk az L-hez tartozó értékeit. Ezután létrehozunk 2 db 4x1-es tömböt, ahol a PR és a PRv értékeit tároljuk. Szükségünk van 3 db változóra, az i-re a j-re és a h-ra. Ezeket egy for ciklusban fogjuk használni. Elindítunk egy for ciklust, amelyet az első feladatban már alkalmaztunk, majd a for cikluson belül for ciklusokkal "elküldjük" az i-t 0-tól a 4-ig, hogy a PR-re beállítsuk a PRv értékeit, majd egy újabb for ciklusban L értékeit szorozzuk az új PR értékekkel. Itt létrehozunk egy temporary változót, amit majd végül a PRv értékét fogja beállítani. A matematikai műveletek jól láthatóak és könnyedén megérthetőek a programunkban. Szükség van még egy if-re, amiben beállítjuk, hogy ha a PR és PRv távolsága kisebb, mint 0.00001 akkor állítsa meg a for ciklust. Az örökké tartó for cikluson kívül pedig meghívjuk a kiír függvényünket.

2.7. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

Definánunk kell egy függvényt, ami az ikerprímek reciprokosszegét fogja visszaadni. Meg kell adnunk egy listát, ami tartalmazza az összes prímet x-ig. Ezekkel kivonást végezünk, míg nem találunk egy olyan párt melynek a különbsége 2. Ekkor mondhatjuk, hogy ez a két prím szám ikerpríme egymásnak. Ez végül visszatérési értékként a számok reciprokosszegét fogja megadni. Ahhoz, hogy ábrázolni tudjuk szükségünk lesz egy koordinátarendszerre, amelyben az x tengelyt felosztjuk 13-tól 1000000-ig tízezresével, az y értékekkel pedig az előbb használt függvényünk fogja meghatározni. A plot függvény segítségével kirajzolhatjuk az ábrát, ami demonstrálni fogja nekünk a Brun téTELt.

```
library(matlab)

stp <- function(x){

    primes = primes(x)
    diff = primes[2:length(primes)]-primes[1:length(primes)-1]
```

```
idx = which(diff==2)
t1primes = primes[idx]
t2primes = primes[idx]+2
rt1plust2 = 1/t1primes+1/t2primes
return(sum(rt1plust2))
}

x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan

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

A Monty Hall probléma egy műsoron alapszik. Ebben a műsorban a játékosoknak három ajtó közül kellett választaniuk. Kettő ajtó mögött értéktelen dolog lapult, egy ajtó mögött viszont egy értékes nyeremény. Amikor egy játékos kiválasztott egy ajtót, akkor Monty kinyitott egy másik ajtót, ami mögött egy értéktelen dolog lapult. Ezt követően a játékos eldönthette, hogy marad-e az eredeti kiválasztott ajtó mellett, vagy inkább vált. A legtöbb ember úgy gondolná, hogy nem járunk jobban azzal, ha váltunk, hiszen ígyis-úgyis 50% esélyünk van a jó ajtó kiválasztására, de ha jobban belegondolunk akkor az esélyeink $1/3 - 2/3$ arányban állnak a váltás javára. Az alábbi program demonstrálja is nekünk az esetet. Hosszú futási idő után kapunk egy eredményt 10000000 kísérletre lebontva, ami ha minden igaz akkor $1/3 - 2/3$ lesz az arány a váltás javára. A programot átírhatjuk könnyedén, hogy a kiserletek_szama változót megnöveljük, vagy csökkentjük és akkor több/kevesebb esetet is megvizsgálhatunk.

```
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(holvart),1)]
```

```
}
```

```
valtoztatesnyer = which(kiserlet==valtoztat)
```

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

```
length(nemvaltoztatesnyer)
```

```
length(valtoztatesnyer)
```

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

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



3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:

```
#include <stdio.h>

int main()
{
    int szam;
    printf("Írj be egy számot:\n");
    scanf("%d", &szam);
    printf("Az átalakítandó szám: %d\n", szam);
    for(int i = 0; i < szam; i++)
    {
        printf("1");
    }
    printf("\n");
}
```

Az unáris, vagyis az egyes számrendszerben az 1-esek száma jelzi az adott szám értékét. Tehát egy szám decimálisban, például a 3, unárisban úgy néz ki, hogy 111, tehát úgy számolhatjuk ki, hogy 1+1+1, ami végül hármat ad. A programunk a for ciklus segítségével számolja a megadott decimális számot, majd annyi 1-est ír le amennyit megadtunk neki.

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

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

Megoldás forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_1.pdf?fbclid=IwAR0vO

Tisztázzunk két fogalmat, generatív grammatika, illetve környezetfüggetlen nyelvtan. Noam Chomsky nevéhez köthető a generatív grammatika elmélete. Az elmélet alapja, hogy egy kezdőszimbólum, terminálisszimbólumok, nemterminális szimbólumok és előállítási szabályok segítségével végtelen szó alkotható

az adott nyelvben. A generatív grammaikánál különböző osztályokról beszélhetünk. Ide tartozik a környezetfüggetlen és a környezetfüggő generatív grammaika. Az első kizárálag annyi feltételt szab, hogy az előállítási szabály bal oldalán csak nemterminális állhat, jobb oldalán pedig terminálisokból és nemterminálisokból álló szó, mely akár üres is lehet. A környezetfüggetlen nyelvek ezzel szemben meghatározák, hogy a szabály bal oldalán egy terminálisokból és nemterminálisokból álló szónak kell állni, melynek hossza nagyobb nullánál. A jobb oldal pedig tartalmaz legalább egy nemterminálist és hossza $>=$ mint a bal oldal hossza.

Példa környezetfüggetlen grammaikára mely az adott nyelvet generálja: S, X, Y „változók” a, b, c „konsztansok” $S \rightarrow abc$, $S \rightarrow aXbc$, $Xb \rightarrow bX$, $Xc \rightarrow Ybcc$, $bY \rightarrow Yb$, $aY \rightarrow aaX$, $aY \rightarrow aa$ S-ből indulunk ki Környezetfüggetlen: $X \rightarrow P$, X VN beli, P (VN U VT)* beli,

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:

```
#include <stdio.h>

int main()
{
    for (int i = 10; i < 20; i++) {
        printf("i: %d\n", i);
    }

    return 0;
}
```

A C89-nél még a for cikluson belül nem lehetett használni a deklarálást, majd később az újabb C99-es verzióban bevezették ezt a funkciót is, megkönnyítve a for ciklusok létrehozását. Fordításhoz: -std=c89 kapcsolóval fordíthatjuk C89-ben, -std=c99-el pedig C99-ben. A fent található kód C89-es verzióban nem fog tudni lefutni, viszont az újabban már igen. Ezen kívül a c89 még nem támogatja a megjegyzés használatára a //t, helyette /**/t használhatunk.

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 állunk és ne kispályázzunk!

Megoldás forrása:

```
% {
#include <string.h>
int betuk_szama = 0, szavak_szama = 0, szamok_szama = 0, sorok_szama = 0;
```

```
%}  
%%  
. ++betuk_szama;  
\n ++sorok_szama;  
[0-9]+ {++szavak_szama; ++szamok_szama,  
    printf("szam=[%s]", yytext);  
    betuk_szama += strlen(yytext) ;}  
[a-zA-Z] [a-zA-Z0-9]* {++szavak_szama; betuk_szama += strlen(yytext);}  
%%  
int main()  
{  
    yylex();  
    printf("%d betű, %d szó, %d szám, %d sor\n", betuk_szama, szavak_szama, ←  
        szamok_szama, sorok_szama);  
    return 0;  
}
```

Ez a program nem csak a számokat nézi meg, hanem a betűk számát, a szavak számát, a számok számát, illetve a sorok számát is. Az első pár sorba kerültek a deklarációk. Itt értékeket adunk nekik, hogy 0 legyen minden egyik. Ezután megadtuk, hogy a bejövő karakterekre a program miként reagáljon. Ide a tartozik a számok és betűk helyes kezelése, a pont karakter leüttésének eltárolása. Az enter leütése után kiértékeljük, hogy hány darab betű, szó, szám, illetve sor került beírásra. Szabályok:

```
. ++betuk_szama;  
\n ++sorok_szama;  
[0-9]+ {++szavak_szama; ++szamok_szama,  
    printf("szam=[%s]", yytext);  
    betuk_szama += strlen(yytext) ;}  
[a-zA-Z] [a-zA-Z0-9]* {++szavak_szama; betuk_szama += strlen(yytext);}
```

3.5. I33t.l

Lexelj össze egy l33t cipher! -> l3x3l_ össz3 3gy lEEt (1ph3rt!

Tutor: Ádám Petra

Megoldás forrása: ../Hello_chomsky/l33t/l33t.l

A l33t-et úgy kell elképzelniink, hogy bizonyos betűket számokkal, vagy más szimbólumokkal helyettesítünk. Például a leetből így jött a l33t, vagy az 1337. A kódban fent defináljuk, hogy egyes karakterekre, hogyan reagáljon a program. Azaz például az a betű beírásakor használjon 4-est @-ot, vagy /-v karaktert. Ezután véletlenszerűen eldönti a program, hogy a megadott karakterek közül melyiket fogja használni. Az inputra érkező karaktereket a szabálynak megfelelően átalakítja. Például: Debreceni Egyetem -> d3br3(3n1 3gy3t3m

3.6. A források olvasása

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

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

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

Bugok

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

i.

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

Ha a SIGINT jel nem volt figyelmen kívül hagyva, akkor a jelkezelő függvény kezelje, ha figyelmen kívül volt hagyva, akkor pedig hagyja továbbra is figyelmen kívül hagyva.

ii.

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

Egy for ciklus, ami 0-tól indul és az 5-ig tart.

iii.

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

Egy for ciklus, ami 0-tól indul és az 5-ig tart, annyi különbséggel, hogy itt "hátul" növeli az i értékét. A hátul növelés alatt azt értjük, hogy az i++ esetén először felveszi az i értéket, majd megnöveli egyel, ++i esetén pedig először növeli meg egyel és csak utána veszi fel az értéket.

iv.

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

Egy for ciklus, ami 0-tól indul és 5-ig tart. Van egy tömbünk, ami folyamatosan eltárolja az i értékküket. Ez egy bugos kód, mert nincs jól leírva a végrehajtás sorrendje.

v.

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

Egy for ciklus, ami 0-tól indul és n-ig tart és addig amíg a *d++ visszatérési értéke érvényes. (Az "n" egy előre deklarált érték)

vi.

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

A printf függvényünk kiírat kettő darab int típusú változót, úgymond egész számot, amit a két hátsó függvény, az f(a, ++a) és az f(++a, a) ad vissza. Ez egy bugos kód, mert nincs megadva, hogy milyen sorrendben legyen kiértékelve.

vii.

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

A printf függvényünk kiírat kettő darab int típusú változót, először az "f" függvény visszatérési értékét, majd pedig az "a" változót.

viii.

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

A print függvényünk kiírat kettő darab int típusú változót, először az "f" függvény visszatérési értékét, ami egy memóriacímen van tárolva, majd pedig az "a" válozót.

Megoldás forrása:

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

void jelkezelő(int a)
{
    printf("Elkaptam a jelet: %d", a);
}

int main()
{
    for(int i=0; i<5; ++i)
    {
        printf("%d", i);
    }
}
```

Egy másik lehetőség:

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

void sighthandler(int);

int main () {
    signal(SIGINT, sighthandler);

    while(1) {
        printf("Alszik...\n");
        sleep(1);
    }
    return(0);
}

void sighthandler(int signum) {
    printf("Elkaptam a jelet: %d\n", signum);
    exit(1);
}
```

A programba beillesztjük a különböző megadott sorokat, majd ellenőrizzük, hogy működik-e. Az első sorokban ugye a szokásos includeolással kezdünk, majd létrehozunk egy sighthandler függvényt. A main-ben a bejövő jeleket érzékeli a függvény, ha nem történik semmi akkor kiírja, hogy alszik és altatja az első

szálat. A sighandler függvénynél ahová már signum integrer-t is társítunk kiírja, hogy elkapta a jelet, majd exitel belőle.

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

1. minden x esetén van olyan y, ami nagyobb, mint x és y prím. → Végtelen prímszám van.
2. minden x esetén van olyan y, ami nagyobb, mint x és y ikerprím. → minden számnál vannak nagyobb ikerprímek.
3. Van olyan y, ami minden x esetén (ahol x prím), igaz, hogy x kisebb, mint y. → Van olyan szám ami nagyobb minden prímnél.
4. Van olyan y, ami minden x esetén y kisebb, mint x akkor x nem prím. → Ha létezik olyan y, ami minden x-nél kisebb, akkor x nem prím.

3.8. Deklaráció

Vevezd 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 mutatót visszaadó függvény
- egészre mutató mutatót visszaadó függvényre mutató mutató
- egészet visszaadó és két egészet kapó függvényre 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;`

Az "a" egy egész szám.

- `int *b = &a;`

A "b" egy mutató, ami az "a" egész számra mutat.

- `int &r = a;`

Az "r" referencia az "a" egész számra.

- `int c[5];`

A "c" egy 5 dimenziós tömb, melynek elemei egész számok.

- `int (&tr)[5] = c;`

A "tr" egy ötdimenziós referencia a "c" 5 dimenziós tömbre.

- `int *d[5];`

A "d" egy 5 dimenziós tömb, melynek elemei egész számra mutató mutatók.

- `int *h();`

A "h" egy függvény, melynek visszatérési értéke egy egész számra mutató mutató.

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

Az "l" egy függvény, melynek visszatérési értéke egy egész számra mutató mutató, továbbá az "l"-re is mutatnak mutatók.

- `int (*v(int c))(int a, int b)`

A "v" egy függvény, melynek visszatérési értéke egy "c" nevű egész számra mutató mutató és "a" és "b" egész számok.

- `int (*(*z)(int))(int, int);`

A "z" egy függvény, melynek visszatérési értéke egy egész számra mutató mutató lenne, ha értelmezhető lenne, továbbá a "z"-re is mutatnak mutatók. Továbbá van két int, név nélkül.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás forrása:

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

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

    printf("%p\n", &tm);

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

    printf("%p\n", tm);

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

    printf("%p\n", tm[0]);

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



```
for (int j = 0; j < i + 1; ++j)
    tm[i][j] = i * (i + 1) / 2 + j;

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

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

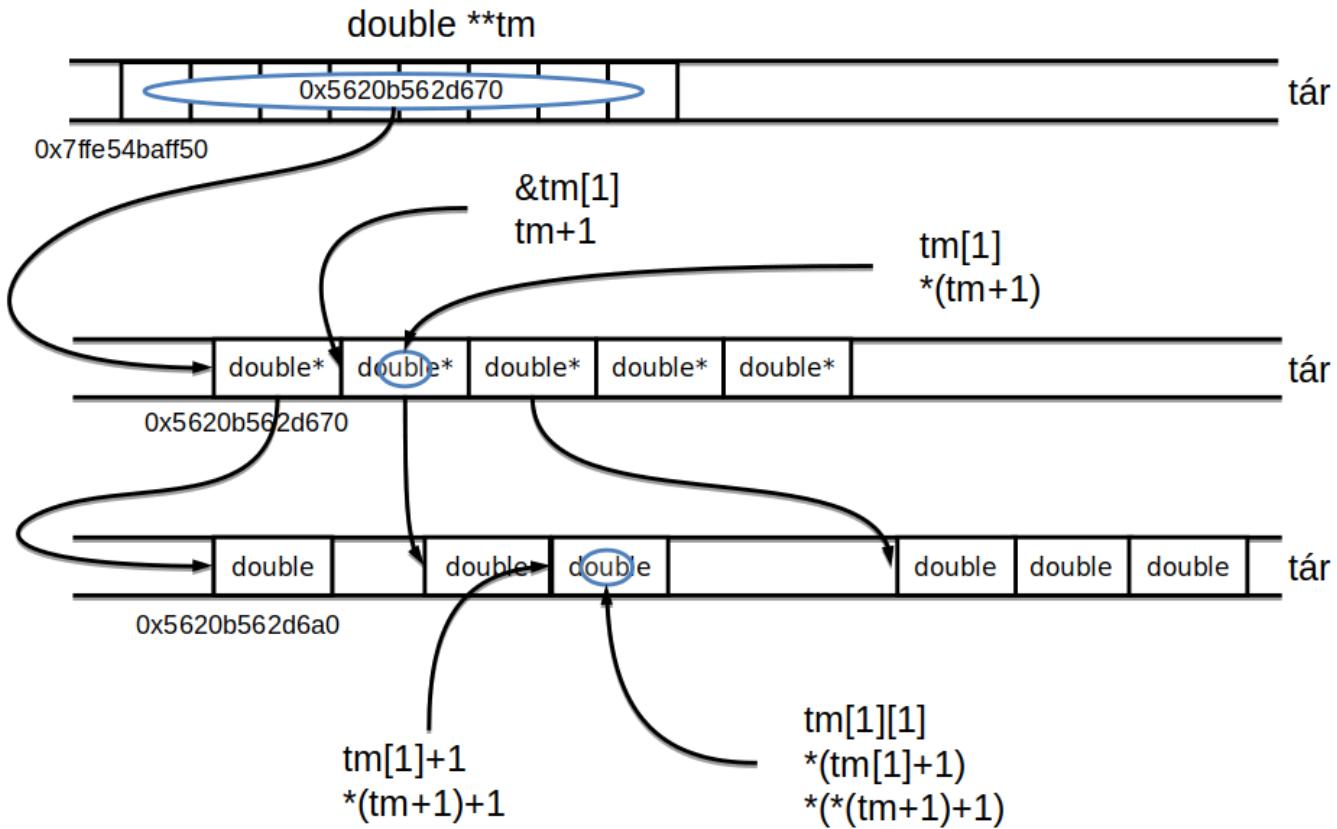
for (int i = 0; i < nr; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tm[i][j]);
    printf ("\n");
}

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

free (tm);

return 0;
}
```





Kép forrása: [Bátfai Norbert](#)

A háromszögmátrix olyan kvadratikus, azaz négyzetes mátrix (kvadratikus mátrix: olyan mátrix, amelyben a sorainak és oszlopainak a száma megegyezik), melynek a főátlója alatt, vagy felett csupa nulla szerepel. A malloc függvény miatt legfelül az stdlib.h headert includeolni kell. Ezután következik a deklaráció. Deklarálunk egy double típusú pointerre mutató pointert. Az if-es részben a malloc függvényt használjuk, ami helyet foglal a memóriában és egy pointert ad vissza. A malloc megkapja, hogy mekkora területet foglaljon le `nr*8`, ami esetünkben $5*8$, azaz 40 bajt. Az if-el vizsgáljuk, hogy sikeresen megtörtént-e a helyfoglalás és, hogy a malloc visszaad-e a `double*`-okra egy mutatót. Ha sikeres volt, akkor a program továbbhalad, ha viszont nem volt sikeres, akkor kilép a programból. A következő szakaszban egy for ciklust látunk, ami 0-tól `nr`-ig megy, ami esetünkben 0-tól 5-ig. Ezen belül memóriát foglalunk a `tm[i]`-edik elemének (az elsőnek 8 bajtot, aztán 16-ot, 24-öt és így tovább), amiről visszad a malloc egy pointert. Persze itt is ellenőrizve van, hogy sikeres-e a helyfoglalás és pointer létrehozás. Elérkeztünk a mátrixunk megszerkesztéséhez. Két egymásba ágyazott for ciklus segítségével tudunk végigmenni az elemeken. Itt feltöljtük a mátrixot, majd a következő, szintén egymásba ágyazott for ciklus segítségével íratjuk ki mátrix értékeit. Láthatjuk, hogy a 4. sorban lévő értékeket, hogyan változtathatjuk meg. Ezután újra kiíratjuk a mátrixunkat és futtatás után kiderül, hogy tényleg megváltoztak az utolsó sorban az értékek. A program legvégén érünk el a feladat leírásában is említett free függvényhez. Ennek segítségével tudjuk felszabadítani már korábban `tm` által lefoglalt lefoglalt memóriát.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Tutoriált: Ádám Petra, Tuba Adrienn

Megoldás forrása:

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

    char kulcs[MAX_KULCS];
    char buffer[BUFFER_MERET];

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

    while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
    {

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

    {
        buffer[i] = buffer[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;

    }

        write (1, buffer, olvasott_bajtok);

    }
}
```

Az XOR művelet bitenként hasonlít össze két operandust. Kétféle értéket adhat vissza, attól függően, hogy a vizsgált két bit megegyezik-e vagy sem. Ha megegyeznek, akkor 0-t ad vissza, ha nem akkor 1-et. Jelen esetben ez a két operandus a forrás bemenet, amit titkosítani akarunk és egy kulcs, amire a titkosításhoz van szükség. A programnak adnunk kell egy tiszta fájlt, amit le fog titkosítani nekünk. Ezt a szöveges fájlt a következőképp titkosíthatjuk:

```
./e chocapic <tiszta.txt >titkos.txt
```

Az 'e' a már compilelt fájlunk neve, ugye futtatni szeretnénk. A 'chocapic' a titkosítási kulcsunk, a tiszta.txt a bemeneti állomány, a titkos.txt pedig az ebből keletkező titkosított állomány. Visszafelteni a 'chocapic' kulcsszó megadásával tudjuk a következőképp:

```
./e chocapic <titkos.txt
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

```
public class javatitk {

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

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;
        while((olvasottBájtok =
               bejövőCsatorna.read(buffer)) != -1) {

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

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

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {
        try {
            new javatitk(args[0], System.in, System.out);
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
}
```

A fordításhoz telepítenünk kell egy csomagot:

```
sudo apt install default-jdk
```

A java kódot a következőképpen fordítjuk:

```
javac javatitk.java
```

Futtatás:

```
java javatitk chocapic > titkos.txt
```

Itt is a chocapic lesz a kulcs és a titkos.txt' nevű fájlba irányítjuk a szöveget, amit titkosítunk. A parancs lefuttatása után a terminálba egépeljük, amit titkosítani szeretnénk, majd Ctrl+D. Ezután elkészül a 'titkos.txt' fájlunk.

A titkosító töréséhez szükség van a kulcsra. Az alábbi módon tudjuk törni a titkosított szöveget:

```
java javatitk chocapic < titkos.txt
```

Hasonló ez a feladat, mint az előbbi. Itt is titkosítanunk kell egy tiszta szöveget EXOR-ral, de most nem C-ben, hanem Java-ban. Az egész kódunk egy jó nagy class-ból áll, aminek két nagyobb része van. Kezdjük a main résszel. A main itt máshogyan néz ki, mint ahogy azt C-ben már megszoktuk. Ráadásul itt láthatunk először példát a kivételkezelésre. (try, catch) A try és a catch használata, nem csak Java-ban, hanem C++-ban is igen elterjedt. Hiba esetén a try "dobja", a catch "elkapja" a hibát és küld egy hibaüzenetet a terminálba. A javatitk függvényen belül utasításokat hajtunk végre, olyanokat amiket már C-ben is csináltunk. Létrehozunk egy byte-okból álló tömböt. A getBytes() függvény segítségével olvassuk be a kulcsot a kulcs tömbbe. A buffer tömbnek ugyanúgy, ahogy a korábbi C-s feladatban, 256 bájtból álló területet foglalunk. Innen már nagyon hasonlóan működik a program, mint a C-s testvére. A while cikluson belül itt is található egy for ciklus, ahol elemenként össze EXOR-ozzuk a buffer tartalmát a kulccsal. Végül kiíratjuk a puffer tartalmát.

4.4. C EXOR törő

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

Tutoriált: Ádám Petra

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

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

double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
```

```
int sz = 0;
for (int i = 0; i < titkos_meret; ++i)
    if (titkos[i] == ' ')
        ++sz;

return (double) titkos_meret / sz;
}

int
tiszta_lehet (const char *titkos, int titkos_meret)
{
    // a tiszta szöveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az átlagos szóhossz vizsgálatával csökkentjük a
    // potenciális töréseket

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

    int kulcs_index = 0;

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

        titkos[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;

    }

}

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

    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);

}
```

```
int
main (void)
{
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;

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

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

    // osszes kulcs eloallitasa
    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;

                                    if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
                                        printf("Kulcs: [%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
                                               ii, ji, ki, li, mi, ni, oi, pi, titkos);

                                    // ujra EXOR-ozunk, igy nem kell egy masodik buffer
                                    exor (kulcs, KULCS_MERET, titkos, p - titkos);
                                }

    return 0;
}
```

A program lényege, hogy a titkosított fájlokat fel tudjuk törni, avagy vissza tudjuk alakítani a titkos.txt-t tiszta.txt-vé. A program elején lévő _GNU_SOURCE új lehet számunkra. Erre a strcasestr használata miatt van szükség. Láthatjuk, hogy a kulcsméret 8-ra van állítva, azaz feltételezzük, hogy a kulcs 8 elemből áll. Az átlagos_szohossz függvénytel kiszámítjuk a bemenet átlagos szóhosszát. Majd a tiszta_lehet függvény megvizsgálja, hogy a fejtésben lévő kód tiszta-e már. Itt elérkeztünk a programunk egy újabb gyengeségéhez, ugyanis a program feltételezi, hogy a tiszta szöveg valószínűleg tartalmazza a gyakori magyar szavakat, illetve az átlagos szóhossz vizsgálatával akarja csökkenteni a lehetséges töréseket. Ha ezeknek nem felel meg a tiszta szöveg, akkor nem tudjuk feltörni. Az exor függvény hasonlóan működik, mint a titkosításnál. Ezáltal visszakapjuk a tiszta szöveget, elvégre ha valamit duplán EXOR-ozunk, akkor önmagát kapjuk. Végül elérünk az exor_tores függvényhez, ami 0-át valamint 1-et ad vissza, a szöveg tisztaságától függően. A main-en belül elvégezzük a szükséges deklarációkat, majd egy while ciklussal folyamatosan olvassuk a bajtokat, a bemenet végéig, vagy amíg a bufferünk tele nem lesz. A következő for ciklussal azokat a helyeket, amik megmaradtak a bufferben kinullázuk. Majd jön egy időigényes rész, ahol az összes lehetséges kulcsot előállítjuk. A végén meghívjuk az exor_tores függvényt, aminek ha 1 a visszatérési értéke, akkor kiírja a program a kulcsot és a már feltört szöveget.

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 feladat megoldásához telepítenünk kell a neralnet-et. Ennek segítségével építhetjük fel a neurális hálónkat. A neurális hálónk egy kimeneti és kettő darab bemeneti neuronnal fog operálni. Ahhoz, hogy tanítsuk a programot adjunk meg neki három vektort. A vektorban az oszlopok összetartoznak, egy számhármast alkotnak, aminek a harmadik értéke az első kettő között végzett logikai OR eredménye. Ennek segítségével írja majd ki nekünk a program a súlyokat. Végezetül pedig kirajzolatjuk a hálónkat..

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR      <- c(0,1,1,1)

or.data <- data.frame(a1, a2, OR)

nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE,   ←
                  stepmax = 1e+07, threshold = 0.000001)

plot(nn.or)

compute(nn.or, or.data[,1:2])
```

Ez a neurális háló meg tudja oldani a logikai OR műveletet, majdnem pontosan. A feladat lényege, hogy a program úgy oldja meg a neki kiadott feladatot, hogy nem mondta el neki, hogy hogyan kellene megtennie ezt. Tulajdonképpen példákkal "okosítottuk" fel a programot, ami ezen példákon tanulva sikeresen megoldja a feladatot és még olyan "kiskapukra" is kitér, amik eddig számunkra ismeretlenek voltak. Végül elmondhatjuk, hogy a programunk algoritmusok segítsége nélkül saját maga oldja meg a feladatot. A logikai ÉS és EXOR műveleteket elvégző neurális hálók az elsőhöz hasonlóan építhetők fel.

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
OR      <- c(0,1,1,1)
AND     <- c(0,0,0,1)

operand.data <- data.frame(a1, a2, OR, AND)

nn.operand <- neuralnet(OR+AND~a1+a2, operand.data, hidden=0, linear.output= FALSE,
                        stepmax = 1e+07, threshold = 0.000001)

plot(nn.operand)

compute(nn.operand, operand.data[,1:2])


a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR   <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=0, linear.output=FALSE,
                      stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

compute(nn.exor, exor.data[,1:2])
```

4.6. Hiba-visszaterjesztéses perceptron

C++

Mi is az a perceptron? Gyakorlatilag egy egy rétegből álló neurális háló. Működése nagyon egyszerűen leírható. A megadott input adatokat megszorozzuk a hozzájuk rendelt súlyval. Ezekkel a súlyokkal tudjuk meghatározni hogy az output szempontjából mely inputok lesznek meghatározóbbak és melyek kevésbé. Ezután a súlyozott input értékeit összegezzük. Az így kapott súlyozott összegen végül alkalmazzuk az aktivációs függvényt. Ez a lépés teszi lehetővé hogy az adatok könnyebben kiértékelhetőek legyenek. Leggyakrabban olyan függvénykete alkalmazunk, amelyek gyorsan 0 vagy 1 felé tartanak, így az outputot egy kis kerekítés után bináris értékben kapjuk.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Tutor: Egyed Anna

Megoldás forrása:

```
// mandelpngt.c++
// 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 <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
#define ITER_HAT 32000

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

    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    // 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;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 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 elértük, 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 < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
```

```
ujreZ = reZ * reZ - imZ * imZ + reC;
ujimZ = 2 * reZ * imZ + imC;
reZ = ujreZ;
imZ = ujimZ;

++iteracio;

}

kepadat [j] [k] = iteracio;
}

}

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;

}

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

if (argc != 2)
{
    std::cout << "Hasznalat: ./mandel fajlnev";
    return -1;
}

int kepadat [MERET] [MERET];

mandel (kepadat);

png::image < png::rgb_pixel > kep (MERET, MERET);

for (int j = 0; j < MERET; ++j)
{
    //sor = j;
    for (int k = 0; k < MERET; ++k)
    {
        kep.set_pixel (k, j,
                      png::rgb_pixel (255 -
                                      (255 * kepadat [j] [k]) / ITER_HAT ←
                                      ,
                                      255 -
                                      (255 * kepadat [j] [k]) / ITER_HAT ←
                                      ,
                                      )
```

```
    255 -  
    (255 * kepadat[j][k]) / ITER_HAT ←  
    ) );  
}  
  
}  
  
kep.write (argv[1]);  
std::cout << argv[1] << " mentve" << std::endl;  
}
```

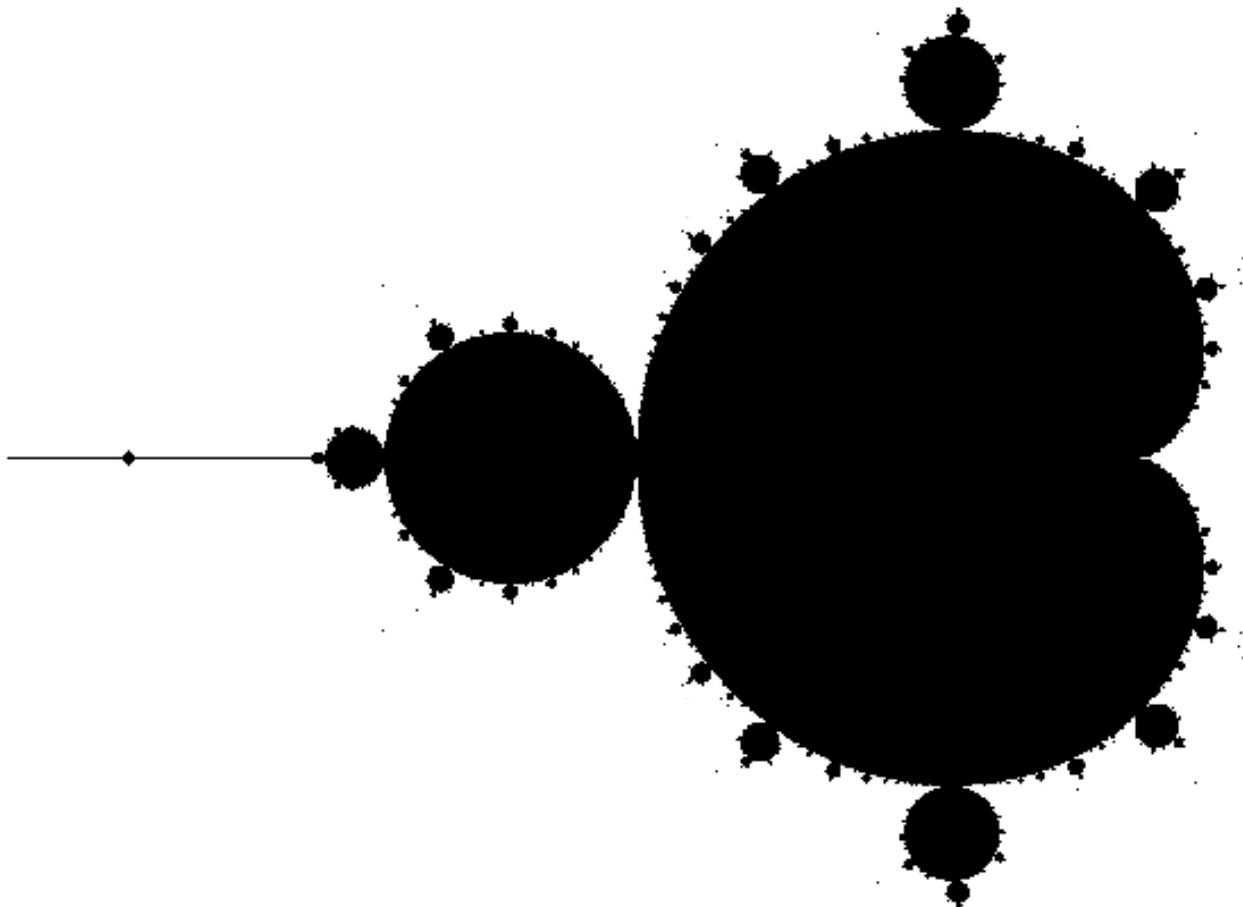
Ahhoz, hogy ezt sikeresen létre tudjuk hozni telepítenünk kell a linuxra a libpng++-dev -et, amit a **sudo apt-get install libpng++-dev** parancssal tudunk telepíteni. Ezután ha minden jól csináltunk akkor le fogja tudni fordítani nekünk a fordító a programot. Ez a libpng++-dev a későbbi feladatoknál is szükséges lesz.

Fordítás: **g++ mandel.cpp -lpng -O3 -o mandel**

Futtatás: **./mandel mt.png**

Kép megnyitása terminálból: **eog mt.png**

Ha minden jól megy akkor a program legenerál nekünk egy mt.png nevű képet, amely a következőképp néz ki:



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

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása:

```
// Verzio: 3.1.2.cpp  
// Forditas:  
// g++ 5.2.cpp -lpng -O3 -o 5.2
```

```
// Futtatas:  
// ./5.2 mandel.png 1920 1080 2040 ←  
// -0.01947381057309366392260585598705802112818 ←  
// -0.0194738105725413418456426484226540196687 ←  
// 0.7985057569338268601555341774655971676111 ←  
// 0.798505756934379196110285192844457924366  
// ./5.2 mandel.png 1920 1080 1020 ←  
// 0.4127655418209589255340574709407519549131 ←  
// 0.4127655418245818053080142817634623497725 ←  
// 0.2135387051768746491386963270997512154281 ←  
// 0.2135387051804975289126531379224616102874  
// Nyomtatás:  
// a2ps 5.2.cpp -o 5.2.cpp.pdf -1 --line-numbers=1 --left-footer="BATF41" ←  
// HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro=color  
// ps2pdf 5.2.cpp.pdf 5.2.cpp.pdf.pdf  
//  
//  
// 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/>.  
  
#include <iostream>  
#include "png++/png.hpp"  
#include <complex>  
  
int  
main ( int argc, char *argv[] )  
{  
  
    int szelesseg = 1920;  
    int magassag = 1080;  
    int iteraciosHatar = 255;  
    double a = -1.9;  
    double b = 0.7;  
    double c = -1.3;  
    double d = 1.3;
```

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

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

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

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

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

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

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

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

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

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

            ++iteracio;
        }
}
```

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

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

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

}
```

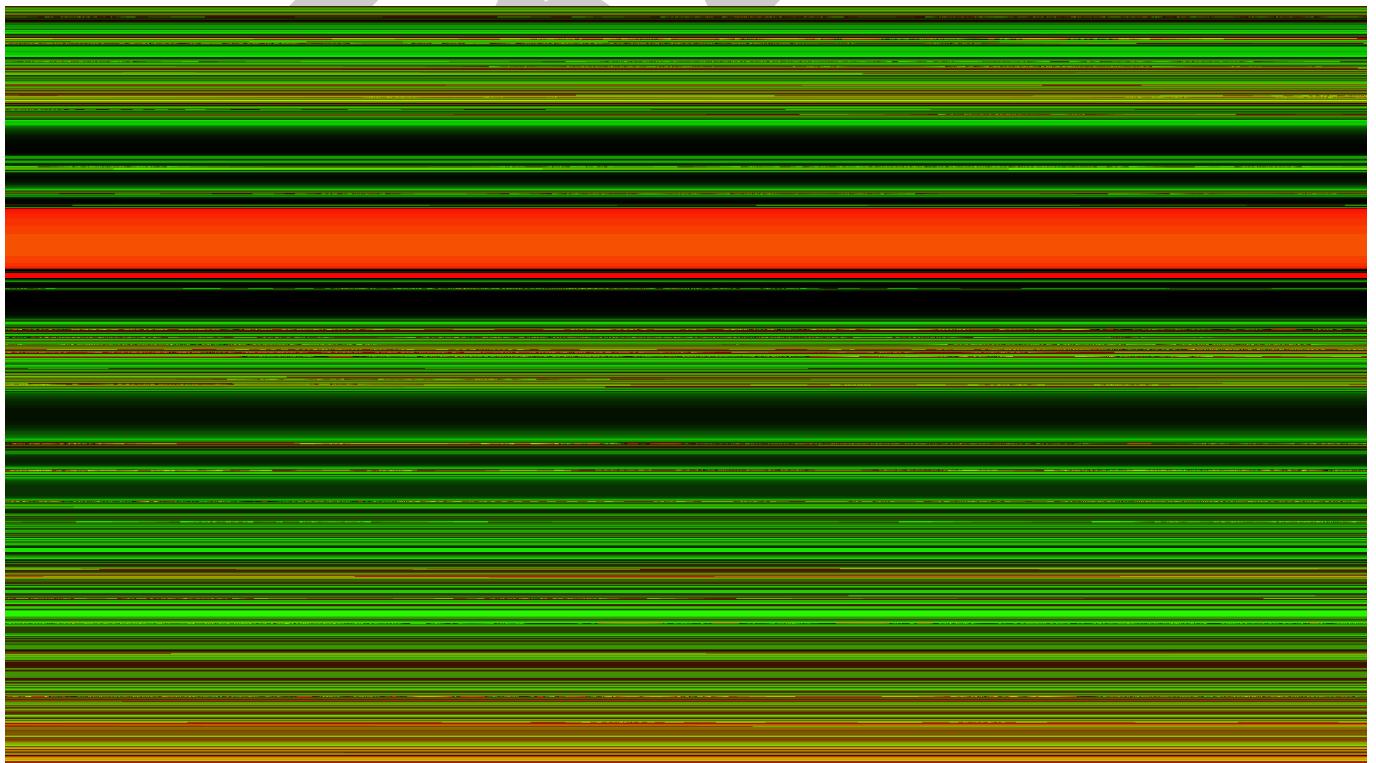
Ha az előző feladatban megoldottuk a fordításhoz szükséges könyvtár telepítését akkor ennek is hibátlanul működnie kell. A program használatakor figyelni kell arra, hogy a ./mandel parancs lefuttatása után a következőképp épül fel a program: [kép neve] [kép szélessége] [kép magassága] [összetettség] [x mettől] [x meddig] [y mettől] [y meddig], arra kell figyelni, hogy az [x mettől] az kisebb legyen, mint az [x meddig] és az y-nál ugyanígy. Tehát az első koordináta mindig nagyobb legyen a másodiknál.

Fordítás: **g++ mandel.cpp -lpng -O3 -o mandel**

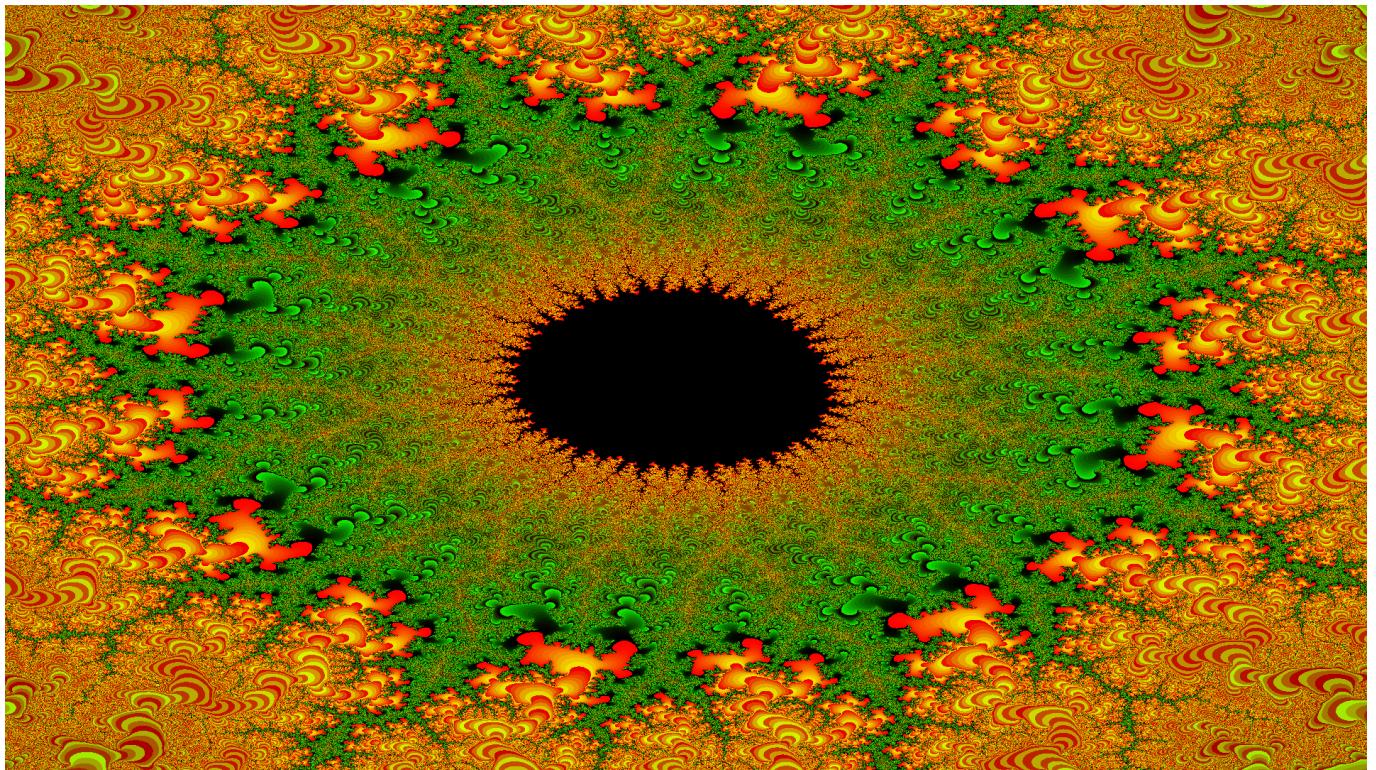
Futtatás: **./mandel mandel.png 1920 1080 1020 0.4127655418209589255340574709407519549131 0.4127655402135387051768746491386963270997512154281 0.2135387051804975289126531379224616102874**

Kép megnyitása terminálból: **eog mandel.png**

Hibásan elkészített fájl:



Helyesen elkészített fájl:



5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbqRzY76E>

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 5.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./5.3 mandel.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatás:
// a2ps 5.3.cpp -o 5.3.cpp.pdf -1 --line-numbers=1 --left-footer="BATF41 ←
// HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro=color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↵
// Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );

    }
    else
    {
        std::cout << "Hasznalat: ./5.3 fajlnev szelesseg magassag n a b c d ↵
                     reC imC R" << std::endl;
        return -1;
    }
}
```

```
}

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

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

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

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

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

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

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

        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {

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

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

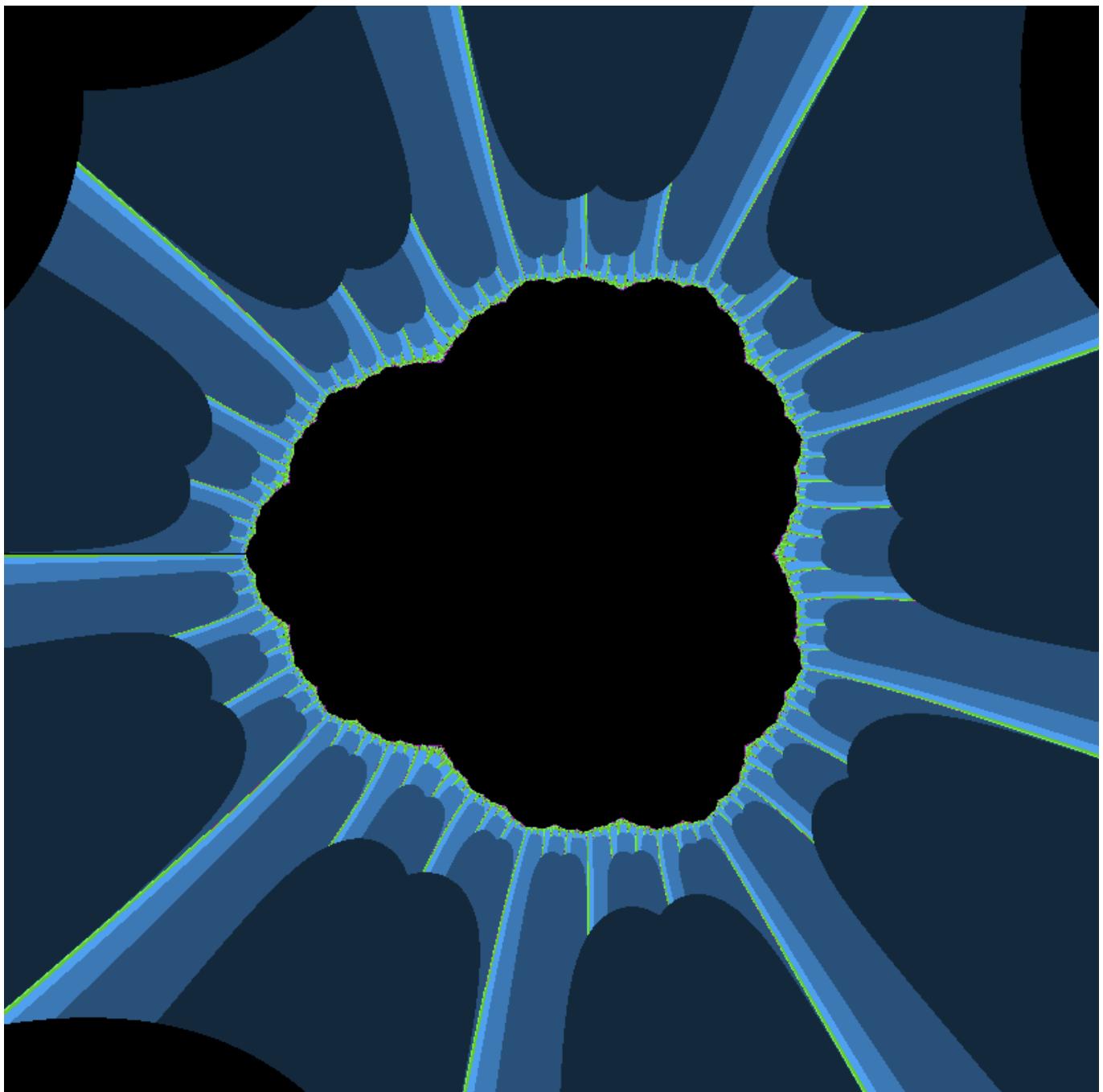
    int szazalek = ( double ) y / ( double ) magassag * 100.0;
    std::cout << "\r" << szazalek << "%" << std::flush;
}

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

Fordítás: **g++ mandel.cpp -lpng -O3 -o mandel**

Futtatás: **./mandel mandel.png 800 800 10 -2 2 -2 2 .285 0 10**

Kép megnyitása terminálból: **eog mandel.png**



5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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. oszlopban 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 < iteraciosHatar)
{
    // 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 (int));  
  
    // dim3 grid (MERET, MERET);  
    // mandelkernel <<< grid, 1 >>> (device_kepadat);  
  
    dim3 grid (MERET / 10, MERET / 10);  
    dim3 tgrid (10, 10);  
    mandelkernel <<< grid, tgrid >>> (device_kepadat);  
  
    cudaMemcpy (kepadat, device_kepadat,  
               MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);  
    cudaFree (device_kepadat);  
  
}  
  
int  
main (int argc, char *argv[])  
{  
  
    // Mérünk időt (PP 64)  
    clock_t delta = clock ();  
    // Mérünk időt (PP 66)  
    struct tms tmsbuf1, tmsbuf2;  
    times (&tmsbuf1);  
  
    if (argc != 2)  
    {  
        std::cout << "Használat: ./mandelpngc fajlnev";  
        return -1;  
    }  
  
    int kepadat [MERET] [MERET];
```

```
cudamandel (kepadat);

png::image < png::rgb_pixel > kep (MERET, MERET);

for (int j = 0; j < MERET; ++j)
{
    //sor = j;
    for (int k = 0; k < MERET; ++k)
    {
        kep.set_pixel (k, j,
            png::rgb_pixel (255 -
                (255 * kepadat[j][k]) / ITER_HAT,
                255 -
                (255 * kepadat[j][k]) / ITER_HAT,
                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;

}
```

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Tutor: Tóth Attila

Megoldás forrása:

```
// Verzio: 3.1.1.cpp
// Forditas:
// g++ 5.5.cpp `libpng-config --ldflags` -O3 -o 3.1.1
// Futtatas:
// ./5.5 mandel.png 1920 1080 2040 ←
// -0.0194738105730936639226058559870580211281 ←
// -0.0194738105725413418456426484226540196687 ←
```

```
0.7985057569338268601555341774655971676111 ←
0.798505756934379196110285192844457924366
// ./5.5 mandel.png 1920 1080 1020 ←
0.41276554182095892553405747094075195491310.41276554182458180530801428176346234
0.2135387051768746491386963270997512154281 ←
0.2135387051804975289126531379224616102874

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

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

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

    if ( argc == 9 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        a = atof ( argv[5] );
        b = atof ( argv[6] );
        c = atof ( argv[7] );
        d = atof ( argv[8] );
    }
    else
    {
        std::cout << "Hasznalat: ./5.5 fajlnev szelesseg magassag n a b c d" << ←
            std::endl;
        std::cout << "Most az alapbeallitasokkal futtatjuk " << szelesseg << " ←
            "
            << magassag << " "
            << iteraciosHatar << " "
            << a << " "
            << b << " "
            << c << " "
            << d << " " << std::endl;
        //return -1;
    }

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

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

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

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

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

        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam
        reC = a + k * dx;
        imC = d - j * dy;
        std::complex<double> c ( reC, imC );

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

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

            ++iteracio;
        }

        iteracio %= 256;

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

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

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

Fordítás: g++ mandel.cpp 'libpng-config --ldflags' -O3 -o mandel

Futtatás: ./mandel mandel.png 1920 1080 1020 0.41276554182095892553405747094075195491310.4127655402135387051768746491386963270997512154281 0.2135387051804975289126531379224616102874

Kép megnyitása terminálból: **eog mandel.png**

A mandelbrot nagyítóval, a nevéről adódóan következtethető, hogy nagyítani tudunk. Ez a nagyítás úgy működik, hogy nem fog nekünk nagyon bepixelesedni a kép, hiszen a halmaz újraszámol nekünk minden nagyításnál. Az "N" betű lenyomásával növelhetjük a kép élességét, így jobban kivehetőek lesznek a részletek. Az "N" betű tulajdonképpen megnöveli az "n"-ek számát a kódunkban, tehát több "n"-ből fog állni a kép, így részletesebb lesz.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJnssY>, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmaz_nagyito_javaval/

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

```
/*
 * MandelbrotHalmazNagyító.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /**
     * A nagyítandó kijelölt területet bal felső sarka.
     */
    private int x, y;
    /**
     * A nagyítandó kijelölt terület szélessége és magassága.
     */
    private int mx, my;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nyígtani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     *
     * @param a           a [a,b]x[c,d] tartomány a koordinátája.
     * @param b           a [a,b]x[c,d] tartomány b koordinátája.
     * @param c           a [a,b]x[c,d] tartomány c koordinátája.
     * @param d           a [a,b]x[c,d] tartomány d koordinátája.
     * @param szélesség   a halmazt tartalmazó tömb szélessége.
     * @param iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
                                    int szélesség, int iterációsHatár) {
        // Az ős osztály konstruktörának hívása
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        // Egér kattintó események feldolgozása:
    }
}
```

```
addMouseListener(new java.awt.event.MouseAdapter() {
    // Egér kattintással jelöljük ki a nagyítandó területet
    // bal felső sarkát:
    public void mousePressed(java.awt.event.MouseEvent m) {
        // A nagyítandó kijelölt területet bal felső sarka:
        x = m.getX();
        y = m.getY();
        mx = 0;
        my = 0;
        repaint();
    }
    // Vonszolva kijelölünk egy területet...
    // Ha felengedjük, akkor a kijelölt terület
    // újraszámítása indul:
    public void mouseReleased(java.awt.event.MouseEvent m) {
        double dx = (MandelbrotHalmazNagyító.this.b
                     - MandelbrotHalmazNagyító.this.a)
                    /MandelbrotHalmazNagyító.this.szélesség;
        double dy = (MandelbrotHalmazNagyító.this.d
                     - MandelbrotHalmazNagyító.this.c)
                    /MandelbrotHalmazNagyító.this.magasság;
        // Az új Mandelbrot nagyító objektum elkészítése:
        new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+←
            x*dx,
            MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
            MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
            MandelbrotHalmazNagyító.this.d-y*dy,
            600,
            MandelbrotHalmazNagyító.this.iterációsHatár);
    }
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // A nagyítandó kijelölt terület szélessége és magassága:
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
*/
 * Pillanatfelvételek készítése.
 */
public void pillanatfelvétel() {
    // Az elmentendő kép elkészítése:
    java.awt.image.BufferedImage mentKép =
        new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
```

```
java.awt.Graphics g = mentKép.getGraphics();
g.drawImage(kép, 0, 0, this);
g.setColor(java.awt.Color.BLUE);
g.drawString("a=" + a, 10, 15);
g.drawString("b=" + b, 10, 30);
g.drawString("c=" + c, 10, 45);
g.drawString("d=" + d, 10, 60);
g.drawString("n=" + iterációsHatár, 10, 75);
if(számításFut) {
    g.setColor(java.awt.Color.RED);
    g.drawLine(0, sor, getWidth(), sor);
}
g.setColor(java.awt.Color.GREEN);
g.drawRect(x, y, mx, my);
g.dispose();
// A pillanatfelvétel képfájl nevének képzése:
StringBuffer sb = new StringBuffer();
sb = sb.delete(0, sb.length());
sb.append("MandelbrotHalmazNagyitas_");
sb.append(++pillanatfelvételszámláló);
sb.append("_");
// A fájl nevébe belelevesszük, hogy melyik tartományban
// találtuk a halmazt:
sb.append(a);
sb.append("_");
sb.append(b);
sb.append("_");
sb.append(c);
sb.append("_");
sb.append(d);
sb.append(".png");
// png formátumú képet mentünk
try {
    javax.imageio.ImageIO.write(mentKép, "png",
        new java.io.File(sb.toString()));
} catch(java.io.IOException e) {
    e.printStackTrace();
}
}
/**
 * A nagyítandó kijelölt területet jelző négyzet kirajzolása.
 */
public void paint(java.awt.Graphics g) {
    // A Mandelbrot halmaz kirajzolása
    g.drawImage(kép, 0, 0, this);
    // Ha éppen fut a számítás, akkor egy vörös
    // vonallal jelöljük, hogy melyik sorban tart:
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
}
```

```
        }
        // A jelző négyzet kirajzolása:
        g.setColor(java.awt.Color.GREEN);
        g.drawRect(x, y, mx, my);
    }
    /**
     * Példányosít egy Mandelbrot halmazt nagyító obektumot.
     */
    public static void main(String[] args) {
        // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
        // tartományában keressük egy 600x600-as hálóval és az
        // aktuális nagyítási pontossággal:
        new MandelbrothalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```

Fordítás: **javac MandelbrotHalmazNagyító.java**

Futtatás: **java MandelbrotHalmazNagyító**



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.

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

A polártranszformáció segítségével random számokat tudunk kiszámolni. Ezt az algoritmust használja a Java random szám generátor függvénye is. C++ kód:

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

class PolarGen {

public:
    PolarGen(); //konstruktor
    ~PolarGen() {} //destruktur

    double kovetkezo(); //random lekérés

private:
    bool nincsTarolt;
    double tarolt; //random értéke

};

PolarGen::PolarGen() { //a konstruktor kifejtése
    nincsTarolt = false;
```

```
    std::srand (std::time(NULL)); //random inicializálás
};

double PolarGen::kovetkezo() { //random lekérő függvény kifejtése
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;

        do{
            u1 = std::rand () / (RAND_MAX + 1.0); //innentől jön az algoritmus
            u2 = std::rand () / (RAND_MAX + 1.0);
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);

        double r = std::sqrt ((-2 * std::log (w)) / w);

        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

        return r * v1; //idáig tart az algoritmus
    }

    else
    {
        nincsTarolt = !nincsTarolt; //ha van korábbi random érték, akkor azt ←
            adja vissza
        return tarolt;
    }
};

int main()
{

    PolarGen rnd;

    for (int i = 0; i < 10; ++i) std::cout << rnd.kovetkezo() << std::endl; ←
        //10 random szám generálása
}
```

Elsősorban létre kell hoznunk egy osztályt, ahol a random számokat fogjuk elkészíteni. Ez kettő részből fog összetevődni:, a `public`, illetve a `private` részből. A `public` részben található elemek elérhetőek a class-on kívül is, viszont a `private` részben találhatóak csak a class-on belül. A konstruktur csak akkor hajtódiik végre, amikor létrehozzuk a `PolarGen` típusú objektumot. Meg kell jegyezni, hogy ez csak egyszer hajtódiik végre, és ez hiában a `public` részen belül található, többet már nem tudjuk meghívni. A destruktőr hasonlóképpen működik, annyi különbséggel, hogy a program futása végén hajtódiik végre. A `kovetkezo()`-vel fogjuk a random számokat kiszámolni, továbbá ez a függvény fogja tartalmazni nekünk azt az algoritmust, amivel ellneőrizzük, hogy van-e tárolt random számunk, és ha nincs akkor generál

kettőt, az egyiket visszaadja, a másikat pedig eltárolja. A konstruktur jelen esetben, ad egy alapértelmezett értéket a nincsTarolt változónak, majd meghívja a srand() függvényt, ami a random számokat fogja generálni. A main-ben létrehozzuk a PolarGen típusú változónkat, ezzel pedig 10 random számot tudunk generálni.

Java kód:

```
public class PolarGenerator
{
    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
        if(nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do{
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2* u1 -1;
                v2 = 2* u2 -1;
                w = v1*v1 + v2*v2;
            } while (w>1);

            double r = Math.sqrt((-2 * Math.log(w) / w));
            tarolt = r * v2;
            nincsTarolt = !nincsTarolt;
            return r * v1;
        }
        else
        {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }

    public static void main(String[] args)
    {
        PolarGenerator g = new PolarGenerator();
        for (int i = 0; i < 10; ++i)
        {
            System.out.println(g.kovetkezo());
        }
    }
}
```

Java-ban az egész forrás egy nagy class része. A C++ kódban tudtuk tömbösíteni a private és public elemeket, itt viszont minden elő ki kell írni, hogy melyik csoporthoz tartozik. Ebben a kódban is megtalálható egy konstruktor, ami a nincsTárolt értékét igazzá definiálja. Végül a main-ben legeneráljuk a random számokat.

6.2. LZW

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

Az LZW algoritmus egy tömörítő eljárás, amelynek lényege annyi, hogy a bemeneti 0-ákból és 1-ekből egy bináris fát épít. A fát az alapján építi fel, hogy megnézi, hogy van-e 0-ás, vagy 1-es gyermek és ha nincs akkor minden létrehoz egyet, majd visszaugrik a gyökérre. Ha van, akkor a 0-ás vagy az 1-es gyermekre lép, és addig lépked lefelé a fában, ameddig nem talál egy olyan részfát, ahol létre kellene hoznia egy új gyermeket. Ahhoz, hogy ezt a kódot le tudjuk fordítani szükségünk lesz az std_lib_facilities.h include-ra.

```
#include "std_lib_facilities.h"
#include <iostream>
#include <string>

string fullCode;
int tbs;
int subfak;

struct binfa
{
    char kulcs;
    binfa* bal;
    binfa* jobb;
    binfa* szulo;
};

binfa* createfa(char value, binfa* fszulo)
{
    binfa* fa = new binfa;
    fa->szulo = fszulo;
    fa->kulcs = value;
    fa->bal = NULL;
    fa->jobb = NULL;
    return fa;
}

int insertfa(binfa*& gyoker, int num)
{
    if(fullCode[num] == '0'){
        if(gyoker->bal == NULL){
            gyoker->bal = createfa('0', gyoker);
            num = num+1;
        }else{
```

```
        num = insertfa(gyoker->bal, num+1);
    }
} else if(fullCode[num] == '1')
{
    if(gyoker->jobb == NULL) {
        gyoker->jobb = createfa('1', gyoker);
        num = num+1;
    } else{
        num = insertfa(gyoker->jobb, num+1);
    }
}
return num;
}

int faLength(binfa* gyoker)
{
    if(gyoker == NULL)
    {
        return 0;
    } else{
        return 1+faLength(gyoker->bal)+faLength(gyoker->jobb);
    }
}

int faHeight(binfa* gyoker)
{
    if(gyoker == NULL)
    {
        return 0;
    } else{
        return 1+max(faHeight(gyoker->bal), faHeight(gyoker->jobb));
    }
}

int faAgak(binfa* gyoker)
{
    if(gyoker == NULL)
    {
        return 0;
    } else if(gyoker->bal == NULL && gyoker->jobb == NULL)
    {
        return 1;
    } else{
        return faAgak(gyoker->bal)+faAgak(gyoker->jobb);
    }
}

int faAgakSum(binfa* gyoker, int count)
{
    if(gyoker == NULL)
```

```
{  
    return count;  
}  
else{  
    if(gyoker->bal != NULL && gyoker->jobb != NULL)  
{  
        return faAgakSum(gyoker->bal, count+1)+faAgakSum(gyoker->jobb, ←  
            count+1);  
    }else if(gyoker->bal != NULL)  
{  
        return faAgakSum(gyoker->bal, count+1);  
    }else if(gyoker->jobb != NULL)  
{  
        return faAgakSum(gyoker->jobb, count+1);  
    }  
}  
return count;  
}  
  
int faAgakListazas(binfa* gyoker, int count)  
{  
    if(gyoker == NULL)  
{  
        return count;  
    }else{  
        if(gyoker->bal != NULL && gyoker->jobb != NULL)  
{  
            return faAgakListazas(gyoker->bal, count+1)+faAgakListazas( ←  
                gyoker->jobb, count+1);  
        }else if(gyoker->bal != NULL)  
{  
            return faAgakListazas(gyoker->bal, count+1);  
        }else if(gyoker->jobb != NULL)  
{  
            return faAgakListazas(gyoker->jobb, count+1);  
        }  
    }  
    string tmp = "";  
    binfa* current = gyoker;  
    while(current != NULL)  
{  
        tmp += (current->kulcs);  
        current = current->szulo;  
    }  
    int i;  
    for(i=0; i<tmp.length(); i++)  
{  
        if(tmp[tmp.length()-i-1] != '#')  
        {  
            cout << tmp[tmp.length()-i-1];  
        }  
    }  
}
```

```
    }
    cout << "(" << i-1 << ")" << "\n";
    return count;
}

double faAgakSzoras(binfa* gyoker, double count, binfa* base)
{
    if(gyoker == NULL)
    {
        return count;
    }else{
        if(gyoker->bal != NULL && gyoker->jobb != NULL)
        {
            return faAgakSzoras(gyoker->bal, count+1, base)+faAgakSzoras( ←
                gyoker->jobb, count+1, base);
        }else if(gyoker->bal != NULL)
        {
            return faAgakSzoras(gyoker->bal, count+1, base);
        }else if(gyoker->jobb != NULL)
        {
            return faAgakSzoras(gyoker->jobb, count+1, base);
        }
    }

    double avg = tbs;
    avg /= subfak;
    return pow(count-avg, 2);
}

int faSum(binfa* gyoker)
{
    if(gyoker == NULL)
    {
        return 0;
    }else{
        char tmp;
        if(gyoker->kulcs == '#') { tmp = '0'; }
        else{ tmp = gyoker->kulcs; }
        int n = tmp - '0';
        return n+faSum(gyoker->bal)+faSum(gyoker->jobb);
    }
}

int main()
{
    string input;
    getline(cin, input);
    string code(input);
    int tordeles = 60;
    if(code.length() <= 1000)
```

```
{  
    cout << "Input adatok(Tördelés " << tordeles << " elemenként):\n";  
    for(int i = 0; i < code.length(); i++)  
    {  
        cout << input[i];  
        if(((i+1)%tordeles == 0 && i != 0) || i == code.length()-1)  
        {  
            cout << "\n";  
        }  
    }  
  
    binfa* gyoker = createfa('#', NULL);  
    int num = 0;  
    fullCode = code;  
    while(code.length() > num)  
    {  
        num = insertfa(gyoker, num);  
    }  
    cout.precision(9);  
    double avg = faSum(gyoker);  
    double faavg = faAgakSum(gyoker, 0);  
    subfak = faAgak(gyoker);  
    avg /= (faLength(gyoker)-1);  
    tbs = faavg;  
    faavg /= subfak;  
    double sz = faAgakSzoras(gyoker, 0, gyoker);  
    double sz2 = faAgakSzoras(gyoker, 0, gyoker);  
    sz /= subfak;  
    sz2 /= subfak-1;  
    sz = pow(sz, 0.5);  
    sz2 = pow(sz2, 0.5);  
    int l, h;  
    l = faLength(gyoker);  
    h = faHeight(gyoker);  
    cout << "\nA fa ágainak száma: " << subfak << "\n";  
    cout << "A fa ágainak átlagos hossza: " << faavg << "\n";  
    cout << "Az ágak hosszainak szórása: " << sz << "\n";  
    cout << "Az ágak hosszainak korrigált szórása: " << sz2 << "\n";  
    cout << "A fa elemszáma: " << l-1 << " (" << l << ") \n";  
    cout << "A fa magassága: " << h-1 << " (" << h << ") \n";  
    cout << "A fa elemeinek átlaga: " << avg << "\n";  
    if(subfak<=50)  
    {  
        cout << "\nA fa ágia:\n";  
        faAgakListazas(gyoker, 0);  
    }  
}
```

A kódunk a következőképp épül fel: Elsőnek is deklarálunk néhány dolgot, mely a későbbiekben lesz fontos

a számunkra, mint például a fullCode, vagy a subfak. Ezután felépítjük a binfa struktúránkat, ahol van egy kulcsunk, egy jobbra, valamint balra mutatónk, és egy szülő mutatónk. Ezután megmondjuk a programnak, hogy miképp hozza létre a fát, majd létrehozunk egy újabb funkciót ahol a fa beszúrását fogja elvégezni a program. Ezután különböző funkciókat hozunk létre: fa ágainak hossza, magassága, és magukat a faágakat. Ahhoz hogy ezeket igazán meg tudjuk érteni le kell görgetnünk a main részhez, ahol láthatjuk, hogy miket fog kiírni nekünk a program. Próbáltam egyszerű, könnyen értelmezhető neveket adni, így könnyedén lehet társítani az adott mondathoz egy funkciót. Ez feladat volt nekünk Bevezetés a programozásban tárgyból, ezért pont ezeket csináltam meg.

6.3. Fabejárás

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

Tutor: Puskás Csaba Zsolt

Ahhoz, hogy nekikezdjünk a feladatnak először is két dolgot kell értelmeznünk. Mi is az az inorder, mi az a posztorder és végül mi az a preorder bejárás? Az inorder bejárásnál elsőnek dolgozzuk fel a bal oldali gyermeket a fában és utána a gyökérelemeimet, majd végezetül a jobb oldali gyermeket. Az inorder bejárást használtuk eddig. A posztorder bejárás abban különbözik, hogy elsőnek a bal oldali gyermeket, másodjára a jobb oldali gyermeket, végül pedig a gyökérelemeimet dolgozzuk fel. Végezetül a preorder bejárás a gyökérelemekkel kezd, majd a bal és a jobb oldali gyermeket dolgozza fel. A kód:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string.h>

#define null NULL

class Binfa
{
private:
    class Node
    {
public:
        Node(char c=' ')
        {
            this->c=c;
            this->left = null;
            this->right = null;
        }
        char c;
        Node* left;
        Node* right;
    };
    Node* fa;

    Node * masol ( Node * elem, Node * regifa ) {
```

```
Node * ujelem = NULL;

    if ( elem != NULL ) {
        ujelem = new Node ( elem->c );
        ujelem->left = masol ( elem->left, regifa ) ;
        ujelem->right = masol ( elem->right, regifa ) ;

        if ( regifa == elem )
            fa = ujelem;

    }

    return ujelem;
}

public:
Binfa(): fa(&gyoker)
{
}

Binfa ( const Binfa & regi )
{
    std::cout << "LZWBinFa copy ctor" << std::endl;

    gyoker.left = masol ( regi.gyoker.left, regi.fa ) ;
    gyoker.right = masol ( regi.gyoker.right, regi.fa ) ;

    if ( regi.fa == & ( regi.gyoker ) )
        fa = &gyoker;

}

Binfa ( Binfa && regi )
{
    std::cout << "LZWBinFa move ctor" << std::endl;

    gyoker.left = regi.gyoker.left;
    gyoker.right = regi.gyoker.right;

    regi.gyoker.left = nullptr ;
    regi.gyoker.right = nullptr ;

}

Binfa& operator= ( const Binfa & regi )
{
    std::cout << "LZWBinFa copy ctor" << std::endl;
```

```
gyoker.left = masol ( regi.gyoker.left, regi.fa ) ;
gyoker.right = masol ( regi.gyoker.right, regi.fa ) ;

if ( regi.fa == & ( regi.gyoker ) )
    fa = &gyoker;

}

Binfa& operator= (Binfa && regi)
{
    std::cout << "LZWBinFa move ctor" << std::endl;

    gyoker.left = regi.gyoker.left;
    gyoker.right = regi.gyoker.right;

    regi.gyoker.left = nullptr ;
    regi.gyoker.right = nullptr ;

}

void operator<<(char c)
{
    if(c=='0')
    {
        if(fa->left == null)
        {
            fa->left = new Node('0');
            fa = &gyoker;
        }
        else
        {
            fa = fa->left;
        }
    }
    else
    {
        if(fa->right == null)
        {
            fa->right = new Node('1');
            fa = &gyoker;
        }
        else
        {
            fa = fa->right;
        }
    }
}

void preorder(Node* elem,int depth=0)
{
```

```
if (elem==null)
{
    return;
}
if (depth)
{
    char *spaces;
    spaces =(char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0;i<depth;i+=2)
    {
        spaces[i]='-';
        spaces[i+1]='-';
    }
    spaces[depth*2]='\0';

    printf("%s%c\n",spaces,elem->c);
}
else
{
    printf("%c\n",elem->c);
}
preorder(elem->left,depth+1);
preorder(elem->right,depth+1);
}

void inorder(Node* elem,int depth=0)
{
    if (elem==null)
    {
        return;
    }
    inorder(elem->left,depth+1);
    if (depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
    inorder(elem->right,depth+1);
}
```

```
}

void postorder(Node* elem, int depth=0)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left, depth+1);
    postorder(elem->right, depth+1);
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0; i<depth; i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
}

void destroy_tree(Node* elem)
{
    if(elem==null)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if(elem->c != '/') delete elem;
}

Node gyoker;

};

void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
}
```

```
    printf("--postorder\tA bináris fa postorder bejárása\n");
}

int main(int argc, char** argv)
{
    srand(time(0));
    Binfa bfa,bfa2;
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            bfa<<'1';
            bfa2<<'0';
        }
        else
        {
            bfa<<'0';
            bfa2<<'1';
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1], "--preorder")==0)
        {
            std::cout<<"bfa"<<std::endl;
            bfa.preorder(&bfa.gyoker);
            std::cout<<"bfa2"<<std::endl;
            bfa2.preorder(&bfa.gyoker);
        }
        else if(strcmp(argv[1], "--inorder")==0)
        {
            std::cout<<"bfa"<<std::endl;
            bfa.inorder(&bfa.gyoker);
            std::cout<<"bfa2"<<std::endl;
            bfa2.inorder(&bfa.gyoker);
        }
        else if(strcmp(argv[1], "--postorder")==0)
        {
            std::cout<<"bfa"<<std::endl;
            bfa.postorder(&bfa.gyoker);
            std::cout<<"bfa2"<<std::endl;
            bfa2.postorder(&bfa.gyoker);
        }
        else
        {
            usage();
        }
    }
    else
```

```
{  
    usage();  
}  
  
bfa2=std::move(bfa);  
  
bfa2.preorder(&bfa2.gyoker);  
  
bfa.destroy_tree(&bfa.gyoker);  
    bfa2.destroy_tree(&bfa.gyoker);  
return 0;  
}
```

6.4. Tag a gyökér

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

```
class elem  
{  
public:  
    int adat;  
    int szint;  
    elem* jobb;  
    elem* bal;  
    elem(int, int);  
};  
  
elem :: elem(int a, int sz)  
{  
    adat= a;  
    bal=NULL;  
    jobb=NULL;  
    szint= sz+1;  
}  
  
class tree{  
public:  
    elem* gyoker;  
    tree ( tree && regi )  
    {  
        cout << "LZWBinFa move ctor" << endl;  
        gyoker = regi.gyoker;  
        regi.gyoker = nullptr;  
    }  
    tree(elem*);
```

```
};  
  
tree :: tree(elem* root)  
{  
    gyoker = root;  
}
```

6.5. Mutató a gyökér

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

A fára való hivatkozás egy gyökérelelemre mutató mutatóval történik. Ilyen esetben az a probléma merül fel, hogy a fa nem egy egész objektumként jön létre, így a másolás és a mozgatás bonyolultabban történik.

```
class elem  
{  
public:  
    int adat;  
    int szint;  
    elem* jobb;  
    elem* bal;  
    elem(int, int);  
  
};  
  
elem :: elem(int a, int sz)  
{  
    adat= a;  
    bal=NULL;  
    jobb=NULL;  
    szint= sz+1;  
}
```

6.6. Mozgató szemantika

Írj az előző programhoz mozgató konstruktort és értékkadást, a mozgató konstruktor legyen a mozgató értékkadásra alapozva!

Tutor: Ádám Petra

Egy új gyökérelemet kell állítanunk a tree-re, a régi gyökérelemet pedig nullpointerre állítjuk, így a fa objektum nem lesz már a program számára hivatkozható.

```
#include <iostream>  
#include <cstdlib>  
#include <ctime>  
#include <string.h>
```

```
#define null NULL

class Binfa
{
private:
    class Node
    {
public:
    Node(char c='/')
    {
        this->c=c;
        this->left = null;
        this->right = null;
    }
    char c;
    Node* left;
    Node* right;
};

Node* fa;

Node * masol ( Node * elem, Node * regifa ) {

    Node * ujelem = NULL;

    if ( elem != NULL ) {
        ujelem = new Node ( elem->c );

        ujelem->left = masol ( elem->left, regifa ) ;
        ujelem->right = masol ( elem->right, regifa ) ;

        if ( regifa == elem )
            fa = ujelem;
    }

    return ujelem;
}

public:
Binfa(): fa(&gyoker)
{
}

Binfa ( const Binfa & regi )
{
    std::cout << "LZWBinFa copy ctor" << std::endl;
    gyoker.left = masol ( regi.gyoker.left, regi.fa ) ;
}
```

```
gyoker.right = masol ( regi.gyoker.right, regi.fa ) ;

if ( regi.fa == & ( regi.gyoker ) )
    fa = &gyoker;

}

Binfa ( Binfa && regi )
{
    std::cout << "LZWBinFa move ctor" << std::endl;

    gyoker.left = regi.gyoker.left;
    gyoker.right = regi.gyoker.right;

    regi.gyoker.left = nullptr ;
    regi.gyoker.right = nullptr ;

}

Binfa& operator= ( const Binfa & regi )
{
    std::cout << "LZWBinFa copy ctor" << std::endl;

    gyoker.left = masol ( regi.gyoker.left, regi.fa ) ;
    gyoker.right = masol ( regi.gyoker.right, regi.fa ) ;

    if ( regi.fa == & ( regi.gyoker ) )
        fa = &gyoker;

}

Binfa& operator= (Binfa && regi )
{
    std::cout << "LZWBinFa move ctor" << std::endl;

    gyoker.left = regi.gyoker.left;
    gyoker.right = regi.gyoker.right;

    regi.gyoker.left = nullptr ;
    regi.gyoker.right = nullptr ;

}

void operator<<(char c)
{
    if(c=='0')
    {
        if(fa->left == null)
        {
            fa->left = new Node('0');
        }
    }
}
```

```
    fa = &gyoker;
}
else
{
    fa = fa->left;
}
}
else
{
    if(fa->right == null)
    {
        fa->right = new Node('1');
        fa = &gyoker;
    }
    else
    {
        fa = fa->right;
    }
}
}

void preorder(Node* elem, int depth=0)
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
    preorder(elem->left, depth+1);
    preorder(elem->right, depth+1);
}

void inorder(Node* elem, int depth=0)
{
```

```
if (elem==null)
{
    return;
}
inorder(elem->left, depth+1);
if (depth)
{
    char *spaces;
    spaces =(char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0; i<depth; i+=2)
    {
        spaces[i]='-';
        spaces[i+1]='-';
    }
    spaces[depth*2]='\0';

    printf("%s%c\n", spaces, elem->c);
}
else
{
    printf("%c\n", elem->c);
}
inorder(elem->right, depth+1);
}

void postorder(Node* elem, int depth=0)
{
    if (elem==null)
    {
        return;
    }
    postorder(elem->left, depth+1);
    postorder(elem->right, depth+1);
    if (depth)
    {
        char *spaces;
        spaces =(char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0; i<depth; i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';

        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
}
```

```
}

void destroy_tree(Node* elem)
{
    if(elem==null)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if(elem->c=='/') delete elem;
}

Node gyoker;

};

void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}

int main(int argc, char** argv)
{
    srand(time(0));
    Binfa bfa,bfa2;
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            bfa<<'1';
            bfa2<<'0';
        }
        else
        {
            bfa<<'0';
            bfa2<<'1';
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1],"--preorder")==0)
        {
            std::cout<<"bfa"<<std::endl;
            bfa.preorder(&bfa.gyoker);
```

```
        std::cout<<"bfa2"<<std::endl;
        bfa2.preorder(&bfa.gyoker);
    }
    else if(strcmp(argv[1], "--inorder")==0)
    {
        std::cout<<"bfa"<<std::endl;
        bfa.inorder(&bfa.gyoker);
        std::cout<<"bfa2"<<std::endl;
        bfa2.inorder(&bfa.gyoker);
    }
    else if(strcmp(argv[1], "--postorder")==0)
    {
        std::cout<<"bfa"<<std::endl;
        bfa.postorder(&bfa.gyoker);
        std::cout<<"bfa2"<<std::endl;
        bfa2.postorder(&bfa.gyoker);
    }
    else
    {
        usage();
    }
}
else
{
    usage();
}

bfa2=std::move(bfa);

bfa2.preorder(&bfa2.gyoker);

bfa.destroy_tree(&bfa.gyoker);
bfa2.destroy_tree(&bfa.gyoker);
return 0;
}
```



7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása: `../Hello_conway/hangyak`

Tutor: Ádám Petra

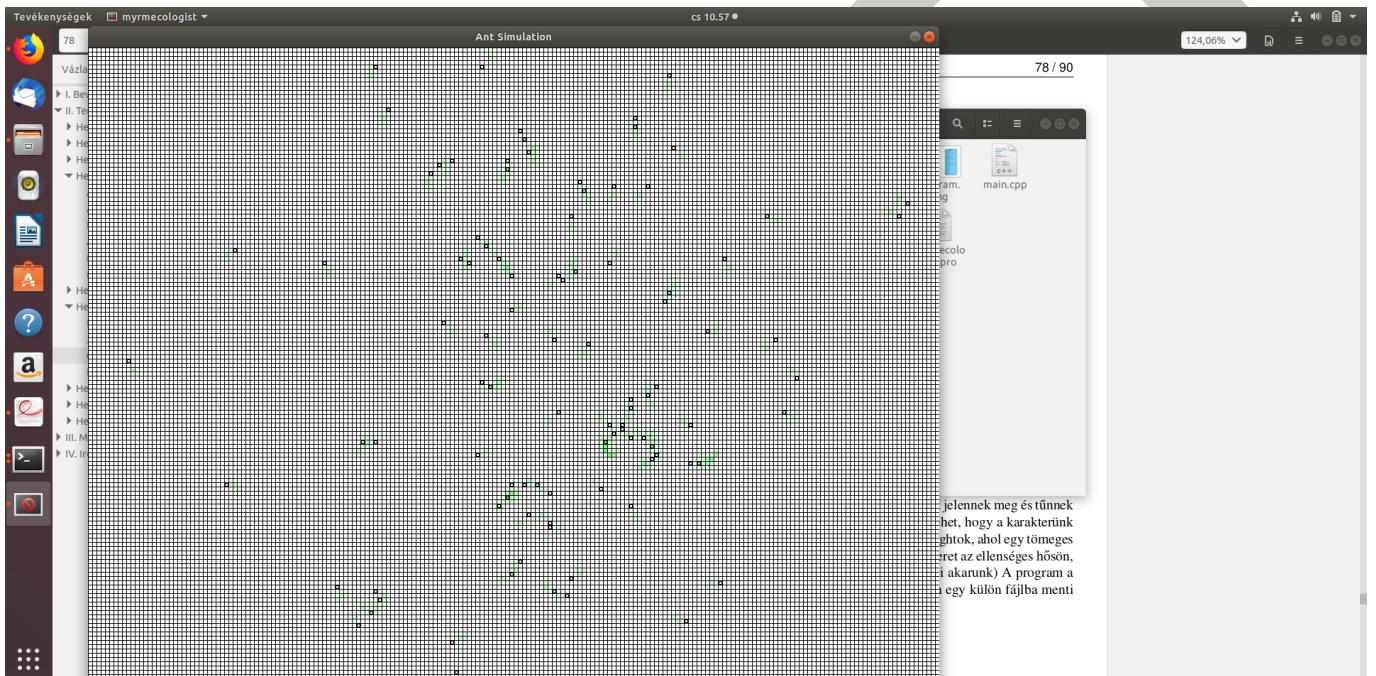
Ha a mandelbrotnál már telepítettük a Qt-t, akkor ismerős lehet számunkra, viszont itt ügyelni kell arra, hogy 6.2-nél újabb verziójú Qt-t használunk, különben nem fog jól működni a kódunk. Biológiai szempontból érdekes ez a kód, mert a hangyák vonulását szimulálja. A kódban meg tudjuk adni, hogy milyen irányban haladjanak a hangyák. Feromont húznak maguk után, így ha egy hangya találkozik egy feromonnal akkor annak az irányába kezd el menni. A működési elve annyi, hogy egy classban van deklarálva egy hangya 3 tulajdonsága. Ezt a class-t sokszorosítva több hangyat kaphatunk. Egy másik osztályban kirajzoljuk a hangyákat, egy harmadikban pedig mozgatjuk azokat.

```
QCommandLineOption szeles_opt ( {"w", "szelesseg"}, "Oszlopok (cellakban ↔
) szama.", "szelesseg", "200" );
QCommandLineOption magas_opt ( {"m", "magassag"}, "Sorok (cellakban) ↔
szama.", "magassag", "150" );
QCommandLineOption hangyszam_opt ( {"n", "hangyszam"}, "Hangyak szama ↔
.", "hangyszam", "100" );
QCommandLineOption sebesseg_opt ( {"t", "sebesseg"}, "2 lepes kozotti ↔
ido (millisec-ben).", "sebesseg", "100" );
QCommandLineOption parolgas_opt ( {"p", "parolgas"}, "A parolgas erteke ↔
.", "parolgas", "8" );
QCommandLineOption feromon_opt ( {"f", "feromon"}, "A hagyott nyom ↔
erteke.", "feromon", "11" );
QCommandLineOption szomszed_opt ( {"s", "szomszed"}, "A hagyott nyom ↔
erteke a szomszedokban.", "szomszed", "3" );
QCommandLineOption alapertek_opt ( {"d", "alapertek"}, "Indulo ertekek a ↔
cellakban.", "alapertek", "1" );
QCommandLineOption maxcella_opt ( {"a", "maxcella"}, "Cella max erteke ↔
.", "maxcella", "50" );
```

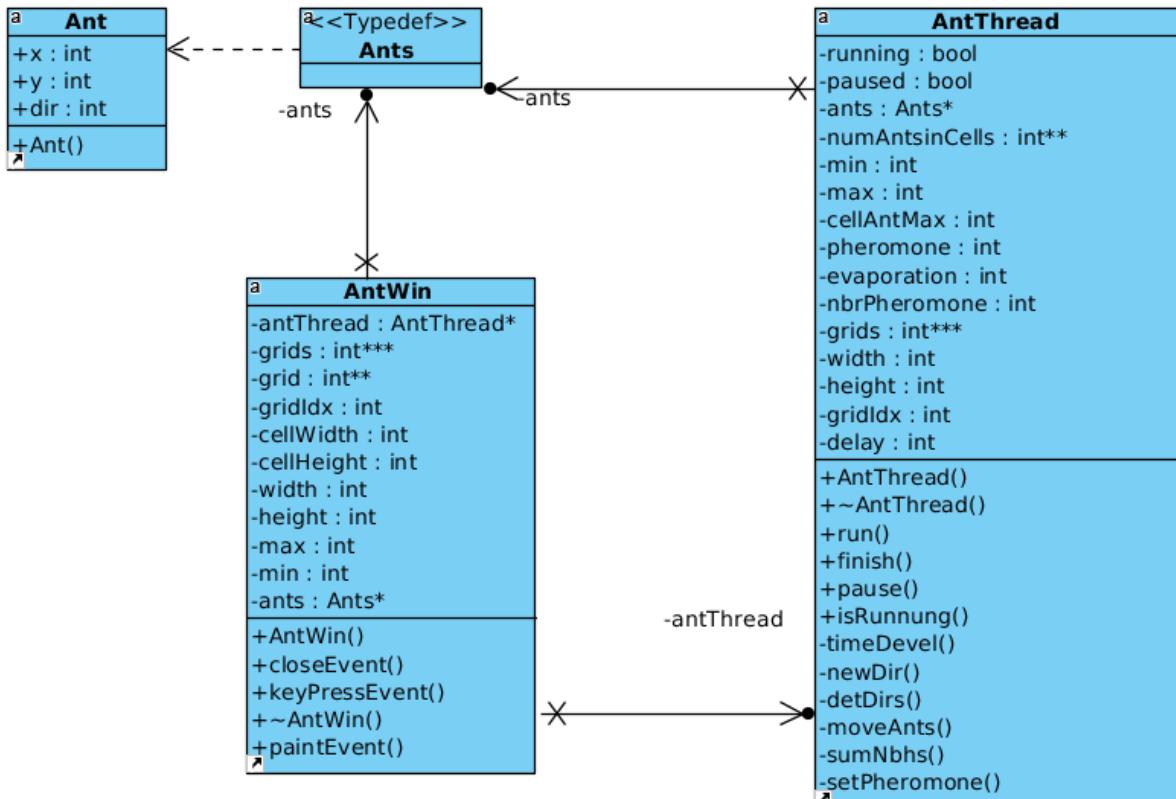
```
QCommandLineOption mincella_opt ( {"i","mincella"}, "Cella min erteke ←
    .", "mincella", "2" );
QCommandLineOption cellamerete_opt ( {"c","cellameret"}, "Hany hangya ←
    fer egy cellaba.", "cellameret", "4" );
```

A fent látható bemeneti értékekkel be lehet állítani, hogy milyen széles legyen, milyen magas, hánnyal, milyen sebességgel, stb.. a szimulációnk. A sor végén található számértékek az alapértelmezett értékek.

Ahhoz, hogy megkapjuk a futtatható fájlunkat több fájlra is lesz szükségünk. A .h fájlokra, a .cpp fájlokra és a .pro fájlokra. Ha ezek megvannak akkor a Qt-val egy qmake paranccsal létrehozhatjuk a Makefile-unkat, amit végül a make paranccsal egy programmá alakíthatunk, amit pedig egyszerűen csak a ./ parancssal futtathatunk is. Persze a fent említett bemeneti értékeket állítgathatjuk ízlésünk szerint.



A diagrammot, a Visual Paradigm nevű programmal készítettem el.



7.2. Java életjáték

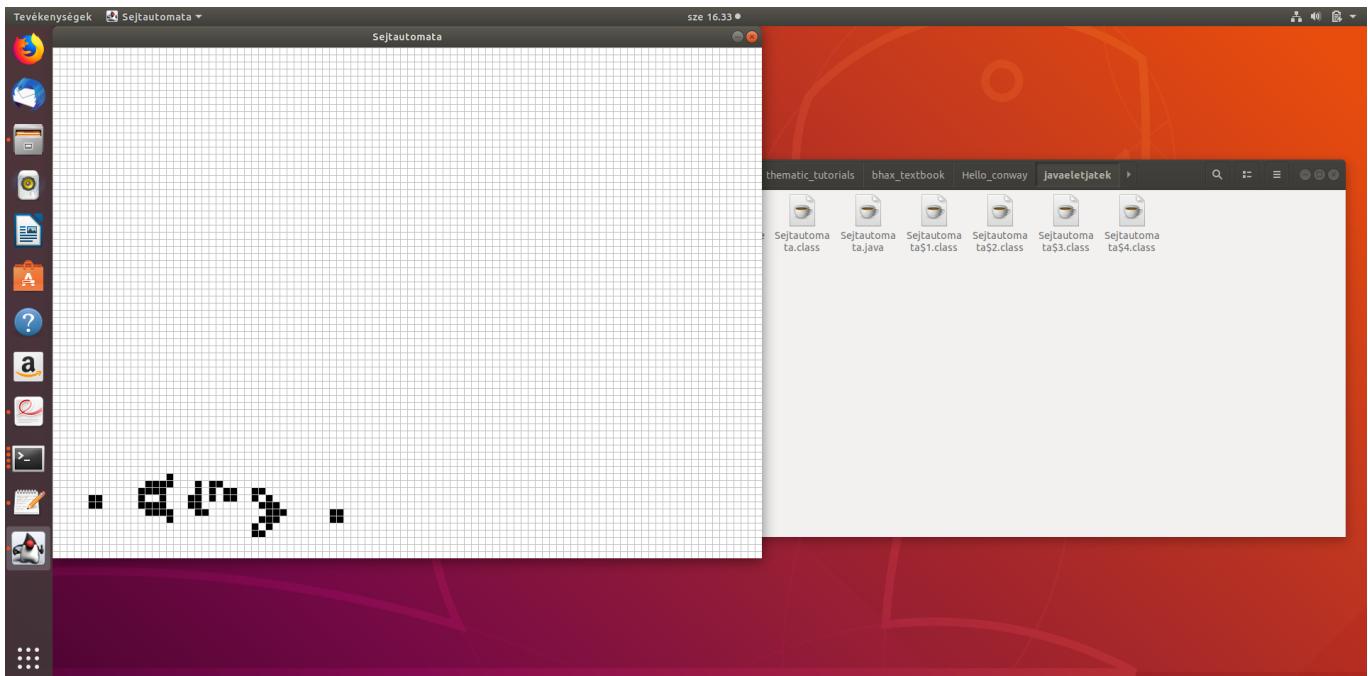
Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás forrása: `../Hello_conway/javaeletjatek`

Kettő egyszerű parancccsal már el is érhetjük, hogy a forrásunk lefusson és elemezni tudjuk a játékot.

```
javac Sejtautomata.java
java Sejtautomata
```

Ez a program tulajdonképpen egy sejtautomata, amelyet Melvin Conway alkotott meg. (Ezért is lett ez a fejezet címe.) A sejtautomatában megadhatunk feltételeket a sejtek életben maradásához, születéséhez és halálozásához. A feltételek a szimulációkban teljesülni fognak, így megfigyelhetjük a sejtjeink miként születnek, miként élnek és miként halnak meg. A Conway féle sejtautomatának 3 szabálya van: 1. Egy sejt csak akkor maradhat életben, ha 2 vagy 3 szomszédja van. 2. Ha egy sejtnek 3-nál több szomszédja van, akkor túlnépesedés miatt meghal, viszont ha 2-nél kevesebb szomszédja van akkor pedig az elszigetelődés miatt fog meghalni. 3. Egy sejt akkor születik meg, ha egy üres cella mellett 3 szomszéd van.



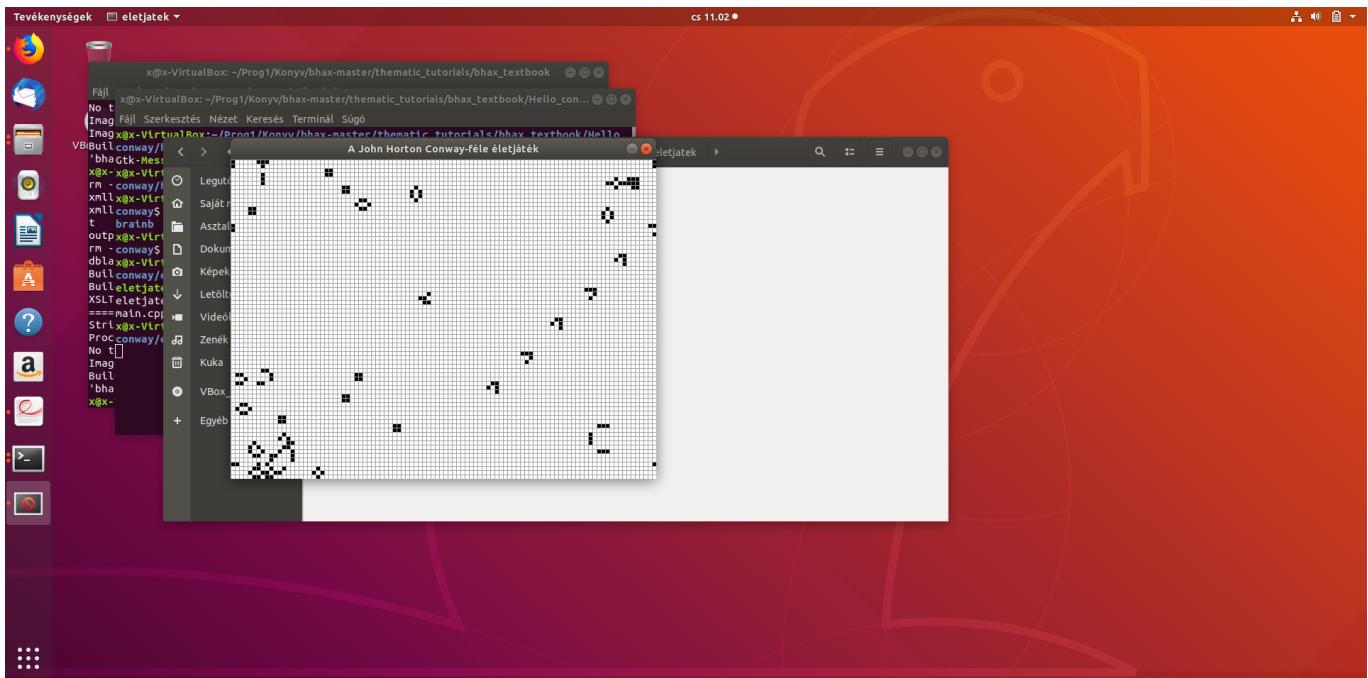
7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: [../Hello_conway/eletjatek](#)

Ez a program az előzőnek a C++ változata. Gyakorlatilag populációk kialakulását és vándorlását szimulálhatjuk a program segítségével. Ha többen kezdenek lenni egy "kolóniában" akkor elkezdenek vándorolni, majd új kolóniákat alapítanak. Ez ugye az előzőben olvasható szabályok által kerül lebonyolításra.

Ahhoz, hogy megkapjuk a futtatható fájlunkat több fájlra is lesz szükségünk. A .h fájlokra, a .cpp fájlokra és a .pro fájlokra. Ha ezek megvannak akkor a Qt-val egy qmake parancssal létrehozhatjuk a Makefile-unkat, amit végül a make parancssal egy programmá alakíthatunk, amit pedig egyszerűen csak a ./ parancssal futtathatunk is.



7.4. BrainB Benchmark

Megoldás forrása: `../Hello_conway/BrainB`

Ez a program leginkább az esportolók mérésére szolgál. Az esport széles körében is szűkítenünk kell inkább a MOBA játékosok felé az invervallumunkat, ugyanis az FPS, vagy más típusú játékosokra nem igazán jöhetsz ki kimagasló eredmények. Ezt mindenki meg is érti, hogy miért. A program működése annyi, hogy 10 percen keresztül rajta kell tartanunk az egerünket a "karakterünkön". Ezt a karaktert egy külön osztályban határozzuk meg. A szimuláció közben különböző zavaró tényezők jelennek meg és tűnnek el. Lehet, hogy a karakterünk előtt jelenik meg az a bizonyos zavaró tényező, de lehet, hogy a karakterünk mögött. Jó példa lehet erre a League of Legends, vagy a Dota 2-ben kialakult Teamfightok, ahol egy tömeges harc alakul ki és továbbra is tudunk kell a karakterünk hollétéről, vagy éppen az egeret az ellenséges hősön, vagy a szövetséges hősön kell tartani (attól függően, hogy támadunk, vagy healelni akarunk) A program a futás közben adatokat gyűjt, melyet a program bezárásakor, vagy a program végén egy külön fájlba menti ki.



DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó: <https://youtu.be/j7f9SkJR3oc>

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa...
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Az MNIST egy adatbázis, melyben kézzel írott számjegyeket tárolunk. Leginkább ezt az adatbázist a képfelismerő programok tanítására használják. A programunk is ezen adatbázis alapján fogja felismerni a kézzel írott számjegyeinket.

Elsősorban telepítenünk kell a TensorFlow-ot és a python-t, hogy el tudjuk kezdeni a feladatmegoldást.

```
import argparse

from tensorflow.examples.tutorials.mnist import input_data

import tensorflow as tf

import matplotlib.pyplot

FLAGS = None

def readimg():
    file = tf.read_file("sajat8a.png")
    img = tf.image.decode_png(file)
    return img

def main(_):
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

    x = tf.placeholder(tf.float32, [None, 784])
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y = tf.matmul(x, W) + b
```

```
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, ←
    y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(←
    cross_entropy)

sess = tf.InteractiveSession()

tf.initialize_all_variables().run()

print("-- A halozat tanitása")
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
        print(i/10, "%")
print("-----")

# Test trained model
print("-- A halozat tesztelese")
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("-- Pontosság: ", sess.run(accuracy, feed_dict={x: mnist.test.←
    images,
                           y_: mnist.test.labels}))
print("-----")

print("-- A MNIST 42. tesztképenek felismerése, mutatom a számot, a ←
    továbblepeshez csukd be az ablakat")

img = mnist.test.images[42]
image = img

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm.←
    .binary)
matplotlib.pyplot.savefig("4.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

print("-- A saját kezi 8-asom felismerése, mutatom a számot, a ←
    továbblepeshez csukd be az ablakat")

img = readimg()
image = img.eval()
```

```
image = image.reshape(28*28)

matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm ←
    .binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ ←
        mnist/input_data',
                        help='Directory for storing input data')
    FLAGS = parser.parse_args()
    tf.app.run()
```

A programunk tehát egy 28x28-as képből képes felismerni a számjegyeket a Tensorflow segítségével.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

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

Megoldás forrása: <https://github.com/Microsoft/malmo>

A Malmö az informatikában a mesterséges intelligenciával foglalkozik. A Malmö elérhető a <https://github.com/M> oldalon, ahol akár az első feladatban használt pip módszerrel is dolgozhatunk. A Minecraftban az x,y,z koordináták alapján nézhetjük meg Steve helyzetét és az őt körülvevő környezetet. Más más biomokban is létrehozhatjuk Steve-t. A program figyeli Steve irányát is, hogy merrefelé néz. A Minecraftban az irányok a következőképp néznek ki: 180 fok észak, 90 fok nyugat, 0 fok dél, -90 fok kelet. A program nézi a Stevet körülvevő blockokat, az alapján próbálja őt irányítani. Nándi okosan kitalálta, hogy több kockát nézve jobb eredményre juthatunk. A program lényege tulajdonképpen annyi, hogy a Minecraft-ban Steve szemével nézzük a világot. Itt kilistázzuk, hogy miket láthat Steve, és ha a cursor rajta van egy blockon akkor azt kiírjuk. Ha a block egy "AIR" azaz levegő block, vagyis egy üres kocka akkor Steve haladhat előre, viszont ha más akkor elkezd jobbra fordulni mindaddig míg egy Air-t nem talál. Ha érzi, hogy előtte egy

block van egyel lejjebb akkor arra fel tud ugrani, valamint így a vízből is kimászik egy bizonyos határig. A programot annyival kiegészíteném, hogy a videóban is látszik, hogy a "TALLGRASS" block esetén is elfordul, viszont azt is nyugodtan lehetne úgy érzékelni, mint az "AIR"-t, vagyis szerintem azon is átmehetne nyugodtan Steve, hiszen ez nem akadály számára, mert nem rendelkezik collisionnel, a "TALLGRASS" block.



9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

A LISP nyelvet leginkább a GIMP hekkeléséhez fogjuk használni. A GIMP egy képszerkesztő program, gyakorlatilag olyan, mint a Photoshopp, csak egyszerűbb, felhasználóbarátabb és kicsikét talán gyengébb is, de nem ez a lényeg, hane az, hogy a GIMP-en belül úgy nevezett Script-fu-kat tudunk majd használni, amelyekkel a nevéből adódóan is Scripteket adhatunk hozzá a GIMP-hez. Mivel én viszonylag gyakran használtam és használom a GIMP-et így tudom, hogy a Szűrők menüpont alatt található meg a Script-fu, egyből a Python-fu alatt. Itt egy Konzolt kapunk amibe írhatjuk az utasításainkat. A faktoriális, nem egy ismeretlen kifejezés, a matematikában találkozhattunk vele. Egy szám faktoriálisa például: $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, tehát elmondhatjuk, hogy $n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdots \cdot 1$. Lispben a faktoriális kiszámítása a következőképpen néz ki iteratív módon:

```
(define (fakt n)
  (if (= 0 n)
      (set! n 1)
    )
  (let loop ((variable (- n 1)))
    (if (> variable 1)
        (begin
          (set! n (* n variable))
          (loop (- variable 1))
        )
      )
    )
  n
)
```

Defináljuk a fakt nevű függvényt, aminek "n" az első paraméterje. Az első if-ben megmondjuk, hogy ha az "n" értéke 0, akkor a faktoriális értéke legyen 1, ugyanis $0! = 1$. A következő sorban megadunk egy változót, aminek az értéke majd az "n" értéke és az 1 különbsége lesz. Ha az így kapott érték nagyobb, mint 1 akkor az "n" értékét megszorozzuk a változónk értékével, végül majd a ciklusváltozó értékét csökkentjük 1-el, így fogjuk megkapni a legvégén álló "n"-t, vagyis a végeredményt. Nézzük a rekurzív megoldást:

```
(define (fakt n)
  (
  if (< n 1)
    1
  (* n (fakt (- n 1))) ;else ag
  )
)
```

Itt is először defináljuk az egy paraméteres fakt nevű függvényünket. A rekurzív megoldásban ciklus nélkül oldjuk meg a faktoriális "problémáját". A probléma lényegében itt annyi lenne, hogy egyszer fut csak le a függvényünk, de láthatjuk, hogy ezt megoldottuk azon módon, hogy visszahivatkoztunk az eredeti függvényre. Tehát ha a bejövő "n" paraméter értéke kisebb, mint 1 akkor 1-et ad végeredményül, de ha nem akkor pedig önmagát fogja meghívni.

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

A Króm Effekt egy GIMP-en belüli Script-fu-val elérhető. Egy általunk előre megadott szövegre kell rárakni a Króm Effektet úgy, hogy Script-Fu-t használunk.

```
(; bhax_chrome3.scm
;
; BHAX-Chrome creates a chrome effect on a given text.
; Copyright (C) 2019
; Norbert Bátfai, batfai.norbert@inf.unideb.hu
; Nándor Bátfai, batfai.nandi@gmail.com
;
; This program is free software: you can redistribute it and/or modify
; it under the terms of the GNU General Public License as published by
; the Free Software Foundation, either version 3 of the License, or
; (at your option) any later version.
;
; This program is distributed in the hope that it will be useful,
; but WITHOUT ANY WARRANTY; without even the implied warranty of
; MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
; GNU General Public License for more details.
;
; You should have received a copy of the GNU General Public License
; along with this program. If not, see <https://www.gnu.org/licenses/>.
;
; Version history
;
; This Scheme code is partially based on the Gimp tutorial
```

```
; http://penguinetes.com/b2evo/index.php?p=351
; (the interactive steps of this tutorial are written in Scheme)
;
; https://bhaxor.blog.hu/2019/01/10/ ←
; a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv
;

(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)

; (color-curve)

(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) ) )



(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))

  (list text-width text-height)
)
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-chrome text font fontsize width height color ←
  gradient)
```

```
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
      LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-width (car (text-wh text font fontsize)))
    (text-height (elem 2 (text-wh text font fontsize)))
    (layer2)
  )

;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND )
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
  ))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
  height 2) (/ text-height 2)))

(set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←
  LAYER)))
;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)

;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
  LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)

;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT- ←
  LINEAR 100 0 REPEAT-NONE
  FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
  3)))
```

```
;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
 0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 '(255 0 0) "←
Crown molding")

(script-fu-register "script-fu-bhax-chrome"
  "Chrome3"
  "Creates a chrome effect on a given text."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 19, 2019"
  ""
  SF-STRING      "Text"        "Bátf41 Haxor"
  SF-FONT        "Font"        "Sans"
  SF-ADJUSTMENT  "Font size"   '(100 1 1000 1 10 0 1)
  SF-VALUE       "Width"       "1000"
  SF-VALUE       "Height"      "1000"
  SF-COLOR       "Color"       '(255 0 0)
  SF-GRADIENT    "Gradient"    "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```

A Króm effektet hekkelgethetjük különféle módokban. Néhány fő argumensre van szükségünk ahhoz, hogy el tudjuk készíteni a krómozott effektünket. Az alapértelmezett argumens értékek az alsó pár sorban láthatóak. Először is a STRING, az maga lesz a szöveg, amit ki írjon nekünk a program, a FONT az egyértelműen a betűtípus, az ADJUSTMENT, az a betű nagysága, az ezután következő értékek a szélesség és a magasság, a COLOR az az effekt színe, végül pedig a GRADIENT, az a csillológás típusa.



skyLine





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

A Mandala egy olyan alakzat, amit szavak forgatásával állíthatunk elő. Engem személy szerint a Mandala szó a Mantra szóra emlékeztet, ami pedig vallásban a buddhizmushoz köthető meditációval kapcsolatos terminus. Véleményem szerint a Mandala végső alakja is igazán hasonlít a Buddha vallásban megtalálható motívumokhoz.

```
; bhax_mandala9.scm
;
; BHAX-Mandala creates a mandala from a text box.
; 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
;
; This Scheme code is partially based on the Python code
; Pat625_Mandala_With_Your_Name.py by Tin Tran, which is released under ←
; the GNU GPL v3, see
; https://gimplearn.net/viewtopic.php/Pat625-Mandala-With-Your-Name-Script ←
;-for-GIMP?t=269&p=976
;
; https://bhaxor.blog.hu/2019/01/10/ ←
; a_gimp_lisp_hackeles_a_scheme_programozasi_nyelv
;

(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  text-width
  )
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  ;;;
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  ;;; ved ki a lista 2. elemét
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
  ;;;
  (list text-width text-height)
  )
)
```

```
; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
    gradient)
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
        LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-layer)
    (text-width (text-width text font fontsize))
    ;;
    (text2-width (car (text-wh text2 font fontsize)))
    (text2-height (elem 2 (text-wh text2 font fontsize)))
    ;;
    (textfs-width)
    (textfs-height)
    (gradient-layer)
  )
  (gimp-image-insert-layer image layer 0 0)

  (gimp-context-set-foreground '(0 255 0))
  (gimp-drawable-fill layer FILL-FOREGROUND)
  (gimp-image-undo-disable image)

  (gimp-context-set-foreground color)

  (set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
  (gimp-image-insert-layer image textfs 0 -1)
  (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
    height 2))
  (gimp-layer-resize-to-image-size textfs)

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
  (set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
  (set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))
)
```

```
(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
-LAYER)))

(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car(gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car(gimp-drawable-height textfs)) 100))

(gimp-layer-resize-to-image-size textfs)

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ←
"gradient" 100 LAYER-MODE-NORMAL-LEGACY)))

(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ (/ ←
width 2) (/ textfs-width 2)) 8) (/ height 2))
```

```
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

  (set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ↵
    )))
  (gimp-image-insert-layer image textfs 0 -1)
  (gimp-message (number->string text2-height))
  (gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ↵
    height 2) (/ text2-height 2)))

  ;(gimp-selection-none image)
  ;(gimp-image-flatten image)

  (gimp-display-new image)
  (gimp-image-clean-all image)
  )
)

; (script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Ruge Boogie" 120 1920 ↵
  1080 '(255 0 0) "Shadows 3")

(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"        "Bátf41 Haxor"
  SF-STRING      "Text2"       "BHAX"
  SF-FONT        "Font"        "Sans"
  SF-ADJUSTMENT   "Font size"   '(100 1 1000 1 10 0 1)
  SF-VALUE        "Width"       "1000"
  SF-VALUE        "Height"      "1000"
  SF-COLOR        "Color"       '(255 0 0)
  SF-GRADIENT     "Gradient"    "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

Ebben a feladatban is a mandala elkészítéséhez különféle argumenseket kell használnunk, de ez szinte semmiben sem változik az előző feladatban megismert argumensektől. Tehát itt is lesz egy STRINGünk, ami az első szöveg lesz, annyi a változás, hogy itt lesz egy második STRING-ünk is. A FONT az egyértelműen a betűtípus, az ADJUSTMENT, az a betű nagysága, az ezután következő értékek a szélesség és a magasság, a COLOR az az effekt színe, végül pedig a GRADIENT, az a csillológás típusa.

Utólag gondoltam Google-ban rákeresek a Mandala szóra és a Wikipédián azt találtam, hogy valóban a buddhista valláshoz köthető, hiszen a mandala a kozmosz, valamint a különféle istenek ábrázolása, így nem volt tulsiósan messze az elköpzelésem ettől az egésztől.





DRAFT

10. fejezet

Helló, Gutenberg!

10.1. Olvasónapló BRIAN W. KERNIGHAN és DENNIS M. RITCHIE A C programozási nyelv című könyvéből

1. fejezet: Alapismeretek

Az első fejezetben megismérhetünk az alapokkal. Például, hogy hogyan írhatunk ki egy „Halló mindenki” parancsot a C nyelvben. Itt találkozhatunk a különböző változó típusokkal, mint például: int, float, char, short, long, double. Megismérhetünk a különböző ciklusokkal, a for-ral és a while-al és még ezen kívül nagyon sok dologgal, ami elengedhetetlen az alapok elindításához. Megismérjük a programok felépítését, az elejtől a végéig, hogy mikből állhat, és mik azok a fontos dolgok, amelyek szükségesek egy program felépítéséhez, majd a lefordításához. Itt már betekintést kaphatunk az if-ek világába, de ez csak később lesz bővebben kifejtve.

2. fejezet: Típusok, operátorok és kifejezések

Ebben a fejezetben megismérjük részletesebben az első fejezetben található változókat, megismérhetünk a deklarációk világával, továbbá megismérjük az állandókat, és azt, hogy azok miként szerepelhetnek a kódunkban és melyik állandó milyen végeredményt ad majd végeredményül. A logikai operátorokról '<' '>' '<=' '>=' '==' is kapunk lényegesen kifejtett ismertetőt.

3. fejezet: Vezérlési szerkezetek

A harmadik fejezet elején az utasításokra kapunk egy példát, hogy egy-egy sor mitől válhat utasítássá. A későbbiekben az if-ek bővített változatát ismerhetjük meg az else és az else iftársaságában. Néhány újabb utasítással is megismérhetünk, ilyen például a switch, amely case-kre bontva nézi meg a bejövő értékeket. Ezután a ciklusokra kerül a sor és azzal kapcsolatosan bővíthetjük tudásunkat. A ciklusoknál bejönnek újabb utasítások: break, continue, goto. Megismérhetjük ezeknek a működését és hasznosságát (vagy épp haszontalanúságát).

4. fejezet: Függvények és a program szerkezete

Ebben a részben, ahogy a fejezet címéből is adódik, a függvényekkel bővíthetjük tudásunkat. A függvények felépítéséről és működéséről olvashatunk az első pár részben. Majd a későbbiekben arról, hogy a program hogyan is állhat össze, de itt már sokkal részletesebben, mint az első fejezetben. Nagyon sok új fogalmat ismerhetünk meg a fejezet olvasása során. Szóba kerülnek például a változók inicializálása, a rekurzió és más hasonló fogalmak.

5. fejezet: Mutatók és tömbök

Az ötödik fejezetben, szintén a címről lehet következtetni a tartalomra. Az ötödik fejezet a mutatókról és a tömbökről szól. Megtudhatjuk, hogy hogyan működnek a mutatók, a kódban hogyan láthatjuk ezeket viszont és, hogy hogyan lehet szakszerűen használni ezeket, valamint a gyakorlatban, milyen előnyei vannak a mutatók használatának. Tovább haladva megismерkedhetünk a tömbökkel, amelyeknél szintén megtudjuk, hogy hogyan létrehozható a tömb, és hogy igazából mit is takar az a tömb. A későbbiekben a mutatókat és a tömböket hozzuk egymással kapcsolatba, valamint a mutatókat az elején tanultakkal (például a függvényekkel) is kapcsolatba hozzuk.

6. fejezet: Struktúrák

A struktúrák a grafikus ábrázoláshoz használhatóak. Itt ismét egy újabb definícióval ismerkedhetünk meg, a struct definíciójával. A továbbiakban az előző fejezetekben elsajtításához kötjük hozzá a struktúrákat, így jöhetnek létre például struktúratömbök, vagy struktúrákat kijelölő mutatók.

7. fejezet: Adatbevitel és adatkivitel

A hetedik fejezetben részletes leírást kapunk az inputról és az outputról. Bő leírást kapunk arról, hogy mi a standard input, illetve mi a standart output. Megismerhetünk olyan függvényeket, melyek adatbevitelre kényszerítik a felhasználót a standard inputról. Ilyen függvény például a scanf függvény. Outputnál megtanulhatjuk a hibakezelést a stderr és az exit segítségével, valamint még számos könyvtári függvényről kaphatunk részletes ismertetőt, leírást.

8. fejezet: Kapcsolódás az UNIX operációs rendszerhez

Ebben a fejezetben megismerhetjük, hogy milyen kódokat használhatunk az UNIX operációs rendszer kezeléséhez egy adott programon belül. Számos rendszerhívást megismерhetünk. Majd nagyon sok olyan információval és példával szolgál a könyv, amelyet a saját könyvünk megírásához is alapul tudunk venni, amelyet készítünk.

Kiértékelés

A könyv kiolvasása bizonyítottan növelte a lexikális tudásomat, hiszen nagyon sok új információt ismertem meg ezáltal, nagyon sok dologra rámutatott amiket nem tudtam. Egyszeri elolvasásra voltak benne olyan részek, melyek még nem igazán rögzültek benuem, de hiszem, hogy ezt a könyvet a későbbiekben is elő fogom venni, mert rendkívül jól vannak benne megfogalmazva az egyes részek és a példákkal, valamint a fejezetek

végén lévő gyakorló feladatokkal folyamatosan tudjuk ellenőrizni az el ← → ōre haladásunkat egy-egy fejezet kiolvasása után.

10.2. Olvasónapló Juhász István Magas Szintű Programozási Nyelvek 1 című könyvéből

1.2. Alapfogalmak

A programozási nyelveknek három szintje van. A gépi nyelv, az Assembly ← → szintű nyelvek és a magas szintű nyelvek.

Forrásprogram: Magas szintű nyelven megírt program.

Fordítóprogram: Forrásprogramból gépi kódú programot készít, amelynek a ← → lépései a következők:

1. lexikális elemzés,
2. szintaktikai elemzés,
3. szemantikai elemzés,
4. kódgenerálás.

Interpreter : Nem hoz létre tárgykódot, utasításként értelmezi és ← → végrehajtja a forrásprogramot

1.3. Nyelvek osztályozása

A nyelveket három részre bonthatjuk. Az első nagyobb rész az imperatív ← → nyelvek, a második nagyobb rész a deklaratív nyelvek és az utolsó az ← → egyéb nyelvek.

Az imperatív nyelvekre a következő tulajdonságok jellemzőek:

- algoritmikus nyelvek,
- utasítások sorozatára épülnek fel,
- a fő eszköze a változók,
- a Neumann-architektúrát alkalmazza.

Két alcsoporthoz van: eljárásorientált nyelvek és az objektumorientált ← → nyelvek.

A deklaratív nyelvek tulajdonságai:

- nem algoritmikus nyelv,
- a problémát megadjuk, a megoldása pedig a nyelvi implementációkba van ← → beépítve,
- a deklaratív nyelvnél nincs sajnos lehetőségünk memóriaműveletekre.

Ennek is van kettő alcsoporthoz: a funkcionális nyelvek és a logikai nyelvek ← → .

Az utolsó csoportba pedig azok a nyelvek kerülnek, amelyek „egyéb” jelzővel ← → vannak ellátva, tehát egyik kategóriába sem sorolhatóak be.

10.3. Olvasónapló a BME C++ könyvből

1. C++ nem objektum-orientált újdonságai

A C++ nyelv a C nyelv objektum-orientált változata ezért is van az, hogy a ← → C++ programok többsége lefordul C fordítóval is. A C++ első verziójának ← →

a neve C with class volt, ami utalt az objektum-orientáltságra. A fejezet kiter azokra a gyengeségekre, amelyek a C nyelvben nem voltak még benne, de a C++-ban úgymond javítva lettek.

1. míg C-ben, ha nem írunk paramétert, akkor korlátlan számú paramétert használhatunk, C++-ban viszont azt jelenti, hogy nincs paraméter, a C++-ban a végtelen számú paramétert a ... beírásával tudjuk elérni.
2. a C-ben kihagyhatjuk a visszatérési értékek típusának meghatározását, mert intnek fogja érzékelni, C++-ban viszont nem fog lefordulni a programunk.
3. a main függvénynél a C++-ban már nem szükséges odaírni a végére a return-t, viszont a C-ben igen.
4. C-ben már elérhetővé vált a bool funkció, amivel true és false értékeket deklarálhatunk.
5. C++-ban használható már a wchar_t funkció, amivel stringeket tudunk reprezentálni. Ez C-ben csak bizonyos includeok használatával érhető el. Ezen kívül még számtalan újítás érkezett a C++-hoz, amit a fejezetben részletesen el is olvashatunk.

2. Osztályok és objektumok

Az objektumorientáltság 3 alapelve:

1. egységbázis. Olyan struktúrák, melyek az adatokat és a műveleteket tárolják.
2. adatrejtés. Nem szabad hagyni, hogy az osztályon kívülről olyan műveleteket hajtsunk végre, amelyek inkonzisztensé teszik az osztályt.
3. behelyettesítés. Lehetővé teszi, hogy speciáli osztályt használunk ott, ahol egyébként általánosabb osztályt használunk.

3. Operátorok és túlterhelésük

Operátor túlterheléshez fordulhatunk, ha beépített típusokkal olyat is szeretnénk értelmezni, ami nem része a nyelvnek. Az operátorok kiértékelésének a sorrendjét a C++ nyelv táblázata határozza meg. Az operátor felfogható speciális függvényként, azzal a különbséggel, hogy a kiértékelés szabályrendszer alapján történik.

4. C++ sablonok

A C++ nyelvben lehetőségünk nyílik arra, hogy sablonokat használunk. Ezekre néha szükség is van, hogy az osztályokat vagy a függvényeket általánosabb módon tudjuk definálni. Például veszünk egy függvényt, amit definálunk de szeretnénk, hogy int-ként és double-ként is működjön. Használhatjuk úgy, hogy lemásoljuk a funkciót, majd ahol int volt oda már double-t írunk. Ez a módszer nem hatékony, ezért helyette sémákat használhatunk. A használni kívánt típust sablonparaméterként adhatjuk meg. Ezt a módszert az osztályoknál is alkalmazhatjuk.

5. Argumentumfüggő névfeloldás

Az argumentumfüggő névfeloldás lényege, hogy ha van egy névterünk akkor, a benne lévő X osztályú objektumot ki kell íratnunk egy hatókör operátorral, hogy melyik névtérből való. Az f függvény esetén viszont nincs szükségünk erre, mert paraméterként adhatjuk neki az objektumot. A fordító az általunk megadott névtérben fog keresni, ahol megtalálja az f függvényt.

6. Típuskonverziók

A C++-ban található típuskonverzió hasonlít a C-ben ismert ↔ típuskonverzióhoz, de a C++-ban való használata sokkal biztonságosabb, ↔ továbbá megjelentek objektumorientáltságot figyelembe veő konverziók is ↔ .

DRAFT

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

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

11.5. C++

[BMECPP] Benedek Zoltán és 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 érdeklődésükért 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.