

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. május 9, v. 1.0.0

Copyright © 2019 Dr. Bátfai Norbert

Copyright © 2019 Sántha Máté Imre

Copyright (C) 2019, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com,

Copyright (C) 2019, Sántha Máté Imre., santha.mate22@gmail.com, mateme@gmail.hu,

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

<http://gnu.hu/fdl.html>

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Sántha, Máté Imre	2019. május 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-03-03	első fejezet kész	Sántha Máté Imre
0.0.6	2019-03-11	Második fejezet kész	Sántha Máté Imre
0.0.7	2019-03-18	Harmadik fejezet kész	Sántha Máté Imre
0.0.8	2019-03-26	Negyedik fejezet kész .	Sántha Máté Imre

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-03-29	olvasónapló kész	Sántha Máté Imre
0.1.0	2019-04-02	Ötödik fejezet kész	Sántha Máté Imre
0.1.1	2019-04-10	Hatodik fejezet kész	Sántha Máté Imre
0.1.2	2019-04-17	Hetedik fejezet kész	Sántha Máté Imre
0.1.3	2019-04-24	Nyolcadik fejezet kész	Sántha Máté Imre
0.1.4	2019-04-30	Kilencedik Fejezet Kész	Sántha Máté Imre
0.1.5	2019-05-05	Hibák javítása.	nbatfai
1.0.0	2019-05-08	Első teljes kiadás	nbatfai

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	10
2.7. 100 éves a Brun tétel	13
2.8. A Monty Hall probléma	14
3. Helló, Chomsky!	17
3.1. Decimálisból unárisba átváltó Turing gép	17
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	18
3.3. Hivatkozási nyelv	19
3.4. Saját lexikális elemző	20
3.5. l33t.1	21
3.6. A források olvasása	23
3.7. Logikus	24
3.8. Deklaráció	25

4. Helló, Caesar!	27
4.1. int *** háromszögmátrix	27
4.2. C EXOR titkosító	29
4.3. Java EXOR titkosító	30
4.4. C EXOR törő	32
4.5. Neurális OR, AND és EXOR kapu	34
4.6. Hiba-visszaterjesztéses perceptron	36
5. Helló, Mandelbrot!	38
5.1. A Mandelbrot halmaz	38
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	42
5.3. Biomorfok	45
5.4. A Mandelbrot halmaz CUDA megvalósítása	48
5.5. Mandelbrot nagyító és utazó C++ nyelven	54
5.6. Mandelbrot nagyító és utazó Java nyelven	54
6. Helló, Welch!	59
6.1. Első osztályom	59
6.2. LZW	60
6.3. Fabejárás	61
6.4. Tag a gyökér	62
6.5. Mutató a gyökér	62
6.6. Mozgató szemantika	63
7. Helló, Conway!	64
7.1. Hangyaszimulációk	64
7.2. Java életjáték	64
7.3. Qt C++ életjáték	65
7.4. BrainB Benchmark	66
8. Helló, Schwarzenegger!	68
8.1. Szoftmax Py MNIST	68
8.2. Szoftmax R MNIST (Passz)	69
8.3. Minecraft-MALMÖT(Passz)	69

9. Helló, Chaitin!	70
9.1. Iteratív és rekurzív faktoriális Lisp-ben(Passz)	70
9.2. Gimp Scheme Script-fu: króm effekt	70
9.3. Gimp Scheme Script-fu: név mandala	74
10. Helló, Gutenberg!	79
10.1. Programozási alapfogalmak	79
10.2. Programozás bevezetés	80
10.3. Programozás	81
III. Második felvonás	82
11. Helló, Arroway!	84
11.1. A BPP algoritmus Java megvalósítása(Passz)	84
11.2. Java osztályok a Pi-ben(Passz)	84
IV. Irodalomjegyzék	85
11.3. Általános	86
11.4. C	86
11.5. C++	86
11.6. Lisp	86

Ábrák jegyzéke

2.1. PageRank : - Kép Wikipedia:	13
2.2. Monty Hall Forrás:Wikipedia:	16
3.1. 1.	19
4.1. Forrás:Batfai Norbert	29
5.1. Mandelbrot Forrás :Wikipedia:	42
5.2. Julia halmaz Forrás :www.t-es-t.h	48
5.3. CUDA Forrás: NVIDIA Developer	53
7.1. John Horton Conway-féle életjáték Forrás : Batfai Norbert	66
7.2. BrainB Benchmark : Batfai Norbert	67
8.1. TensorFlow Forrás:SAP Blogs	68

Előszó

Amikor programozónak terveztem állni, ellenezték a környezetemben, mondván, hogy kell szövegszerkesztő meg táblázatkezelő, de az már van... nem lesz programozói munka.

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz allokálni igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Mindenesetre a programozás a teoretikus kultúra csúcsa. A GNU mozgalomban látom annak garanciáját, hogy ebben a szellemi kalandban a gyerekeim is részt vehessenek majd. Ezért programozunk.

Hogyan forgasd

A könyv célja egy stabil programozási szemlélet kialakítása az olvasóban. Módszere, hogy hetekre bontva ad egy tematikus feladatcsokrot. Minden feladathoz megadja a megoldás forráskódját és forrásokat feldolgozó videókat. Az olvasó feladata, hogy ezek tanulmányozása után maga adja meg a feladat megoldásának lényegi magyarázatát, avagy írja meg a könyvet.

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyerekeknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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 dlatex programmal a könyved pdf változatát, íme:

```
batfai@entropy:~$ cd glrepos/bhax/thematic_tutorials/bhax_textbook/
batfai@entropy:~/glrepos/bhax/thematic_tutorials/bhax_textbook$ make
rm -f bhax-textbook-fdl.pdf
xmllint --xinclude bhax-textbook-fdl.xml --output output.xml
xmllint --relaxng http://docbook.org/xml/5.0/rng/docbookxi.rng output.xml  ←
--noout
output.xml validates
rm -f output.xml
dlatex bhax-textbook-fdl.xml -p bhax-textbook.xls
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
=====
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dlatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/tree/master/Prog1/Turing/Loop>

Ahhoz hogy egy processzormagot 100%-on dolgoztassunk egy végtelen ciklusra van szükségünk . Jelen esetben egy While ciklust láthatunk . A következő esetben hogy minden magot megdolgoztassunk kellene fog az OpenMP! Itt includeolnunk kell a omp.h könyvtárat illetve a ciklusunkat a #pragma omp paralell függvényben kell elhelyeznünk . Végül pedig hogy egy mag se dolgozzon egy egyszerű sleep függvényt kell alkalmazunk illetve includeolni kell az unistd.h könyvtárat.

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

Tehát itt láthatjuk az első megoldását ! Itt egyetlen egy magot terhelünk 100%ban , nincs is másra szükségünk csak egy egyszerű while ciklus alkalmazására.

```
#include <stdio.h>  
#include <omp.h>  
  
int main() {  
  
    #pragma omp parallel  
    while(1){}
```



```
    return 0;
}
}
```

A feladat második részében láthatjuk hogy a feladat megfelelő megoldásához includeolni kellett egy könyvtárat névszerint az om.h könyvtárat . Ezután nincs is másra szükségünk csak a while programunka a while ciklus elé be kell illesztenünk a #pragma omp parallel függvényt. Ennek segítségével egyszerűen tudjuk dolgoztatni az összes magot a számítógépünkben.

```
include <unistd.h>

int main() {
    while(1) {
        sleep(1);
    }
}
```

Az utolsó részben az volt a feladat hogy egy magot 0 %ban dolgoztassunk. Ez is egy viszonylag egyszerű feladat nincs másra szükségünk mint a sleep funkcióra ennek a programba való beillesztésével elérhetjük hogy egy magunk 0%ban dolgozzon

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 készítjük most el az alábbi T1000-set, amelyben a Lefagy-ra építő Lefagy2 már nem tartalmaz feltételezett, csak konkrét kódot:

```
Program T1000
{
    boolean Lefagy(Program P)
    {
        if(P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

akkor most hogy fog működni? Sehogy, mert ilyen `Lefagy` függvényt, azaz a T100 program nem is létezik.

Velemenyem szerint Nem tudunk olyan programot írni amely képes egy előre megírt programról meghatározni hogy az a program képes lesz-e a lefutásra vagy nem lesz-e képes azaz le fog fagyni.

2.3. Változók értékének felcserélése

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

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Turing/valtozocsere>

2 változó felcserélésére többféle módszer alkalmazható. Mintpeldaul használhatunk egy segédváltozót vagy különböző műveletekkel is megoldható a feladat. Én személy szerint a második opciót választottam és különböző matematikai műveletek segítségével oldottam meg a feladatot! Lássuk is hogy működik a kód. Tehát Van 2 változónk az `a` és `b`! Az első változó azaz az `a` helyére felvesszük az `a+b` összegét azaz az első változó + a második változó összegét. Utána a `b` helyére felvesszük `a-b` összegét tehát az `a` kezdőértéke átkerült a `b`-be! Ezután az `a` változóba berakjuk az `a-b` összegét tehát az $(a+b)-((a+b)-b)$ összegét ezzel pedig a `b` kezdőértéke átkerült az `a`-ba.

```
#include <stdio.h>

int main() {

    int a = 6;
    int b = 2;

    printf("a = %d ", a);
    printf("b = %d \n", b);

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

    printf("a = %d ", a);
    printf("b = %d \n", b);

}
```

2.4. Labdapattogás

Először `if`-ekkel, majd bármiféle logikai utasítás vagy kifejezés használata nélkül írd meg egy olyan programot, ami egy labdát pattogat a karakteres konzolon! (Hogy mit értek pattogatás alatt, az alábbi videókon láthatod.)

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

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Turing/labda>

A feladatunk a labdapattogás feladat lesz. Velemenyem szerint ezzel a feladatban a bevp prog során mindenki találkozott hisz tobbszor is előkerült mind a sima mind az Ifes változata. A kódot olvasva láthatjuk hogy több mindent is deklarálnunk kell. Meg kell adnunk a labda kezdőhelyzetét ezen felül az ablak méretét illetve a labda mozgásának pattogásának mértékét. Ahogy láthatjuk a kódban egész sok függvényt fogunk alkalmazni ilyen például a refresh ami a nevéből adódóan a program frissítéséért felel ilyen a getmaxyx() amellyel a Dinamikusságot adjuk a programnak vagy éppen ilyen a mvprintw() aminek segítségével magát a labdát rajzoltatjuk ki vagy ilyen az uspeel() amellyel a labdánk gyorsaságát állíthatjuk be és továbbá ott van még az initscr függvény is amellyel a terminal méretétével van kapcsolatban. A Maga program az IF függvények használatával állapítja meg hogy éppen elérte-e a labda az adott méretű terminal valamelyik szélét. Ha elérte akkor a labda irányt vált.

```
// nem saját kód
#include <stdio.h>
#include <curses.h>
#include <unistd.h>
int main ( void )
{
    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 );

        x = x + xnov;

        y = y + ynov;

        if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
            xnov = xnov * -1;
        }
        if ( x<=0 ) { // elerte-e a bal oldalt?
            xnov = xnov * -1;
        }
        if ( y<=0 ) { //
```

```
        ynov = ynov * -1;
    }
    if ( y>=my-1 ) {
        ynov = ynov * -1;
    }
}
return 0;
}
}
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Turing/shift.c>

Tehat ebben a megoldásban a Bitshift nevezetű operátorral fogunk dolgozni. Ahogy láthatjuk a megoldás során Shiftelésre leszünk szükségünk. Ez azt jelenti hogy folyamatosan haladunk egy adott irányban. Ezt jelen esetben egy while ciklust használva fogunk megcsinálni de lehetne akár más ciklust is alkalmazni. Miután a elshifteljük 0hoz es sikeres volt a muvelet illetve nem vétettünk hibát gyakorlatilag megkapjuk a szó méretét bitben

```
#include <stdio.h>

int main() {
    int i = 1;
    int a = 0;
    while (i != 0) {
        i <<= 1;
        a++;
    }

    printf("Szóhossz: %d bit\n", a);
}
}
```

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: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Turing/Pagerank>

Szóval a Page-Rank Maga egy algoritmus amely forradalmasította illetve a mai napig meghatározza az interneten való keresést ! A legtöbb keresésre specializálódott oldal ezt használja többet között a google is , nemcsoda hisz a Google 2 alapítója hozta létre ezt az algoritmust még annó 98'ba egyetemi tanulmányaik során stanford városában Ennek segítségével kapunk számunkra megfelelő releváns oldalakat a kereséseink során !A rendszer úgy működik hogy a PageRank rendszer alapján listázza ki az oldalakat és ennek alapján Hozza fel az oldalakat a keresők számára! A rendszer szerint amelyik oldalra többen kíváncsiak azaz hivatkoznak és kattintanak rá tehát nagy a népszerűsége ezáltal nagyon magas a Pagerankja . A Programunk úgy néz ki hogy van egy végtelen ciklusunk ami gyakorlatilag a végtelenségig fut . Ebben található egy feltétel . A ciklus azonnal megszűnik amint a feltételben említett távolság kisebb mint 0.00000001. A megoldáshoz mátrixokat és vektorokat kell alkalmaznunk.A program az adatokat letárolja egy matrixba ami megmutatja az oldalak közötti helyes sorrendet , és hogy melyik oldal mutat azaz hivatkozik a másik oldalra . !Miután a programunk lefutott kifogja dobni számunkra a 4 weboldal megfelelő sorrendjét a pagerank keresési algoritmus alapján.

```
// nem saját kód

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

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

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

double
tavolsag (double PR[], double PRv[], int n)
{
    double osszeg = 0.0;
    int i;
    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

    return sqrt(osszeg);
}

int main(void) {
    double L[4][4] = {

        {0.0, 0.0, 1.0 / 3.0, 0.0},
        {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
        {0.0, 1.0 / 2.0, 0.0, 0.0},
```

```
    {0.0, 0.0, 1.0 / 3.0, 0.0}

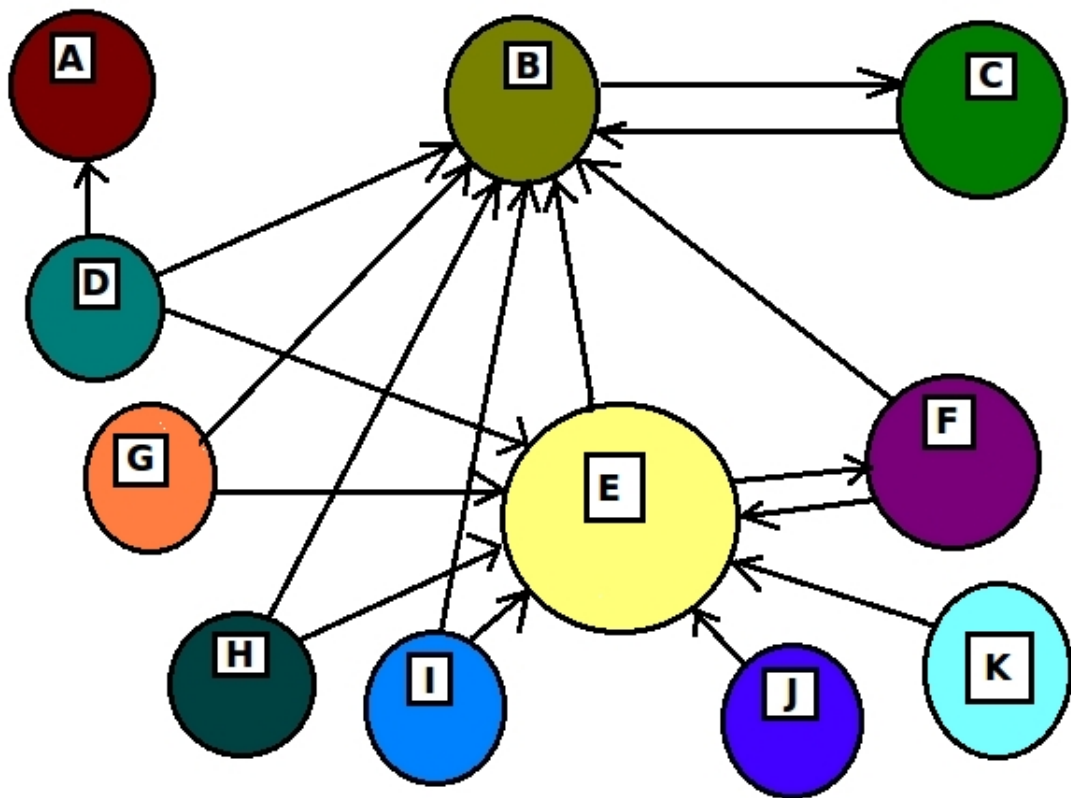
};

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

int i, j;
for (;;)
{
    for (i = 0; i < 4; ++i)
    {
        PR[i] = 0.0;
        for (j = 0; j < 4; ++j)
            PR[i] += (L[i][j] * PRv[j]);
    }
    if (tavolsag (PR, PRv, 4) < 0.00000001)

break;
    for (i = 0; i < 4; ++i)

PRv[i] = PR[i];
}
kiir (PR, 4);
return 0;
}
}
```



2.1. ábra. PageRank : - Kép Wikipedia:

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

Mielőtt belekezdünk a programban érdemes letisztázni hogy miről is kell beszélnünk. Fontos megemlítenünk a prímeket és ikerprímeket hisz ezekről szól a feladat. Tehát prímeznek azokat a (természetes) számokat nevezzük amelyeknek pontosan 2 osztóval rendelkeznek ilyen például a 3, az 5 vagy akár a 7 és a 11 is stb...! Ez a 2 osztó az egy az egy és önmaguk. Kivételül közé sorolhatjuk a 0 illetve a 1-es számot hisz ezekre igazak a feltételek viszont nem tartoznak a prím számok közé. Az Ikerprímek a prímeinek egy olyan esete amikor már tudunk 2 prímről és ezeknek a prímeinek a különbsége kettő tehát például az 5 és a 7 vagy akár a 11 és a 13 stb... Nagyon érdekes hogy még a prímekről mennyit hallunk az ikerprímek fogalma szinte ismeretlen az átlagembereknek. Gyakorlatilag szinte végtelen ikerprímről beszélhetünk, generalizálhatunk, rengeteget ismerünk, alkalmazunk, Tehát a programunk véletlenszerűen generál számunkra prímeket. Miután ezzel végzett elkezdődik összeveti a megadott feltételek alapján a prímeket. Megvizsgálja hogy mekkora az első és a második prím különbsége. Ha nem kettő akkor a program dolgozik tehát ugrik a következő 2 prímre és vizsgálja azokat még nem talál ikerprímet. Ha talál tehát a vizsgálat alap-

jan a kettő különbsége kettő lesz . Eltárolja az adatokat a megfelelő helyekre A program is ezen az elven működik megvizsgálja hogy az első és a második prímnek mekkora a különbsége.Eltárolja az adatokata megfelelő helyekre majd ezéből az adatokból reciprokokat képez , összeadja őket és visszaadja az értéket.Miután megfelelő mennyiségű adatot gyűjtött a a program megfelelő működéséhez a már számítógépes matematika és vizualizáció óráról is ismert plot függvénnyel kirajzoltatja nekünk a megfelelő eredményt.

```
// nem saját kód
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

Ez egy nagyon érdekes televíziós elkezelés . Több televíziós műsorban is megfigyelhető volt az évek alatt. Az elkezeles szerint 3 különböző ajtó található és a játék folyamán ezek közül kell választanunk a megfelelőt , az egyik ajtó mögött egy nyeremény vagy egyéb érték található a másik kettő ajtó mögötti tér viszont üres ! A játékos célja hogy eltalálja a megfelelő ajtót , azaz ,megnyerje a játékot , értékes nyereményekkel legyen gazdagabb a játék után. . Tehat a játékot magát egy musorvezető irányítja az ő feladata nagyon fontos lesz . Az ő jelzésére kell a játékosnak kiválasztani egy ajtót ..Miután ez megtörtént A musorvezető kinyitja az egyik üres ajtót azaz most mar csak kettőből kell eltalálni a megfelelő ajtót , tehát a legtöbb ember szerint most mar 50%-50% esélyünk van a nyerni játékosként . Gyakorlatilag viszont ha a megváltoztatjuk az eredetileg alkotott döntésünk és a fentmarad két ajtóból azt az ajtót választjuk amelyre eddig nem szavaztunk 2/3 esélyünk van a nyeremény megnyerésére .A program elég sok random esetet generál amelyekben kulonbozo variaciok és kombinaciok játszódnak le . Meg kell határozunk az esetek számát hogy ne a végtelenségig generalja az eseteket .Ezután egy ciklussal végig megyünk és megnezzuk a játékok által adott eredményeket .Hogy éppen az adott játékban ki kapott pontot az ekkora kapott pontokat variaciokat kombinaciokat , eredményeket kiértékeljük Majd az összegzett eredményeket kiíratjuk.

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

for (i in 1:kiserletek_szama) {

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

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

  }else{

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

  }

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

}

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

for (i in 1:kiserletek_szama) {

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

}

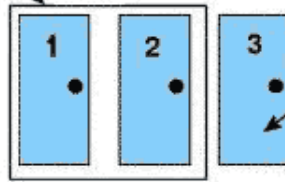
valtoztatesnyer = which(kiserlet==valtoztat)

sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)

}
}
```

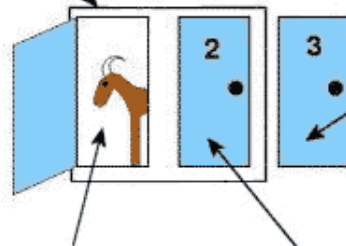
2/3 eséllyel itt a kocsi

1/3 eséllyel itt



2/3 eséllyel itt a kocsi

1/3 eséllyel itt



0 eséllyel itt, tehát 2/3 eséllyel itt

2.2. ábra. Monty Hall Forrás:Wikipedia:

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Turingfeladat>

Tehat a feladatban decimalis számrendszerből kell átváltanunk Unárisba. Tehat első részben érdemes tisztázni az alapfogalmakat ebben az esetben is . A decimalis számrendszer másnéven 10es számrendszer az a számrendszer amelyet a mindennapokban használunk ez "Default" számrendszer a mindennapokban véleményem szerint habar lehet valaki valahol nem ezt használja .De azért nagytobbsegeben szerintem igen. Az unáris azaz az egyes számrendszer azaz az egyes számrendszert ritkábban használjuk , alkalmazzuk de attól függetlenül hogy ismeretlennek hangzik mindenki találkozott már vele valószínűleg ha nem is magas szinten alkalmazta viszont ezt a palcikas számolás módszerrel élete során 1-1 alkalommal biztosan használta . Ez a legegyszerubb számrendszer fontos hangsúlyozni hogy kizárólag természetes számok alkalmazására lehet használni ! Tehat például törteket nem tudunk vele leírni. Lassuk is a kódot tehát Gyakorlatilag eben a számrendszerben palcikák , vonalak segítségével tudunk leírni számokat !(Egy I , kettő II , öt IIIII , 10 IIIII IIIII) programunk úgy mukodik hogy bekérjük a számot jelen esetünkben ez az "a" változóba kerül. A program megvizsgálja hogy tényleg szám-e .Erre azért van szükség mert nem megfelelő bemeneti adatok esetében a program nem képes a megfelelő lefutásra.Ha a program és a bemeneti adatok megfelelőek akkor a ciklus annyiszor fut le amekkora a bekert számunk azaz amennyi a változo értéke . A programunk minden lefutás után ír egy I-t az outputra azaz a megadott kiementre. Megfelelő lefutás esetén a palcikák száma megegyezik a változóban szereplő szám értékével tehát ha a mi számunk a hármas volt akkor az outputon III ennek kellett megjelennie.

```
// nem saját kód
#include <stdio.h>
#include <iostream>

using namespace std;
int main()
{
```

```
int a=0
cout << "Írjon be egy számot : "<< '\n';
cin >> a ;
for (int i =0 ; i<a , i++ ) {
    cout << "I" ;

}

cout<< '\n';

}
}
```

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

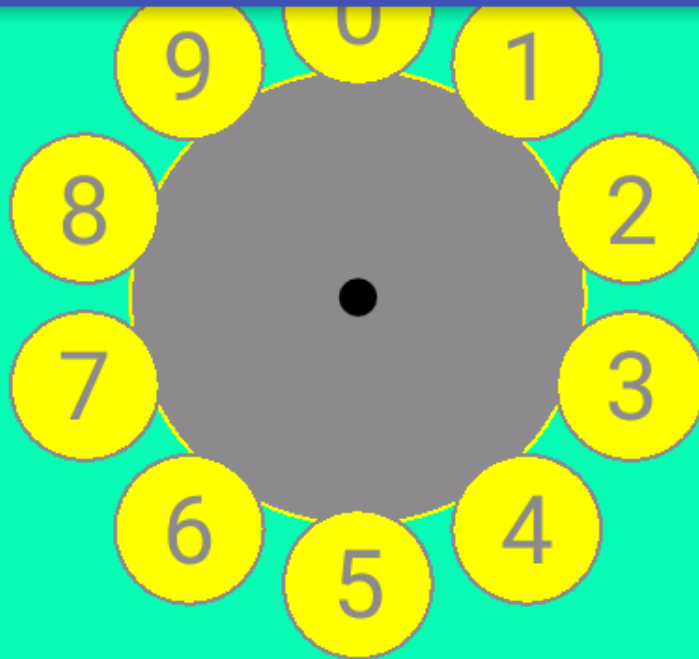
Mutass be legalább két környezetfüggő generatív grammatikát, amely ezt a nyelvet generálja! (Passzolva)

Megoldás videó:

Megoldás forrása:

9:00

SMNISTforHumansExp3, v0.0.3



ms: (600) 620 629 600 0 0 0 0

|v|/[...]: (6) 4/1 5/2 6/1 0/0 0/0

3.1. ábra. 1.

3.3. Hivatkozási nyelv

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

Megoldás videó:

<https://github.com/Samate99/Hello-world/blob/master/Prog1/C89.c>

A BNF az egy ugynevezett metszaniaxis amely a különbozo nyelvtanok leírásához használnak illetve alkalmaznak. Fontos megjegyezni hogy talan ez a metaszinaxis a legyakoribb amint használnak különböző programozási nyelvek nyelvtanainak megfogalmazásához , leírására , igazolására . .Tehat a feladat "feladat" része az hogy a C89 és C99 között kell talanunk különbségeket . Fontos letisztázni hogy nagy mértékben kötődnek egymashoz hisz ez a 2 nyelv 2 különböző változata. Az egyik a már napjainkra kissé elavult a masikat pedig jelenleg is használjuk. Tehat a lenyeg hogy mindkettő a C nyelv egy változata . Ami példánkban van egy kis különbség a 2 szabvány között !A probléma a for cikluson belüli deklaráció , tehát korábbi "C" verizóba mindent fontos volt deklarálni a ciklus előtt hogy a program megfelelően lefusson . Ezzel ellentétben az új kiadásban már akár a cikluson belül is deklarálhatunk nem fogja befolyasolni a program felutását .

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Chomsky/lx.1>

Tehat az elsőre bonyolultnak tűnő lexikálásról lesz szó tehát az a kérdés hogy mi is fog történni. Ha megfelelő a lefutás . A lefutás után a %-jelek között található kódból a lexel létre fog nekünk hozni egy C programot . Egy olyan programot amely a beírt szöveget egy adott szempont szerint vizsgálja .A mi esetünkben valós számokat keres , rögzíti őket illet összeadja őket A program kezdetekor a realnumbers változót nagyon fontos hogy deklaráljuk 0 kezdőértékkel . Ha más értéket adunk meg akkor nem valós adatokat fog megadni a program. Ezután a digit definiáljuk ! Ez lényegében ez egy sablon lesz aminek a segítségével fogja atnezní , és vizsgálni a később számamam megadott szöveget . Tehat ennek a segítségével próbálja kiszurni számunkra a valós számokat Itt megadjuk neki hogy milyen karaktercsoportokra mit adjon vissza tehát .Tehát a mi esetünkben hogy mi az általános leírása egy valós számnak. Ha a később megadott szövegben talál a sablonnak megfelelő adatot akkor realnumbers változó értéke megnő 1-el . Tehat ha talál 5 dolgot amely megfelel az általunk megadott sablonnak akkor a realnumbers értéke 5el fog megnőni. és kiiratjuk ! Végül kiemelném a nagyon fontos yylex() függvény alkalmazását , ennek a segítségével indítjuk el a lexikálást tehát ha ezt nem alkalmazzuk a programunk nem fog megfelelően lefutni .

```
// nem saját kód
#include <stdio.h>
int realnumbers = 0;

%}

digit [0-9]
%%

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

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

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Chomsky/l33t.l>

Ezt programot az előzőhöz hasonlóan a lexikális elemző generátor segítségével fogjuk elkészíteni. Tehát itt is lexelni fogunk. Tehát ebben az esetben is a lexel segítségével fogunk létrehozni egy C kódot. A program első felében létrehozunk egy úgynevezett struktúrát, ami tartalmaz 2 nagyon fontos részt. Tehát első sorban tartalmazza a cserélendő karaktert, illetve tartalmaz 4 karaktert, amik közül véletlenszerűen fogunk választani és ezeket a karaktereket fogjuk berakni a cserélendő karakterek helyére! A program második részében található egy random generator, ami a megadott intervallumban generál különböző számokat számunkat és ezek kiértékelésével foglalkozik. A random generatorban generált random szám segítségével fogja eldönteni a programunk, hogy melyik cserélendő karaktert melyik megfelelő csere karakterre cserélje a program felütés során. Szóval összegezve a program a random generált szám segítségével kiválaszt egy karaktert az egyik struktúrából, majd ezt a karaktert a megfelelő helyre a másik struktúrába athelyezi és így keveri össze a szöveget. Végül pedig az előző feladathoz hasonlóan itt is kiemelném, hogy használnunk kell az yylex() függvényt a program megfelelő működéséhez, mivel ha ezt elhanyagoljuk, akkor a program nem fog lefutni.

```
// nem saját kód  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <ctype.h>  
  
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))  
  
struct cipher {  
    char c;  
    char *leet[4];  
} l337d1c7 [] = {
```



```

{'a', {"4", "4", "@", "/-\\\"}},
{'b', {"b", "8", "|3", "|\"}},
{'c', {"c", "(", "<", "{"}},
{'d', {"d", "|)", "|]", "|\"}},
{'e', {"3", "3", "3", "3\"}},
{'f', {"f", "|=", "ph", "|#\"}},
{'g', {"g", "6", "[", "[+"}},
{'h', {"h", "4", "|-|", "[-\"}},
{'i', {"1", "1", "|", "!"}},
{'j', {"j", "7", "_|", "_/"}},
{'k', {"k", "<", "1<", "|{"}},
{'l', {"l", "1", "|", "|_"}},
{'m', {"m", "44", "(V)", "\\|\"}},
{'n', {"n", "\\|", "/\\/", "/V\"}},
{'o', {"0", "0", "()", "[]\"}},
{'p', {"p", "/o", "|D", "|o\"}},
{'q', {"q", "9", "O_", "(,)"}},
{'r', {"r", "12", "12", "|2\"}},
{'s', {"s", "5", "$", "$\"}},
{'t', {"t", "7", "7", "'|'\"}},
{'u', {"u", "|_|", "(_)", "[_\"}},
{'v', {"v", "\\\/", "\\\/", "\\\/\"}},
{'w', {"w", "VV", "\\\/\\\/", "(/\\\"}},

{'x', {"x", "%", ")(", ")(\"}},
{'y', {"y", "", "", ""}},
{'z', {"z", "2", "7_", ">_"}},

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

};
%}
%%
{
    int found = 0;
    for(int i=0; i<L337SIZE; ++i)
    {
        if(l337d1c7[i].c == tolower(*yytext))
        {
            int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

```

```
        if(r<91)
            printf("%s", l337d1c7[i].leet[0]);
        else if(r<95)
            printf("%s", l337d1c7[i].leet[1]);
        else if(r<98)
            printf("%s", l337d1c7[i].leet[2]);
        else
            printf("%s", l337d1c7[i].leet[3]);
        found = 1;
        break;
    }
}
if(!found)
    printf("%c", *yytext);
}
%%
int main()
{
    srand(time(NULL)+getpid());
    ylex();
    return 0;

}
```

3.6. A források olvasása

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

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

Ha a SIGINT jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a jelkezeslo függvény kezelje. (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 meggyránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

- i.

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

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

```

iii. for(i=0; i<5; i++)
iv.  for(i=0; i<5; tomb[i] = i++)
v.   for(i=0; i<n && (*d++ = *s++); ++i)
vi.  printf("%d %d", f(a, ++a), f(++a, a));
vii. printf("%d %d", f(a), a);
viii. printf("%d %d", f(&a), a);

```

Megoldás forrása:

Megoldás videó:

1. Hat a Sigint jel figyelve volt tehát nem volt figyelmen kívül hagyva akkor az adott jelkezelő függvény kezelje, de ha figyelmen kívül volt hagyva akkor ő is hagyja figyelmen kívül.
2. Hat a második kód nem túl nehéz egy ciklusról beszélhetünk ami 4-ig megy. A ciklus addig fog lefutni még az i értéke kisebb mint 5. Ha igaz a feltétel növeli az i értékét.
3. Szerintem ez a 2. kód ugyanaz azzal a kivetéssel hogy az i végső értéke nem 5 lesz mint az előző esetben hanem 4.
4. A ciklus során elemeket kepezünk, Minden lefutás után egyre nagyobb és nagyobb elemeket kapunk és ezeket betöltjük a Tomb megfelelő elemébe
5. A következő kódban is egy ciklussal találkozhatunk itt addig megy a folyamat még az i kisebb mint n illetve a d eleme megegyezik az s elemével
6. Ebben az esetben kiíratunk 2 számot amelyet az F függvény határoz meg
7. Itt is kiíratunk 2 értéket az egyik az $F(a)$ értéke lesz a másik pedig az $'a'$ értéke
8. Ebben a helyzetben is kiíratunk 2 értéket ebben az esetben az F nagy mértékben meghatározza az a értékét

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```

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

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

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

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

```

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 Az elsőben véleményem szerint az olvasható hogy a prímszámok száma végtelen , Azaz Végtelen sok prímszám van. . A másodikban ugyanugy a prímszámok számáról van szó , Itt arra utal hogy a Ikerprímek száma végtelen . Ez természetesen igaz hisz ha a primszamok száma végtelen akkor az ikerprímek szama is vegtelen A harmadik illetve negyedik esetben csak a "nyers" kiolvasás az eltérő mindkettőt ugy lehet fordítani hogy "véges sok prímszám van"

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```
- ```
int *b = &a;
```
- ```
int &r = a;
```
- ```
int c[5];
```
- ```
int (&tr)[5] = c;
```

- `int *d[5];`
- `int *h ();`
- `int *(*l) ();`
- `int (*v (int c)) (int a, int b)`
- `int ((*z) (int)) (int, int);`

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Egy változót A néven

Egy változóra mutatót

A változó azaz a referenciáját

A következő példában egy öt elemből álló tömbel találkozhatunk.

Az előző feladatban található tömb referenciáját lathatjuk.

Az öt elemből álló tömbre mutató mutatót lathatunk.

Egy függvényre mutatót lathatunk .

Egy függvényre mutató mutató függvényt.

Egy olyan mutató függvényt amely visszaad egy C-t és kap 2 változót az a-t és a b-t

Egy Függvényre mutató függvény mutatót ami az elozohoz hasonlóan visszaad egy értéket azaz egy változót és visszaad 2 változót .

## 4. fejezet

# Helló, Caesar!

### 4.1. int \*\*\* háromszögmátrix

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Ceaser/matrix.c>

Tehat a feladatban egy háromszög Matrixot kell készítenünk. Mátrixokkal már mindenki találkozott a többség talán már gimnáziumban de Diszkrét Matematika órán biztos mindenki találkozott velük .Illetve csodálhatta meg őket. A Háromszög mátrix egy olyan specialis , negyzetes matrix aminek az összes eleme a főatló felett vagy a főatló alatt 0 . Az programnak egy ilyen ennek megfelelő matrixot kell hogy létrehozni illetve használni , . A matrixunk a ebben az esetben 5 sorból fog állni , és mivel már korábban jeleztem hogy ez egy negyzetes mátrix így a sorok / oszlopok számának egyenlőnek kell lennie Tehat 5 sorból és 5 oszlopból kell állni. A megfelelő könyvtárakat ebben a programban is érdemes megfelelően includeolni mert anélkül nem valószínű hogy megfelelő eredményre jutunk. Az oldalak és sorok elkészítéséhez egyetemesen hogy pointereket és ciklusokat kell alkalmaznunk. Deklarálnunk kell az nr-t ami a mi esetünkben 5 lesz . Az nr gyakorlatilag a sorok számát határozza meg a ez a mi esetünkben 5. Ezt fontos hogy a mainen belül vegyük fel . Ahogy a lenti kódon láthatjuk több ciklust is alkalmaznunk kell pontosan For ciklusokat . A kódban láthatjuk hogy a tm értéket többször is ki fogjuk iratni . Láthatjuk hogy több Ciklus is NR-ig azaz 5ig mennek és utána lép ki . Fontos kiemelni a malloc függvényt ennek a segítségével foglalunk le memóriát . Láthatjuk hogy ha ez nem sikerül akkor a program -les hibakóddal kilep . Láthatjuk hogy a for ciklus segítségével létrehozjuk a matrixot . A program végén felszabadítjuk a Malloc függvénnyel lefoglalt memóriát majd befejezzük a programot

```
// nem saját kód
#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)

{

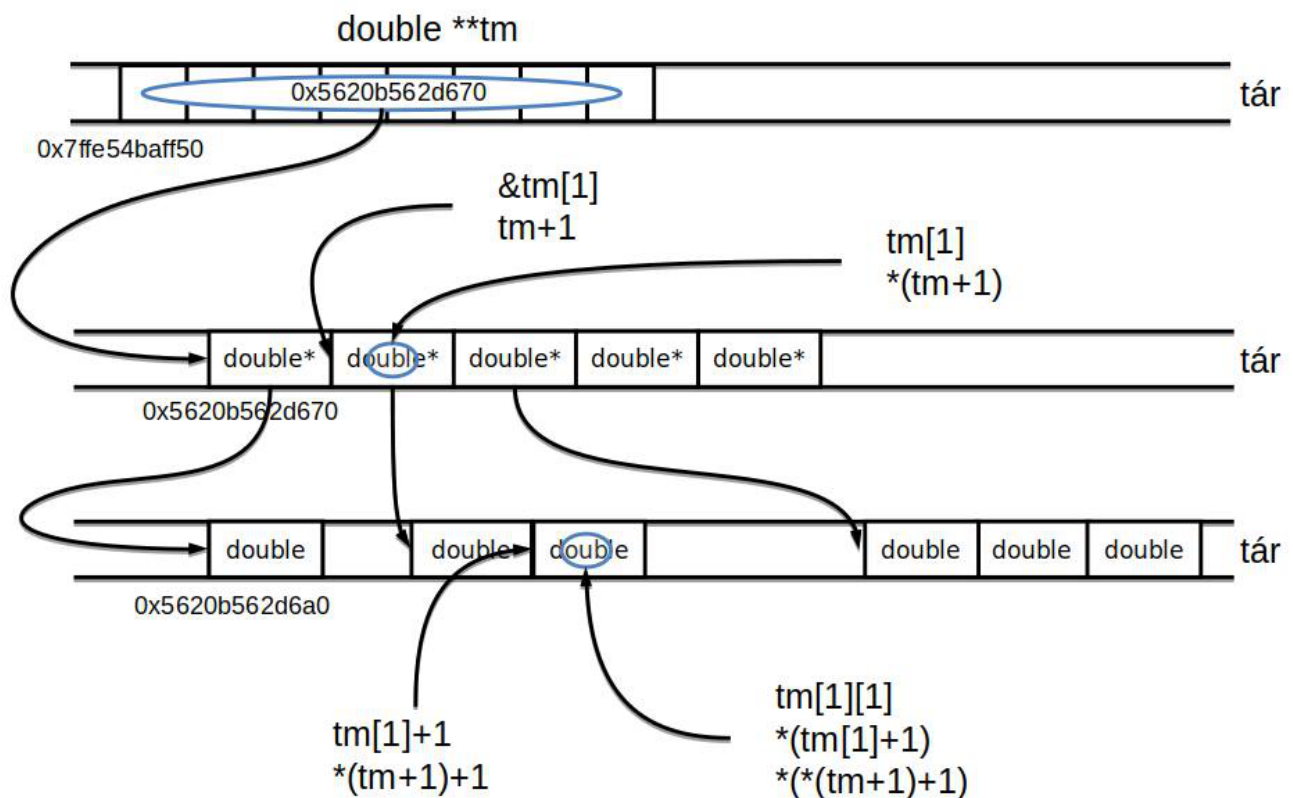
 for (int j = 0; j < i + 1; ++j)
 printf ("%f, ", tm[i][j]);
 printf ("\n");
}
```

```

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

return 0;
}
}

```



4.1. ábra. Forrás: Batfai Norbert

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/EC>

Tehát a feladatunk egy C exor titkosító létrehozása. A feladatban meg kell adnunk 2 nagyon fontos dolgot egy szöveget amit a program letitkosít és egy kulcsot amiből a karaktereket nyeri. A program az előre megadott kulcsban szereplő karakterekkel exorozza, keveri össze a szöveget. Ennek a segítségével hozzuk létre az exorozott szöveget. Nagyon érdekes hogy egyes kulcsok megadásával milyen érdekes kódolt szövegek jönnek létre. A programhoz szükség van 2 TXT állományra az egyikbe a tiszta szöveget kell tennünk, a másikba pedig később fog kerülni a titkos szöveg. A programhoz includeolnunk kell a megfelelő



könyvtárakat . Fontos a mainbe beleírunk hogy argumentumokkal dolgozunk fontos definálni a kulcs méretet illetve a buffer méretet . Illetve a megfelelő dolgokat deklarálnunk kell. Beolvassuk a szöveget illetve a kulcsot . Utana a kódban láthatjuk ahogy egy ciklus segítségével exorozza össze a kulcs karaktereit a szöveg karaktereivel és hozza létre a titkos szöveget .

```
// nem saját kód
#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);
 }
}
```

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/exor.java>

Gyakorlatilag ugyanugy működik mint a C program. Megadunk egy kulcsot és egy tiszta szöveget , a kulcs karaktereit összeexorozzuk a szöveg karaktereivel és ebből kapunk egy olvashatatlan titkos szöveget. A program itt is ugyanugy működik meg kell adnunk az alap adatokat , deklarálnunk kell . Bekerül a szöveg a bufferbe . Utána következik while ciklus itt a kulcs segítségével elkészítjük a kódolt, exorozott szöveget ! Illetve láthatjuk hogy ebben az esetben bekerült egy Try-Catch a kódba. Fontos kiemelni hogy mivel ebben az esetben egy másik nyelvben dolgozunk a programnak más formátumban kell tarotlunk tehát jelen esetben .java lesz a fájlformatum

```
// nem saját kód
public class ExorTitkosito {

 public ExorTitkosito(String kulcsSzoveg,
 java.io.InputStream bejovoCsatorna,
 java.io.OutputStream kimenoCsatorna)
 throws java.io.IOException {

 byte [] kulcs = kulcsSzoveg.getBytes();
 byte [] buffer = new byte[256];
 int kulcsIndex = 0;
 int olvasottBajtok = 0;

 while((olvasottBajtok =
 bejovoCsatorna.read(buffer)) != -1) {
 for(int i=0; i<olvasottBajtok; ++i) {

 buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
 kulcsIndex = (kulcsIndex+1) % kulcs.length;
 }
 kimenoCsatorna.write(buffer, 0, olvasottBajtok);
 }
 }

 public static void main(String[] args) {
 try {
 new ExorTitkosito(args[0], System.in, System.out);

 } catch(java.io.IOException e) {
 e.printStackTrace();
 }
 }
}
```

## 4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/TC>

A feladatunk egy C exor Toró lesz . a különböző jelszó /kulcs feltörések nagyon érdekesek illetve nagyon érdeklik az embereket . Több féle feltörési módszer létezik van melyik random próbálgat jelszavakat / kulcsokat , van amelyik valamilyen algoritmus alapján próbálgatja a kulcsokat/ jelszavakat , vagy van a mindenki számára ismert bruteforce módszer ebben az esetben a program az összes lehetséges variációt / kombinációt kipróbálja és így próbálja feltörni a dolgokat. . Amint megtalálja a kulcs segítségével visszaalakítja a szöveget az eredeti állapotába . Nagyon fontos hogy előre meg kell adnunk hogy hány karakterből áll a kulcs.Ahogy láthatjuk a programban miután beolvassuk a szöveget egy behívott függvény segítségével és megadtuk a kulcs méretét a program elkezd generalni a kulcsokat és elkezd próbálgatni a szövegen . Az exor addig próbálja feltörni a szöveget ameddig vagy le nem alltjuk vagy nem sikerül neki. Ez a módszer rengeteg erőforrást illetve időt igényel emiatt sajnos sok időbe is telhet egy olyan meretű kulcs megtalálása. A program az exor\_tores segítségével ellenőrzi hogy sikeres volt e a művelet vagy haladhat tovább a következőre Ha sikerült akkor a program vagy a standard outputra vagy az előre definiált helyre kiírja a titkos szöveget

```
// nem saját kód
#define MAX_TITKOS 4096

#define OLVASAS_BUFFER 256

#define KULCS_MERET 5

#define _GNU_SOURCE

#include <stdio.h>

#include <unistd.h>

#include <string.h>

double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
 int sz = 0;
 for (int i = 0; i < titkos_meret; ++i)
 if (titkos[i] == ' ')
 ++sz;

 return (double) titkos_meret / sz;
}
```

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

 double szohossz = atlagos_szo_hossz (titkos, titkos_meret);

 return szohossz > 6.0 && szohossz < 9.0

 && strcasestr (titkos, "hogya") && strcasestr (titkos, "nem")
 && strcasestr (titkos, "az") && strcasestr (titkos, "ha");
}

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

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

int
xor_tores (const char kulcs[], int kulcs_meret, char titkos[],
 int titkos_meret)
{
 xor (kulcs, kulcs_meret, titkos, titkos_meret);
 return tisztalehet (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 (char ii = 'A'; ii <= 'Z'; ++ii)
 for (char ji = 'A'; ji <= 'Z'; ++ji)
 for (char ki = 'A'; ki <= 'Z'; ++ki)
 for (char li = 'A'; li <= 'Z'; ++li)
 for (char mi = 'A'; mi <= 'Z'; ++mi)
 {
 kulcs[0] = ii;
 kulcs[1] = ji;
 kulcs[2] = ki;
 kulcs[3] = li;
 kulcs[4] = mi;

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

 // ujra EXOR-ozunk, igy nem kell egy masodik buffer
 exor (kulcs, KULCS_MERET, titkos, p - titkos);
 }
return 0;
}
}
```

## 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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R)

A Neuralis halók nagyon érdekesek . Nezzük is meg a kódot . A kódban azt láthatjuk hogy a neuralis halók bementei adatokat kapnak és logikai értékeket adnak vissza . Ennek a programnak a segítségével nagyon egyszerűen tudunk előállítani rendkívül bonyolult logika kódokat . Tudhatjuk hogy véges számú összesen 7 DB neuralis kapuról beszélhetünk . ezek az AND OR NOT NAND NOR EXOR EXNOR A kódot olvasva láthatjuk hogy mi ezek közül hármat használunk az OR-t and AND-et és a Exor-t.

```
Program T100
```

```
Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.
#
This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
#
You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>
#
https://youtu.be/Koyw6IH5ScQ
```

```
library(neuralnet)
```

```
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])
```

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

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

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

```
plot (nn.orand)

compute (nn.orand, orand.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])

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=c(6, 4, 6), 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++

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Ceaser/perceptron.cpp>

Szóval a feladatunk hasonló az előzőhöz itt is hasonló témakörben fogunk foglalkozni . Gyakorlatilag a program egy gepi tanulással kapcsolatos algoritmus . Tehat a feladatunk nagy mértékben kötődik az előzőhöz , akar beszélhetünk arról is hogy az előző egy változata . Az első kérdés talan az lehet az olvasóba hogy mi is a perceptron. A perceptron egy tanulással kapcsolatos algoritmus . Kiemelném hogy a perceptron

létrejött rengeteg embernek köszönhetjük de az első változatának elkészítője Perceptron Rosenbalt volt . Aki ki nem találta a nevét is volt . A program elég bonyolult de szerintem mindenki számára látható hogy körülbelül 3 reszre oszthatjuk a kódot . Lathatunk egy részt amelyben ahogy látjuk fogadjuk az adatokat lathatunk egy második részt amely képes lesz összegezni az adatokat illetve eltarolni illetve van egy harmadik rész amely képes döntéseket hozni és lathatjuk hogy képes kezelni az adatokat.

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

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

 png::image <png::rgb_pixel> png_image (argv[1]);
 int size = png_image.get_width()*png_image.get_height();
 Perceptron* p = new Perceptron(3, size, 256, 1);
 double* image = new double[size];

 for (int i{0}; i<png_image.get_width(); ++i)
 for(int j{0}; j<png_image.get_height() ++j)
 image[i*png_image.get_width()+j] = png_image[i][j].red;
 double value = (*p) (image);
 std::cout << value << std::endl;

 delete p;
 delete [] image;

}
}
```



## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/mandelpngt.c%2B%2B>

Első lépésként itt is fontos tisztázni hogy mi is az ugynevezett Mandelbrot halmaz . Ez gyakorlatilag egy a komplex számsíkon talált halmaz . Felfedezését Benoit Mandelbrotnak köszönhetjük aki 1980ban találta meg . Tehetjük fel a kérdést hogy miért is fontosak számunkra a komplex számsíkon található számok ! Hát azért mert rengeteg olyan matematika példa és feladat van amelyekre egyéb számhalmazokban nem tudnánk választ vagy pontos választ adni. Ez az a számhalmaz ahol az értelmezhetetlen feladatok értelmezhetőek és szinte minden problémát tudunk orvosolni. Mielőtt belekezdünk a kódok világát böngészni fontos kiemelni hogy a program megfelelő futtatásához a linpng és a linpng++ illetve a png++ telepítése nem elhanyagolható ! Tehát ezek nélkül az eredményünk nem lesz valós ! Tehát a kódban láthatjuk hogy előre meg kell adnunk a képünk méretét jelen esetben ez egy 600x600as kép lesz. Tehát a programban létrehozunk egy rácsot amelyen később különböző pontokat veszünk fel. A programban megfigyelhetünk 2 ciklust amelynek jelentős szerepe lesz a programban. Ha jól megfigyeljük láthatjuk hogy ez a 2 ciklus megy végig és számolja ki a megfelelő irányokba . Ezután felfedezhetünk egy While ciklust is amely a rácsponthoz ellenőrzésre szolgál . Ennek a segítségével állapítja meg a program hogy az előbb kiszámolt rácsponthoz van-e az előre definiált számsíkon. Ha rajta van akkor természetesen lementi az adatokat ha nincs rajta akkor pedig nyilván továbbhalad . Ezzel a módszerrel folyamatosan számolja ki a rács egyre több és több pontját . Miután ezzel végzett és feltöltötte a kép pixeleit a program a már előre megadott helyre létrehozza számunkra ezt a csodás képet. Ezt a halmazt úgy tudjuk létrehozni hogy egy 4 oldalhosszúságú négyzetben lefektetünk egy rácsot . Kiszámoljuk hogy a rács pontjai mely komplex számoknak felelnek meg .

Program T100

```
/ mandelpngt.c++
// 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
//
// Mandelbrot png
// Programozó Páternoszter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↵
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRU8
//

#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, ujureZ, 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 kiindulási 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
 ujureZ = reZ * reZ - imZ * imZ + reC;
 ujimZ = 2 * reZ * imZ + imC;
 reZ = ujureZ;
 imZ = ujimZ;
 ++iteracio;
 }

 kepadat[j][k] = iteracio;
 }
}

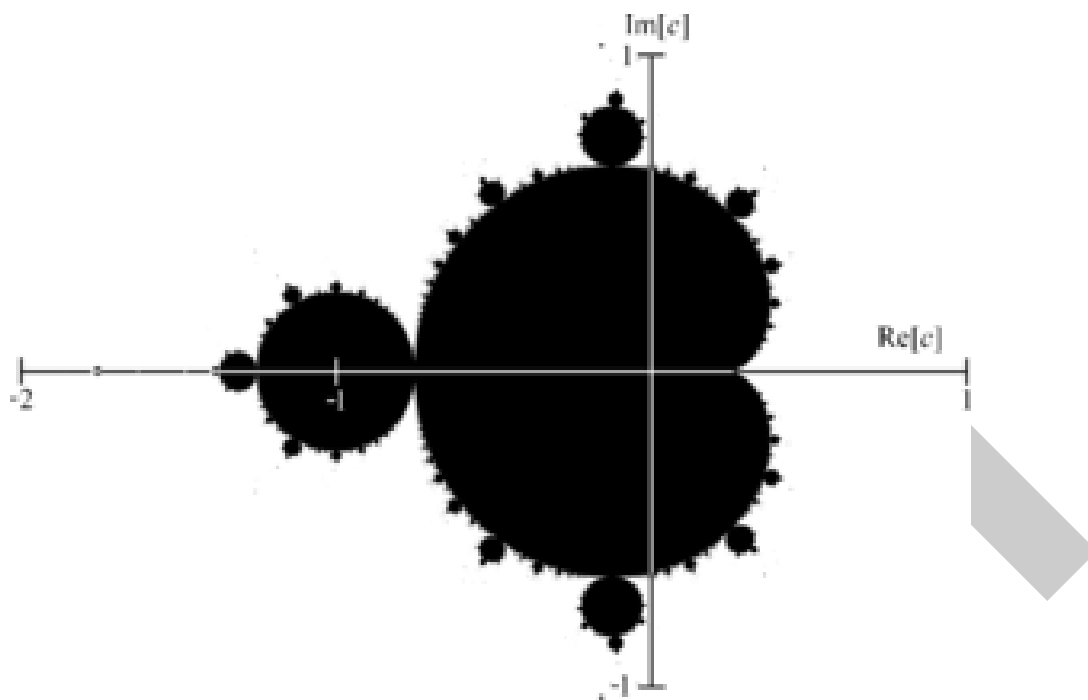
times (&tmsbuf2);
```

```
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: ./mandelpng 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;
}
}
```



5.1. ábra. Mandelbrot Forrás :Wikipedia:

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

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/mandel.cpp>

Tehat az előző feladatban már beszéltünk a Mandelbrot halmazról illetve a komplex számsíkról. Itt külön nem emelném ki a történetét illetve a komplex számsík és a halmaz jelentőségét. A programunk egyszerűen kifogja számolni az előzőhöz hasonlóan a mandelbrot halmazt viszont ebben az esetben az `std::complex` osztályt fogja segítségük hívni a feladathoz. Tehat itt is kiemelném hogy ehhez a programhoz is kellene a már fentebb említett könyvtárak illetve egyéb eszközt telepítése hisz nélkülük ez a program se fog megfelelően lefutni. Szóval mivel itt külön az `std::complex` osztályt kell használnunk érdemes első lépésként ezt importálnunk a programba. Ahogy az előző feladatban ebben is előre meg kell adnunk azaz definíálni kell a kép adatát. A program hasonló képpen működik mint az előző program tehát ebben az esetben is for ciklusok segítségével megy végig a pontokon, egy while ciklussal ellenőrzi magát illetve tölti fel a jónak talált pontokat. Fontos kiemelni hogy a programnak van egy része amely folyamatosan kiírja számunkra hogy hány %nál jár a programunk. Végül de nem utolsó sorban szeretném kiemelni hogy az előzővel ellentétben az itt használunk színezést emiatt ez a csodás kép egy színes kép lesz. Ha mindent jól csináltunk mindent megfelelően leírtunk, deklaráltunk akkor a program lefutása után itt is egy képpel lehetünk gazdagabbak valamelyik meghajtón.

Program T100

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

```
// ./3.1.2 mandel.png 1920 1080 2040 ↵
-0.01947381057309366392260585598705802112818 ↵
-0.0194738105725413418456426484226540196687 ↵
0.7985057569338268601555341774655971676111 ↵
0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ↵
0.4127655418209589255340574709407519549131 ↵
0.4127655418245818053080142817634623497725 ↵
0.2135387051768746491386963270997512154281 ↵
0.2135387051804975289126531379224616102874
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1 --left-footer=" ↵
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.

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

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

 int szelesseg = 1920;
 int magassag = 1080;

 int iteraciosHatar = 255;

 double a = -1.9;
 double b = 0.7;
```

```
double c = -1.3;
double d = 1.3;

if (argc == 9)
{
 szelesseg = atoi (argv[2]);
 magassag = atoi (argv[3]);
 iteraciosHatar = atoi (argv[4]);

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

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

double dx = (b - a) / szelesseg;
double dy = (d - c) / magassag;
double reC, imC, reZ, imZ;
int iteracio = 0;
std::cout << "Szamitas\n";
// j megy a sorokon

for (int j = 0; j < magassag; ++j)
{
 // k megy az oszlopokon
 for (int k = 0; k < szelesseg; ++k)
 {
 // c = (reC, imC) a halo racspontjainak
 // megfelelo komplex szam

 reC = a + k * dx;
 imC = d - j * dy;

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

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

while (std::abs (z_n) < 4 && iteracio < iteraciosHatar)
{
 z_n = z_n * z_n + c
 ++iteracio;
}
kep.set_pixel (k, j,

 png::rgb_pixel (iteracio%255, (iteracio*iteracio) <-
 %255, 0));

}
int szazalek = (double) j / (double) magassag * 100.0;
std::cout << "\r" << szazalek << "%" << std::flush;
}
kep.write (argv[1]);
std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
}
```

## 5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbRzY76E>

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/Biomorf](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf)

Tehetjük fel hogy honnan is erednek a biomorfok . Hát gyakorlatilag egy Hiba , tévedés miatt találtak ra ezekre mint egyéb más dolgokra. Clifford Pickover volt az személy aki megpróbálta a Julia halmazokat kirajzoltatni sajnos az általa megírt program több helyen is hibákkal rendelkezik de szerencséjére ez hozta meg számára sikert hisz ezzel a programmal tudta kirajzoltatni a biomorfok első "csapatát" Tehat itt is rajzoltatni fogunk valamit kiemelném hogy itt is fontos a megfelelő dolgok telepítése hisz ezek nélkül megint rossz eredményhez juthatunk . Kiemelném a julia halmazt hisz ez egy a Mandelbrot halmazhoz hasonló halmaz azzal a kivetellel hogy ami a mandelbrót állandó az a Julia halmazban inkább változó . Tehat itt is egy számolás fog következni az előző példához hasonlóan itt is rácspontokat fog a program számunkra számolni . Kisebb-Nagyobb eltérésekkel s at fogjuk adni illetve ki fogjuk számolni az adatokat illetve fel fogjuk tolni a racspontokat . Miutan feltölttük a racspontokat és megadtuk a változókat illetve a képünk méretét itt is egy PNG formátumú alkotással lehetünk boldogabbak .

```
Program T100
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
```



```
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left-footer=" ↔
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↔
// color
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↔
// Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

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

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

}
else
{
 std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↵
 d reC imC R" << std::endl;
 return -1;
}

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

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

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

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

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

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

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

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

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

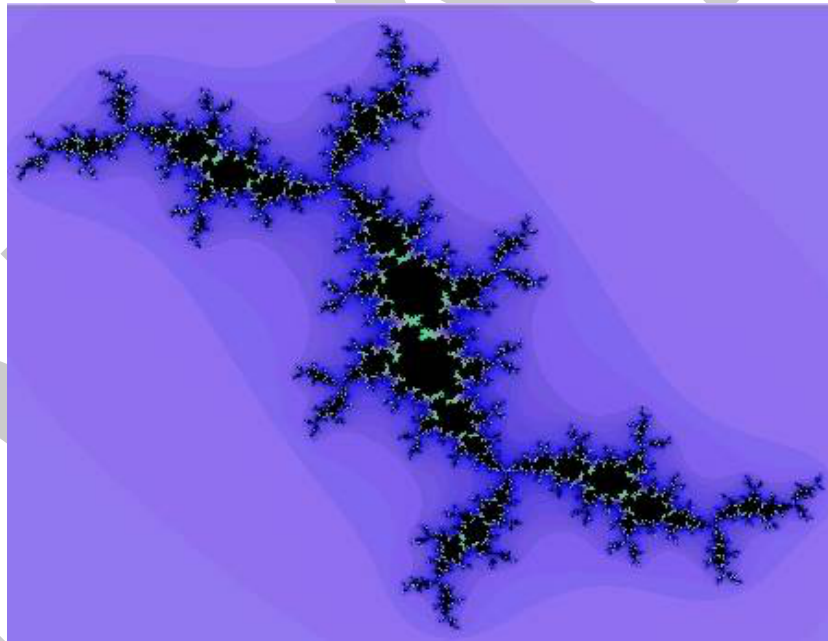
```
 iteracio = i;
 break;
 }
}

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

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

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

}
```



5.2. ábra. Julia halmaz Forrás :[www.t-es-t.h](http://www.t-es-t.h)

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás Forrása : <https://github.com/Samate99/Hello-world/blob/master/Prog1/cuda>

Hát ehhez a feladathoz szükségünk lesz egy megfelelő grafikus kártyához amit az Nvidia-kell legyártani ellenkező esetben nem hiszem hogy képesek leszünk elvégezni a műveleteket. Ebben az esetben is a Mandelbrot halmazt fogjuk számolni viszont a egy elég érdekes számítási platformon, Tehát ahhoz hogy a program lefusson újabb Generációs NVIDIA kártyára lesz szükségünk amely képes a CUDA rendszer kezeléséhez. Ahhoz hogy ez működjön még ilyen körülmények között érdemes ellátogatni az NVIDIA oldalára ahol rengeteg új információval és egyéb szoftverrel lehetünk gazdagabbak illetve szerintem érdemes 1-2 egyéb oldalról is származó cikket elolvasni a CUDA lehetőségei kapcsán. Maga a Cuda egy elég érdekes az NVIDIA által létrehozott platform. Nepszeruseget annal köszönheti hogy ezzel nagyon nagy mértékben tudjuk kihasználni a videokártyák erőforrásait. Tehát a grafikus kártyák legtöbb magját szálal kiváló százalékban leszünk képesek kihasználni. Fontos kiemelni hogy ez a program hasonló elven működik. Itt is feltöltjük a rácspontokat a megfelelő számításokkal illetve itt is kirajzoltatunk egy képet a számítógépünkre

```
Program T100
```

```
// mandelpngc_60x60_100.cu
// Copyright (C) 2019
// Norbert Bátfaí, 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 Páternosztter/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↔
// _01_paruhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUUs
//

#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 éppen a j. sor k. oszlopában 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 elértük, akkor úgy vesszük,
 // hogy a kiindulási 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 << "Hasznalat: ./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 -
```

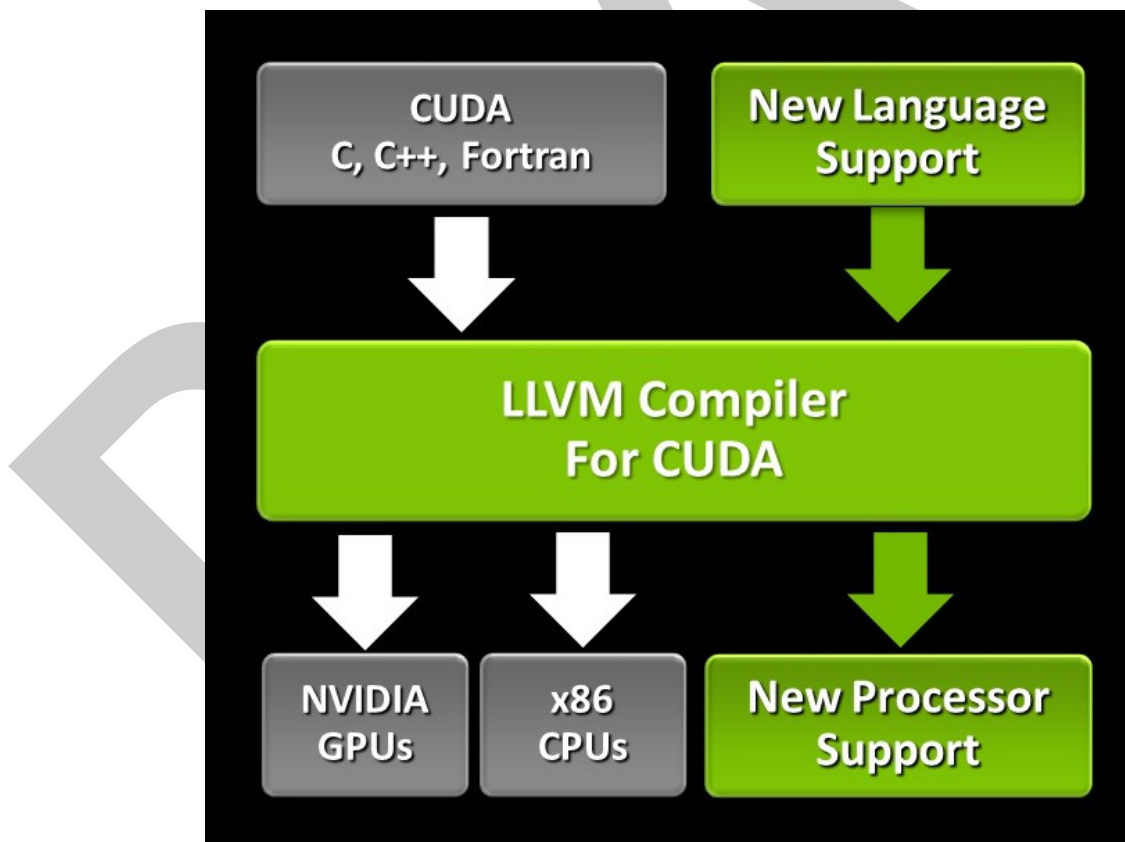
```
(255 * kepadat[j][k]) / ITER_HAT,
255 -
(255 * kepadat[j][k]) / ITER_HAT,
255 -
(255 * kepadat[j][k]) / ITER_HAT));
}
}
kep.write (argv[1]);

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;

}

}
```



5.3. ábra. CUDA Forrás: NVIDIA Developer



## 5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/Qt/Frak/>

Megoldás videó:

A feladat megfelelő megoldásához szükségünk lesz több csomag telepítésére többek között a libqt4-dev csomagra is. Ahogy már ismerjek ezt folyamatot a qmake paranccsal létre kell hoznunk a makefilet és mint ahogy eddig is tettük a Makefileos feladatok esetében le kell futtatnunk. Ha a telepítések és a futtatások sikeresek voltak 4 abalkot fogunk kapni. Mind a 4 képen ugyanaz tekinthető meg. Az egyik képen az eredeti képet láthatjuk a maradék 3 képen pedig erről a képről bezoomolt kepeket tekinthetünk meg. Kiemelném hogy a program nem egyetlen egy fájlból hanem összesen 6 fájlból áll. Mindegyik fajlnak fontos szerepe van van amelyik a mandelbrot halmaz kirajzolásában van szerepe van emlyik van amelyik csak osztályokat tartalmaz viszont a megfelelő működéshez az összesre van szükségünk

## 5.6. Mandelbrot nagyító és utazó Java nyelven

<https://github.com/Samate99/Hello-world/blob/master/Prog1/nagyito.java>

Az előző feladathoz nagyon hasonló feladatot kaptunk. Az előzőhöz hasonlóan ez is ki fog számunkra rajzolni egy képet. Illetve ez a kép is egy Mandelbrot halmaz lesz. Ebben az esetben is fontos a megfelelő fajok, csomagok telepítése mert a program megfelelő futásához elengedhetetlenek. Magát a halmazt szinte ugyanugy hozza létre mint a c++ esetében az eltérő változás a Zoomban tekinthető hisz ebben az esetben ez interaktív lesz. Az eger segítségével tudunk kijelölni részeket majd ezekből a részekből kapunk új képet. Az előzőhöz hasonlóan itt is egy pillanatkeppel lehetünk gazdagabbak, de az előzővel ellentétben nem arról van szó hogy előre megadott kepeket kapunk hanem mi választhatjuk ki hogy a mandelbrot halmaz éppen melyik szegletére vagyunk kíváncsiak.

Program T100

```
* 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 síkon
 * [a,b]x[c,d] tartománya felett kiszámoló és nygítani 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 ű osztály konstruktorának hívása
 super(a, b, c, d, szélesség, iterációsHatár);

 setTitle("A Mandelbrot halmaz nagyításai");

 // Egér kattintó elemények 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();

 }
 });
}
```

```
// Vonzolva 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:

addMouseListener(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 bele vesszük, 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);

}

}
}
```

## 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 eltérő kiszámolt szám.

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/polargen.java>

Tehát első ránézésre a polártranszformációs algoritmus talán ijesztőnek tűnhet viszont ha jobban beleolvasunk a programkódban akkor azért megnyugodhatunk. A polártranszformációval fogunk random számokat generalni amelyeket fel fogunk használni a program különböző részein ! A program több nyelven is megtalálható jelen esetben én egy java nyelven írt kódot mutatok be . Szóval kezdjük is el nézni a programunkat szerintem ami egyből feltűnik a kódot olvasók számára hogy rendelkezünk egy PolarGen Classsal aminek talán az a specifikus tulajdonsága hogy 2 részből áll Beszélhetünk egy Privát illetve egy publikus részről is . De akkor lessük is meg hogy hogyan dolgozik a program . A program megnezi hogy van-e tárolt érték ezután 2 felé válik a történet ha nem rendelkezünk akkor számunk megfelelő mennyiségű értéket . Az egyiket visszaadjuk a programunknak a másikra még később szükségünk lesz emiatt azt eltaroljuk. Tehát utána végzünk pár műveletet és egy ciklus segítségével kiíratjuk az adott számú értéket , eredményt .

Program T100

```
import java.util.Random;
import java.io.*;
import java.lang.Math;
public class PolarGen {

 public final static int RAND_MAX = 32767;
 private static boolean bExists;
 private double dValue;
 static Random cRandomGenerator = new Random();

 public PolarGen() {
 bExists = false;
 }
}
```

```
cRandomGenerator.setSeed(20);
};

public double PolarGet() {
 if (!bExists)
 {
 double u1, u2, v1, v2, w;
 do{
 u1 = cRandomGenerator.nextInt (RAND_MAX) / (RAND_MAX + 1.0); //innent ←
 ől jön az algoritmus
 u2 = cRandomGenerator.nextInt (RAND_MAX) / (RAND_MAX + 1.0);
 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);

 dValue = r * v2;
 bExists = !bExists;

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

 {
 bExists = !bExists; //ha van korábbi random érték, akkor azt adja ←
 vissza
 return dValue;
 }
};

public static void main(String args[]) {
 PolarGen cPolarGen = new PolarGen();
 double dEredmeny = cPolarGen.PolarGet();
 System.out.println(dEredmeny);
}
}
```

## 6.2. LZW

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

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/lzw.c>

Mi is az a binfa . Eddigi Egyetemi pályafutásom során talán a "legrémisztőbb" dolog . Még a bevprog idején

találkoztunk elsőnek hisz a bevprog védés nagyrésztében rajta kellett kisebb változtatásokat létrehozni. . Talan feladata elsőre fel sem tűnt de gyakorlatilag ez egy elég jól működő veszteségmentes tomorítási algoritmus . Nagyon meglepődtem mikor kiderült hogy ezt az algoritmust nagyon széles körben használják szerte az informatika különböző "ágazataiban" . Maga a binfa Terry Welchhez köthető ő volt aki egy előző úgynevezett LZ78as algoritmus továbbfejlesztéseként publikálta . Tehat mint fentebb említettem ez egy tomorítási eljárás amelynek során a kódoló csak a szóbeli idexet kuldi ami egy nagyon ritka és érdekes dolog . Kiemelném hogy ebben az esetben a folyamat dinamikus A binfa megfelelő elkészítéséhez és futatásához érdemes különböző könyvtárakat includeolni . A futtatáshoz szükség lesz egy befilera illetve egy kifilera .

## 6.3. Fabejárás

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

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Fabe.cpp>

A binaris fák bejarasaiban általában 3 módot v -in a -pre és a posztorder különböztetünk meg . A 3 mód között igazából felépítésbeli eltérések találhatók Tehat gyakorlatilag a gyökér és ágak elhelyezkedésétől függ hogy éppen milyen módról beszélhetünk. Kiemelném hogy a program lefutásában illetve "viselkedésében" lényeges változás nem fedezhető fel. Inorder esetben a gyökér középen van az ágai pedig felette illetve alatta Preordernél a gyökér felül van az ágai alatta postordernél a gyökér alul van ágai pedig felette.

Program T100

```
/inorder
void kiir (Csomopont * elem, std::ostream & os)
{ if (elem != NULL)
{
 ++melyseg;
 for (int i = 0; i < melyseg; ++i)
 os << "---";
 kiir (elem->nullasGyermekek (), os);
 os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
 kiir (elem->egyenesGyermekek (), os);
 --melyseg; }
}
o
//preorder
void kiir (Csomopont * elem, std::ostream & os)
{ if (elem != NULL)
{
 ++melyseg;
 for (int i = 0; i < melyseg; ++i)
 os << "---";
 os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
 kiir (elem->nullasGyermekek (), os);
```



```
kiir (elem->egyGyermek (), os);
--melyseg; } }

//postorder
void kiir (Csomopont * elem, std::ostream & os)
{ if (elem != NULL)
{
 ++melyseg;
 for (int i = 0; i < melyseg; ++i) os
 << "---";
 kiir (elem->nullasGyermek (), os);
 kiir (elem->egyGyermek (), os);
 os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::
 endl;
 --melyseg; }
}
```

## 6.4. Tag a gyökér

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

Megoldás forrása: [//github.com/Samate99/Hello-world/blob/master/Prog1/lwzbinfa.cpp](https://github.com/Samate99/Hello-world/blob/master/Prog1/lwzbinfa.cpp)

Na tehát ebben a részben a feladatunk nagyon hasonló az előzőkhöz. Lényegében a már fentebb említett C nyelven megírt binfa egy C++ változatáról beszélhetünk. A program mivel ugyanaz csak egy más programozói nyelvben lett leírva ugyanazt csinálja mint a C nyelven írt binfa ugyanúgy a veszteségmentes tomorításban van fontos szerepe. Erdemes megneznünk magat a kód felépítését. Ahogy láthatjuk a programnak 2 része van rendelkezünk egy privat illetve egy public résszel is. A fában átalakításokra is lesz szükség ebben az esetben a gyökér csomópont változáson megy át. Megfigyelhetjük a csomópontok épülését a fában. A programban továbbá láthatjuk a tagfüggvény túlterhelését. Utána megnezzük hogy az esetben éppen mivel kell dolgoznunk kétféle lehetoseg van vagy 0-as vagy 1-eszel gyakorlatilag mindket részben ugyanaz zajlik le. Nezzük meg az 1-eszel szemleltetve. Megvizsgáljuk hogy van 1 es ha van akkor arra haladunk tovább ha nincs akkor létrehozunk egy egyest. A C++ ba áthelyezett program esetén is fontos a befűle illetve a kifűle használata futtatáskor.

## 6.5. Mutató a gyökér

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

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/lwzbinfa.cpp>

Tehát a feladatunkban továbbra is a binfat kell jobban tanulmányoznunk illetve a binfaban kell különböző módosításokat létrehozunk. A feladatunk hogy a gyökér ne kompozícióban legyen hanem aggregációban

. Ez a gyakorlatban azt jelenti hogy a fa egy részét át kell dolgoznunk illetve hogy megfelelő kapcsolat legyen azaz aggregációs érdemes mutatókat használni . Viszont ha mutatókat használunk az több hibát is előhozhatunk kezdhünk azzal hogy gyakorlatilag más szintaktikai elemeket kell alkalmaznunk illetve átjavítanunk hogy lefusson a program . A legelső lépés az lenne hogy a gyokeret a megfelelő szabályok szerint atírjuk pointerre .

## 6.6. Mozgató szemantika

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

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/z3a9.cpp>

A binfa témakör utolsó állomására értünk ebben a peldaban a feladatunk a mozgató szemantika viselkedésének megvizsgálása lesz. A mozgató szemantika esetében arról beszélhetünk hogy létre kell hoznunk egy Operatort Az ő feladata lesz hogy a program lefutása során lemásolja magát . Természetesen egy adott metódus szerint történnek az események. A ujjepules iranyat a rekurzív függvény határozza meg . A programban az 1es és a 0as gyermekek vizsgálatát ő fogja elvégezni . Ennek segítségével donti el a program hogy merre haladjon . A program egyszeruen uj csomopontokat hoz létre es lemasolja magat . Tehat maga az std::move nem vegez mozgatast

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyaszimulációk

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

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

Megoldás forrása: <https://github.com/Samate99/Hello-world/tree/master/Prog1/Myrmecologist>

Ez a szimuláció egy nagyon érdekes művet . Vegrehatjasahoz viszont több egyéb csomag , program telepítésére van szükség ilyen például a libqt4 csomag amelyet telepítenünk kell . Miután a telepítések lezajlottak kiemelném a qmake használatát , ennek segítségével fogjuk megkapni a makefilet ami a program futtatásához elengedhetetlen. Nézzük is meg a kódot ahogy láthatjuk a main függvényen belül több helyen adatokat kell megadnunk illetve deklarálnunk kell . Láthatjuk hogy a program legalbb 3 elkülöníthető osztállyal /resszel rendelkezik . Ant , Antwim és Anthread lesz ez a 3 osztály. Nézzük is meg őket egyesével nézzük meg hogy mi miért felelős . Tehát az Antwim lesz az osztály amelynek feladata a megjelenítés ebben tudjuk megadni az ablak tulajdonságait , méreteit , szélességét pixeleinek számát stb.... Az Ant maga a virtualis "hangyak" tulajdonságait fogja tartalmazni . Tehát ebben a részben szerepelnek a hangyaink mozgását , koordinátasat rögzítő beállítások . Itt láthatjuk a Hangyak méreteit , illetve mozgásuk előre meghatározott irányát . Vegül nezzük meg az Anthreadet hogy mit is tartalmaz . Az Anthread elég sok dolgot tartalmaz . Így első körben kiemelném hogy tartalmazza a szimulációban résztvevő hangyak számát, illetve tartalmazza a hangyak utvonalaikat , továbbá ebben a részben adhatjuk meg a feromont a feromon párolgásának mértékét és egyéb feromon adatokat amelynek segítségével képesek leszünk a hangyak "irányítására" Kiemelném hogy a programban több előre definiált bind is található ilyen például a P amely feltételezéseim szerint az angol PAUSE szóból eredhet . Lenyomásával természetesen meg tudjuk állítani a folyamatot szüneteltetni tudjuk. A következő ilyen a Q lesz ez az angol QUIT szóból ered gondolom . Ennek lenyomásával kitudunk lépni a folyamatból.

### 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/blob/master/Prog1/Sejtautomata.java>

A feladatunk az Élet jatekanak bemutatása .Maga az élet jateka John Hortony Conway kezei között született meg . Ő alkotta meg eme csodás világnak egyszerű szabályait . Tehat hogy is épül fel ez a világ nezzük is meg . Ha jobban megnezzük a kódot észrevesszük hogy sejteket tartalmaznak szerintem mindenki számára feltűnő lehet hogy nem egyszerű sejtekről van szó . A sejtek tulajdonságokkal rendelkeznek . Számszerint kettővel Beszélhetünk élő és halott sejtekről . Itt jönnek képben a fentebb már említett uriember nevéhez fűződő szabályok . Tehat egy sejt akkor gondolok , vélünk élő sejtnek ha van adott számú szomszédja ami esetünkben 2 szomszédnak kell lennie ahhoz hogy a sejt életben maradjon Ha a sejtünk nem rendelkezik szomszédokkal akkor "elpusztul".De nem örökre Ha valamilyen módon később újra szomszédokat fog kapni a sejtünk újraéled . Tehat a képernyőn pixeleket fogunk feldezeni . A 2 sarokból indulnak el a folyamatok és átlósan töltik meg a képet. A program futása során random sejt csoportok "agyú-golyók" fognak életben maradni illetve meghalni , végül ezekből kerülnek ki a képeken látható alakzatok , mintak Fontos kiemelni hogy ebben a programban is több Bindről beszélhetünk Ilyen például az S az N vagy a G, Az S lenyomásával egy pillanatképpel lehetünk gazdagabbak az eseményről . Az N lenyomásával tudjuk növelni a sejtek méretét végül pedig az I-G gombok segítségével fogunk tudni a folyamat lefolyásának gyorsaságára hatni.

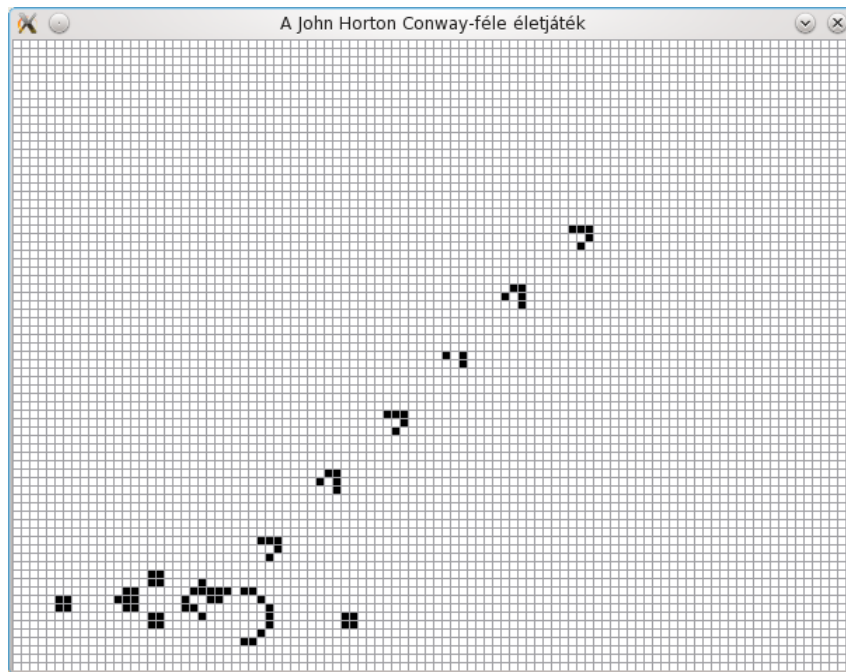
## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/Samate99/Hello-world/tree/master/Prog1/sejtautomata>

Ez a példa szinte teljesen ugyanaz legalábbis a feladat . Egyetlen változást az jelenti hogy ebben az esetben egy másik nyelvben kell dolgoznunk . A mi esetünkben ez a nyelv a nem a java hanem a c++ lesz . A program megfelelő működéséhez , lefutásához , lefordításához különböző szoftverek előtelepítésére . Itt is alkalmaznunk kell a qmake parancsot amellyel le tudjuk generálni az itt is elég fontos makefilet. Ettől kezdve a program ugyanúgy működik mint a java kód . Tehat ugyanúgy megvannak sejtjeink ugyanúgy elindulnak a pixeleken ugyanúgy elhallnak illetve felélednek és így hozzák létre a dolgokat .



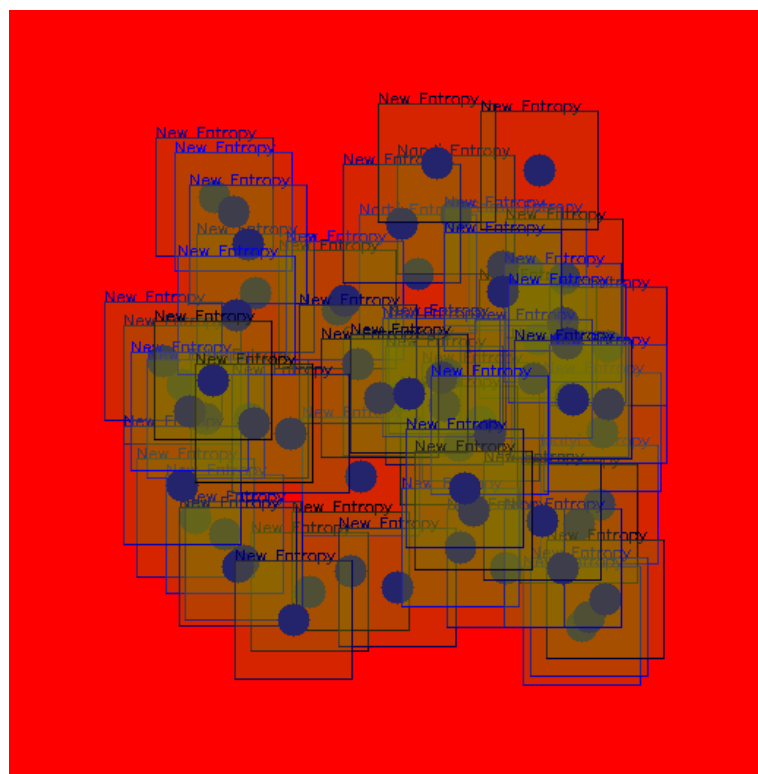
7.1. ábra. John Horton Conway-féle életjáték Forrás : Batfai Norbert

## 7.4. BrainB Benchmark

Megoldás videó:

<https://github.com/Samate99/Hello-world/tree/master/Prog1/brainB>

A BrainB Benchmark talán számomra a legerdekesebb program nagyon szeretem az esportot jelenleg is erősen csogozok és próbálok benne eredményeket elérni. Habár ez a program nem feltétlenül a cs-eknek szól inkább a lolosok, dotások számára lehet hasznos. Tudjuk hogy ez a program a tehetséges esport játékosok felfedezésére illetve a jelenleg is aktív játékosok tesztelésére felmérése lehet hasznos. Megmondom őszintén nagyon érdekelné hogy egy Profi mezőnybe játszó esportoló milyen eredményeket lenne képes elérni a játékban. A programban az eger segítségével kell egy bizonyos pontban tartani az egeret miközben egyéb tényezők jelen esetben negyzetek határoltatnak. Ha jól haladunk a játék szerint akkor egyre gyorsabb lesz illetve a negyzetek száma megnő ha elveszítjük "karakterünket" akkor lassul a mozgás. A program megfelelő használatához telepítenünk kell egyéb programokat például a OpenCV-t illetve a libqt-t. Ha ezt telepítettük már csak el kell indítani a programot és elkezdhetjük tesztelni a képességeinket és akár statisztikákkal is alátámaszthatjuk a bennünk rejlő potenciált.



7.2. ábra. BrainB Benchmark : Batfai Norbert

## 8. fejezet

# Helló, Schwarzenegger!

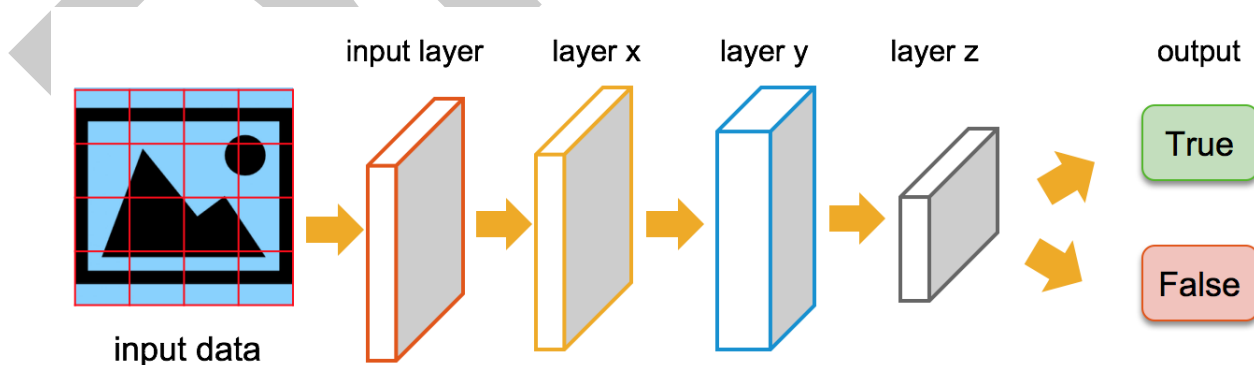
### 8.1. Szoftmax Py MNIST

aa Python

Megoldás videó: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0>

Megoldás forrása:

Ez a program szerintem a könyv egyik legérdekesebb programja működéséhez 2 elengedhetetlen dolog telepítésére van szükség egyfelől szükség lesz egy adatbázisra amit a tensorflowal érhetünk el illetve egy fejlesztői környezetre ami a mi esetünkben a Python lesz . Talan ez az a nyelv amelyet eddigi tanulmányaim során a legkevésbé foglalkoztam de szobatársamnak hála szerencsére ezzel is sikerült találkoznom . Miután telepítettük ezt a 2 dolgot kapunk egy adatbázist , Egy adatbázist ami rengeteg képet tartalmaz .Itt tehetjük fel a kérdést hogy mire is fogjuk használni ezt a rengeteg képet. Ezekkel a képekkel fogjuk megtanítani elemezni a programunkat . A program az adatbázisból rengeteg képpel találkozik , ezzel megtanulja felismerni a kepeket . Rengeteg képpel fogjuk megtanítani 1-1 dologra és további képekkel fogjuk ellenőrizni hogy sikerült e a megfelelő "oktatás" . Ha mindent jól csináltunk és sikerült a dolog akkor ha az előző képekhez hasonló . Tehat ugyanazt a dolgot ábrázoló képet fogunk mutatni a programunk számára a program képes lesz arra hogy felismerje es a megfelelő kategóriához sorolja a képet .



8.1. ábra. TensorFlow Forrás:SAP Blogs

## 8.2. Szoftmax R MNIST (Passz)

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.3. Minecraft-MALMÖT(Passz)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT



## 9. fejezet

# Helló, Chaitin!

### 9.1. Iteratív és rekurzív faktoriális Lisp-ben(Passz)

Megoldás videó:

Megoldás forrása:

### 9.2. Gimp Scheme Script-fu: króm effekt

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

Megoldás videó: [https://youtu.be/OKdAkI\\_c7Sc](https://youtu.be/OKdAkI_c7Sc)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Chrome](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome)

A program megfelelő működéséhez érdemes a GIMP megfelelő változatát használni. Jelen esetben én a Gimp 2.10-es verzióját használtam. Tehát mit is kell csinálnunk... Azt kell tennünk hogy a programunkban meg kell adnunk az alap feltételeket tehát a betűtípust a szöveget a kép méretét a színt és sok egyéb információt. Tudni kell hogy maga a GIMP az egyik legnépszerűbb szoftver illetve hogy nagyon felhasználó barát. Rendkívül jól kezeli a scripteket, külön mappa is van rá hogy kifejezetten ide rakjuk be őket. Tehát ez a feladatunk most is. Gyakorlatilag a programunk forráskódját be kell másolnunk a program forráskódjába a GIMP "Scripts" mappájába. Ezután már csak pár kattintásra van szükségünk hogy elkészítsük a képet. Gyakorlatilag a program úgy működik hogy a program forráskódját be kell másolnunk a Gimp "scripts" mappájába. A programunkban meg kell adnunk a szöveget, a szöveg betűtípust, a szöveg/kép méretet, a színt a szöveg színet és egyéb információkat.

```
; 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://penguinpetes.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"
)
```

```
}
```

### 9.3. Gimp Scheme Script-fu: név mandala

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

Megoldás videó: [https://bhaxor.blog.hu/2019/01/10/a\\_gimp\\_lisp\\_hackelese\\_a\\_scheme\\_programozasi\\_nyelv](https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Mandala](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala)

Tehat az előző feladathoz hasonlóan újra a GIMPel kell alkotnunk jelen helyzetben egy mandalát fogunk készíteni. Szerintem elsőnek fontos tisztázni a mandala jelentését a mandala egy kép amelyekkel a hindu és buddhista vallásban világszemléletben ábrázolnak dolgokat értékek ezalatt isteneket vagy barmifele számukra szent vagy fontos dolgot. A mostani esetben is fontos a GIMP megfelelő verzióját használni a biztos lefutás érdekében. A GIMP szerencsénkre nagyon könnyedén használja a scripteket, saját mappával rendelkezik hogy még könnyebben tudjunk dolgozni vele. Tehat a scriptünket az előző feladathoz hasonlóan itt is bele kell raknunk a a GIMP "Scripts" mappájába Mivel a GIMP egy csodás program és a scripteket könnyen és gördülékenyen kezeli nincs is más dolgunk csak a scripts mappába be kell más Az elozo feladathoz hasonlóan itt is meg kell adnunk még a programban az adatokat. Tehat meg kell adnunk a Szöveget ,a szöveg méretét méretet ,a kép azaz a mandala méretét , a szöveg betűtípusát , szint a szinkonbinációt , es egyéb adatokat. Ha ezt megfelelően adtuk meg es minden klappol akkor a program futása után egy saját mandalával lehetünk gazdagabbak.

```
; 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_hackelese_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"
)
}
```

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Programozási alapfogalmak

[?]

Tehat eljutottunk a könyvünk utolsó részéhez . Ebben a részben a feladatunk egy olvasónapló kidolgozása lesz. Az olvasó napló 3 könyvből fog összetevődni. Az első könyv a méltán híres Juhasz Istvan kezei közül kikerült alkotás lesz amely a Magas Szintű programozási nyelvek 1 címet viseli. Kiemelném Juhasz Istvan munkásságát hisz meghatározó alakja volt a Debreceni Egyetemi informatika oktatásban illetve A diákok a mai napig emlegetik feltöltött jegyzeteit használják , hasznosítják ami nem sok mamár nem tanító oktatóról mondható el véleményem szerint. A második alkotás Benedek Zoltán: Szoftverfejlesztés C++ nyelven című könyv lesz részemről ez is egy igen érdekes olvasmány volt hasonlóan a harmadik kincshez ami Brian W.Kernighan és Dennis M. Ritchie . The C programming language nevet viseli. Tehat akkor kezdjük is feldolgozni a könyveket egyesével . Tehat az első a Juhasz Istvan féle könyv ez egy oldalas alkotás . Az alkotásban több részt különböztethetünk meg . A mu első felében megismerkedhetünk az alapfogalmakkal . A programozási nyelvek 3 szintjevel a gépi nyelvel az assembly nyelvel és az általunk ismert és hasznalt magas szintu programozási nyelvekkel . Tehat nezzük is meg miben különböznek ezek a nyelvek mire jók mire használjuk vagy éppen nem használjuk oket. A gépi nyelv az a processzor számára érthető nyelv . Minden processzor gépi nyelvel rendelkezik . Ez egy rendkívül bonyolult illetve összetett nyelv . Emiatt nem is tanuljuk . Viszont rendkívül sokat foglalkozunk vele hisz az összes Magas szintu nyelven megírt program vagy úgynevezett forrásprogram erre a nyelvre kerül lefordításra . Tehat amikor mi lefordítunk egy programot abban az esetben az általunk leírt magas szintu nyelv atkerül gépi nyelvé . Es így képes lefutni . A szövegben olvashatunk a fordításról tudjuk hogy ez egy nagyon fontos feladat hisz ez kódolja át a másik nyelvre. Fontos kiemelni hogy a program végignézi hogy mit használtunk es hibákat keres a kódolásunkban ez azért fontos mert így könnyedén megkapjuk a hibákat , kiemelném hogy a program egy apró hiba esetén is visszadobja a kódot tehát ha csak egy kis hiba van akkor se fordul le. Ez azért van mert a program egészét nézi nem pedig sorokra vagy részekre bontva fordítja a programot. Az alkotásban továbbá olvashatunk a különböző magasszintu programozási nyelvek különböző szabályairól illetve alap elgondolásaikról. Folytatásként akkor nezzük meg a programozási nyelvek alapelemeit .Ki nem találná mi is a legkisebb építő elem egy programban . Ez az építő elem nem más mint a karakterek . Mik is azok a karakterek. A karaktereként elnevezve elég sok minden tartunk számon kezdve a betűkkel es számokkal úgynevezett speciális karakterekkel. Speciális karakter néven értem a vesszőt nyílakat különböző zárójeleket . Betűket is kiemelném hisz rengeteg betűről beszélhetünk hisz bármely nyelv betűit használhatjuk a kódunkban tehát karakternek minősülnek a kínai írás elemei a ciril betűk és természetesen az általunk ismert betűk . Fontos kiemelni

nem minden fordító és programnyelv képes minden karaktert megfelelően kezelni több program is megküzd az ékezetekkel és egyéb karakterekkel . Kivételt élveznek ezalól a kommentbe írt dolgok mivel ezek az kód olvasójához szólnak , nincs közül a fordításhoz nem fordítja la őket a program . Fontos hogy vannak olyan szovegek karakterek szavak amelyeknek az éppen írt nyelvben rendelkeznek különböző tulajdonsággal. Tehat előre megadott dolgokat hívnak életbe ilyen például az IF a for a while vagy akár a case. elég sok eszközt használunk és többet is kiemelnék illetve kiemel a könyv . Ilyenek például a konstansok amelyek segítségével egy fix értéket rakhatuk a kódba vagy ilyenek a nevesített konstansok amelyek segítségével ugyanugy értékeket tudunk megadni . Ezt a megadási módot deklarálásnak nevezzük itt érdemes szót ejteni az adattípusokról . Elegendő sok adattípus van amelyeket a megfelelő módon kell használnunk beszélhetünk int double char vagy akár a tombi is , és egyéb típusokról . Ezek előre megszabályozzák hogy mire fogjuk használni az adatot . Ezeken felül a könyv szerint egy program írása közben rengeteg dologgal találkozhatunk .Találkozhatunk úgynevezett kifejezésekkel ilyenek például az operandusok operatorok Ezekhez a részhez tartoznak a zárójelek egy része amelyeket rengetegszer használunk egy program írásánál. A következő nagy rész az utasítások része lesz . Ebben a részben találkozhatunk az utasítások 2 csoportjával a deklarációval és a végrehajtásos utasítással. Nos akkor mi is okozta különbség okozott a 2 utasítási rendszer között . Az első esetében a fordításkor használt fordítóprogrammal való szerepe miatt fontos. Ez a fajta utasítás befolyásolja a fordítóprogram fordítását. A második viszont a kódolás során nyilvánul meg ezekkel az utasításokkal már mindannyian találkoztunk ilyenek például az értéadó utasítások és ciklusszervező , az egyéb utasítások.

## 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

A következő számunkra fontos könyv A Brian W.Kernighan és Dennis M. Ritchie . The C programming language nevet viseli.Ebben a könyvben a C nyelv sajátosságaival fogunk megismerkedni.Így kezdésként fontos kiemelni hogy a C nyelv maga rendkívül sok speciális tulajdonsággal rendelkezik nezzük meg akart a vesszőt ha egy sort vagy több sort egy vessző követ akkor általában annak elválasztási funkciót adunk véleményem szerint. A C nyelvben való kódoláskor nem érdemes ilyet használni mivel könnyen lefordulási problémákkal találkozhatunk . Mivel ha egy sort vesszővel zárunk az itt utasításnak minősül . Ebben a nyelvben kifejezzen fontos a kapcsos zárójelek megfelelő használata mert ez is könnyen hibához vezethet. Továbbá fontos kiemelni hogy több dologban megegyezik a többi nyelvvel ugyanugy használhatunk megfelelő ciklusokat ugyanugy Ifel tudunk választ kérni / adni ugyanugy tudunk Switchet alkalmazni hogy elágazásokat vezessünk be. A könyvben külön szóba kerülnek a ciklusok illetve a ciklusok alkalmazása . Hisz tudjuk hogy a különböző ciklusok között (while, do-while , for) akkora különbség nincs tehát ugyanugy feltételt kell megadni. Talán ami fontosabbnak tekinthető hogy van olyan fajta ciklus amely 1x mindenfelepképpen lefut illetve van olyan amelynek ha nincs megfelelő feltétel egyszer sem fut le. A C++ ban már megtanult cout helyett Cben a printf függvényt kell használnunk hogy a megfelelő dolgokat kiirassuk az outputra. Így összességében a C nyelv egy nagyon érdekes nyelv amely talán az egyik nehezebb de véleményem szerint ez ízlések és pofonok hogy éppen ki miben szeret és akar vagy éppen miben köteles programozni.

## 10.3. Programozás

### [BMECPP]

A harmadik és egyben utolsó könyv a Benedek Zoltán: Szoftverfejlesztés C++ nyelven címet viseli. Az alkotás fő témája a C++ nyelv, amely egy nagyon fontos nyelv, és nőtte ki magát napjainkra. Tehát a Benedek Zoltán Főle "BME-S" Könyvben rengeteg C++ információval lehetünk gazdagabbak. A könyvben megismerhetjük a Függvényparaméterek megfelelő működését, megismerhetjük, hogy mikre eppen mit ad vissza. Megismerhetjük, hogy a C++ nyelvben a függvényeket mi azonosítja, tehát megtudjuk, hogy ez a dolog a nevük és az argumentumlistájuk. Tehát a C kóddal ellentétben C++-ban lehet 2 függvény azonos névvel, ha az argumentumlistájuk különbözik. Megismerkedhetünk a C++ memóriakezelési képességeivel. Azaz, hogy ebben az esetben ezt a feladatot Operátorok látják el, C-vel ellentétben, ahol függvény végzik ugyanezt a dolgot. A C++ rengeteg kiváló tulajdonságát láthatjuk meg a könyvben, ilyen például a hibakezelése, ami sokkal átláthatóbb és talán könnyebb is, mint a C-é.

## **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(Passz)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 11.2. Java osztályok a Pi-ben(Passz)

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## **IV. rész**

### **Irodalomjegyzék**

DRAFT



## 11.3. Általános

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

## 11.4. C

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

## 11.5. C++

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

## 11.6. Lisp

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

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

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