

Univerzális programozás

Így neveld a programozód!

Ed. BHAX, DEBRECEN,
2019. február 19, v. 0.0.4

Copyright © 2019 Kiss Máté

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert Ács Kiss, Máté	2019. október 2.	

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	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2019-02-27	Helló, Turing! csomag kész.	kissmate
0.0.6	2019-03-13	Helló, Chomsky! csomag kész.	kissmate
0.0.7	2019-03-20	Helló, Caesar! csomag kész.	kissmate
0.0.8	2019-03-27	Helló, Mandelbrot! csomag kész.	kissmate

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-04-03	Helló, Welch! csomag kész.	kissmate
0.1.0	2019-04-10	Helló, Conway! csomag kész.	kissmate
0.1.1	2019-04-24	Helló, Schwarzenegger! csomag kész.	kissmate
0.1.2	2019-05-01	Helló, Chaitin! csomag kész.	kissmate

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	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	7
2.3. Változók értékének felcserélése	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	13
2.6. Helló, Google!	13
2.7. 100 éves a Brun tétel	15
2.8. A Monty Hall probléma	16
3. Helló, Chomsky!	19
3.1. Decimálisból unárisba átváltó Turing gép	19
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	20
3.3. Hivatkozási nyelv	21
3.4. Saját lexikális elemző	22
3.5. l33t.1	23
3.6. A források olvasása	23
3.7. Logikus	25
3.8. Deklaráció	25

4. Helló, Caesar!	28
4.1. int *** háromszögmátrix	28
4.2. C EXOR titkosító	29
4.3. Java EXOR titkosító	31
4.4. C EXOR törő	32
4.5. Neurális OR, AND és EXOR kapu	34
4.6. Hiba-visszaterjesztéses perceptron	37
5. Helló, Mandelbrot!	39
5.1. A Mandelbrot halmaz	39
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	41
5.3. Biomorfok	43
5.4. A Mandelbrot halmaz CUDA megvalósítása	46
5.5. Mandelbrot nagyító és utazó C++ nyelven	49
5.6. Mandelbrot nagyító és utazó Java nyelven	51
6. Helló, Welch!	54
6.1. Első osztályom	54
6.2. LZW	56
6.3. Fabejárás	61
6.4. Tag a gyökér	62
6.5. Mutató a gyökér	64
6.6. Mozgató szemantika	65
7. Helló, Conway!	66
7.1. Hangyaszimulációk	66
7.2. Java életjáték	69
7.3. Qt C++ életjáték	76
7.4. BrainB Benchmark	76
8. Helló, Schwarzenegger!	80
8.1. Szoftmax Py MNIST	80
8.2. Mély MNIST	83
8.3. Minecraft-MALMÖ	83

9. Helló, Chaitin!	84
9.1. Iteratív és rekurzív faktoriális Lisp-ben	84
9.2. Gimp Scheme Script-fu: króm effekt	85
9.3. Gimp Scheme Script-fu: név mandala	85
10. Helló, Gutenberg!	86
10.1. Programozási alapfogalmak	86
10.2. Programozás bevezetés	88
10.3. Programozás	89
III. Második felvonás	91
11. Helló, Berners-Lee!	93
11.1. Bevezetés a mobilprogramozásba.	93
11.2. Java és C++	94
12. Helló, Arroway!	95
12.1. OO szemlélet	95
12.2. Homokozó	97
12.3. Gagyí	98
12.4. Yoda	99
12.5. Kódolás from scratch	101
13. Helló, Liskov!	104
13.1. Liskov helyettesítés sértése	104
13.2. Szülő-gyerek	106
13.3. Anti OO	108
13.4. Hello, Android!	109
13.5. Ciklomatikus komplexitás	111
IV. Irodalomjegyzék	112
13.6. Általános	113
13.7. C	113
13.8. C++	113
13.9. Lisp	113

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk?

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán 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
- C kapcsán a [KERNIGHANRITCHIE] könyv adott részei.
- C++ kapcsán a [BMECPP] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a [The GNU C Reference Manual](https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf), mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [BMECPP] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.
- Kódjátzsma, <https://www.imdb.com/title/tt2084970>, benne a **kódtörő feladat** élménye.

- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.

DRAFT

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

A könyvben szereplő összes forráskód megtalálható csomagonként összegyűjtve az alábbi linken: <https://github.com/kissmate3/Prog1>

A végtelen ciklus egy olyan ciklus, amelyben a feltétel állandóan adott(igaz), ezért a ciklus nem lép ki, hanem újra és újra lefut. például `for(;;)`

Vannak ciklusok amik a szálakat 100%-ban vagy 0%-ban dolgoztatják.

0%-ban dolgoztat:

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

int main() {
while(1) {
    sleep(100);
}
}
```

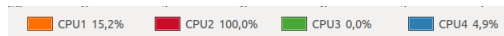
Magyarázat: Az `unistd` header tartalmazza a `sleep()` függvényt. Ezért kell `include`-olni az `stdio.h` header (standart input/output) mellett. Az `int main()` a fő függvényünk. A `while()` pedig a ciklus. A `()` belülre kell írunk a feltételt. Amíg ez igaz, a ciklus újra és újra lefut. A példánkban a ciklusban az 1 szám szerepel. Ez az érték mindig igazat ad vissza, tehát a ciklus állandóan újraindul amíg ki nem löjjük. A `sleep(100)` függvény pedig azért kell, mivel ez altatja a processzor folyamat szálát. A függvényben megadott érték jelenti azt, hogy hány másodpercig altatja a processzort.

CPU1 18,8% CPU2 0,0% CPU3 5,0% CPU4 33,7%

100%-ban dolgoztat egy szálát:

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

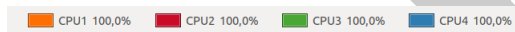

Magyarázat: Az előző példától nem sokban tér el. Az include-k és a ciklus magyarázata megegyezik, az előző példáéval. Itt annyi a különbség, hogy nincs benne a sleep() függvény, azaz a szál nincs altatva. Így a végtelen ciklus 100%-ban dolgoztat 1 szálát.



100%-ban dolgoztat minden szálát:

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

Magyarázat: A programunk, az előzőhöz egy openmp-vel bővült. Ezzel az include-val belépünk, a párhuzamos programozás küszöbére. #pragma omp parallel sor adja azt az utasítást a gépnek, hogy a feladat az összes szálon fusson. (A fordításnál -fopenmp kapcsolóval kell bővítenünk a parancsot.)



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 fogni vagy sem!

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra építő Lefagy2 már nem tartalmaz feltételezett, csak konkrét kódot:

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok: Ha úgy vesszük, hogy a T100 és T1000 létező program és T1000 ben meghívjuk saját magát. A t100 alapján ha a programunkba van végtelen ciklus, akkor igaz értéket ad a Lefagy program a Lefagy2 programnak ,így tehát az is igaz értéket fog adni, viszont ha a Lefagy false értéket ad vissza akkor a Lefagy2 belém egy végtelen ciklusban, tehát a program le fog fagyni. Tehát olyan program mint a T100 nem működik mivel, ha egy olyan program érkezik bele amiben van végtelen ciklus, akkor a program beáll mert a ciklus nem áll meg.

2.3. Változók értékének felcserélése

A feladat két változó értékének felcserélése. Például $a=1$, $b=2$, ebből lesz a megoldás, hogy $a=2$, $b=1$. Napjainkba a számítógép fejlettsége és gyorsasága miatt, már egyszerűen megcsinálhatjuk egy segédváltozóval vagy exort-tal, de régen nagyon sokat számított az erőforrások jó felhasználása, elosztása. Ezért ezek a megoldásoknál sokkal könnyebb volt a számítógépeknek számolni, ha különbséggel vagy szorzással cseréltük fel a változókat. Az utóbbi kettőt nézzük most meg:

Változócsere különbséggel:

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

int main()
{
    int a=1;
    int b=2;

    printf("%s\n%d %d\n", "kulonbseggel:", a, b);

    a=a-b;
    b=a+b;
    a=b-a;
    printf("%d %d\n", a, b);
}
```

Magyarázat: A fejléctet már ismerjük az előző feladatból. A printf() függvény a kiíratáshoz kell majd nekünk. az első argumentum a kiíratás formátuma, a többi pedig a változók kiíratása. A „%d” azt jelenti, hogy egy egész típusú változót fogunk kiíratni, még a „\n” a sortörést jelenti. Maga a feladat egyszerű matematika. Legegyszerűbben a példával lehet megérteni.

$a=1$, $b=2$

$a=1-2=-1$ „a” értéke -1 lesz.

$b=-1+2=1$ „b” értéke 1 lesz, ami az „a” értéke volt.

$a=1-(-1)=2$ az „a” értéke 2, ami a „b” értéke volt

Kész is a cserénk.

Változócsere szorzattal:

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

int main()
{
    int a=1;
    int b=2;

    printf("%s\n%d %d\n", "szorzassal:", a, b);
}
```

```
a=a*b;
b=a/b;
a=a/b;

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

Magyarázat: A megoldás itt annyiban különbözik, hogy nem „+” és „-”, -t használunk hanem „*” és „/” -t.
Példa:

a=1, b=2

a=1*2=2 „a” értéke 2 lesz.

b=-2/2=1 „b” értéke 1 lesz, ami az „a” értéke volt.

a=2/1=2 az „a” értéke 2, ami a „b” értéke volt.

Kész is a cserénk.

2.4. Labdapattogás

```
#include <stdio.h>
#include <urses.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 ) {
        xnov = xnov * -1;
    }
    if ( x<=0 ) {
        xnov = xnov * -1;
    }
    if ( y<=0 ) {
        ynov = ynov * -1;
    }
    if ( y>=my-1 ) {
        ynov = ynov * -1;
    }

}

return 0;
}
```

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

Magyarázat: Az új dolog ami a fejlécnél feltűnik az a curses.h header. Ez képernyő kezelési függvényeket tartalmaz, és a program megjelenítéséhez szükségünk van rá.

A következő részlet:

```
WINDOW *ablak;
ablak = initscr ();
```

Így formázzuk meg a kimenetet. Az initscr () függvény curses módba lépteti a terminált.

A deklarált x és y -on lesz a kezdő értékünk. Az xnov és ynov pedig a lépésközöt mutatja. (lépésenként a koordináta rendszeren xnov, ynov-al való elmozdulást). Az mx és my lesznek a határértékek, hogy a program csak az ablakon belül mozogjon.

A végtelen ciklus következtében, a labda addig pattog, amíg ki nem löjük a programot. A ciklusban az első függvény a getmaxyx () . Ez határozza meg,hogy mekkora az ablakunk mérete. refresh() függvénnyel frissítjük az ablakot. Köztük a mvprintw() függvény az x és y tengelyen megrajolja a „ „ között lévő szöveget, számot vagy karaktert, esetünkben az O-t. Az usleep függvény azt szabályozza mennyi ideig altassa a ciklust még újra indul, azaz milyen gyorsan pattogjon a labda.

```
x = x + xnov;
y = y + ynov;
```

Megnöveljük az értékeket, minden ciklus lefutásnál (mozog a "labda").

A következő négy if-el pedig azt vizsgáljuk, hogy a labda az ablak szélén van e, ha igen akkor -1 -el szorozzuk, ezáltal a labda irányt változtat. A fordításnál -lncurses kapcsolót kell használnunk.

Nézzük ugyan ezt a feladatot "if" nélkül:

Forrás:https://progpatet.blog.hu/2011/02/13/megtalaltam_neo_t

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

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

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

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

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

        clear ();

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

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

Magyarázat: A prgoramunk ugyan azt csinálja mint az "if"-es változata. Csak ugye most logikai kifejezés, utasítás nélkül. A megoldáshoz szükségünk van matematikai számításokra, ehez deklarálunk egész típusú változókat. A számításokat egy végtelen ciklusban számoljuk és mvprinw-vel írjuk ki a képernyőre. A clear()-el minden egyes számítás előtt letisztítjuk az ablakot. az eslő kettő mvprintw-vel a felső és alsó

határokat rajzoljuk ki. A 3 al pedig a "Labdát". Az Usleep függvény itt is a pattogás sebességét határozza meg.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Szóhossz:

```
#include <stdio.h>

int main()
{
    int a=1;
    int bit=0;
    do
    {
        bit++;
        while(a<=1);
        printf("%d %s\n",bit,"bites a szohossz a gepen");
    }
}
```

Ez a program a gépünk szó hosszát fogja kiírni, azaz az int méretét. A feladatot a BogoMIPS ben használt while ciklus feltétellel írjuk meg (A BogoMIPS a processzorunk sebességét lemérő program amit Linus Torvalds írt meg).

A main függvényben az első sor az `int a=1`. Itt deklaráljuk a változót, amivel vizsgáljuk meg a gépünk szóhosszát (Az int méretét). A "bit" változó fogja a lépéseket számlálni. A programot dowhile ciklussal (hátultesztelés) futtatjuk, mivel a sima while nem számítaná bele az első lépést, tehát ha a gépünk 32 bites, a program 31 bitet írna. A ciklus addig fut amíg az "a" nem lesz egyenlő nullával. És akkor mi is az a bitshift operátor. Ugye vesszük az 1-et, `a=1`. ennek a Bináris kódja a 0001, a bitshift operátor egy 0-val eltolja, azaz 0010 kapjuk, ez a 2 szám, a count növekedik tehát az értéke 1 lesz. A ciklus újra lefut és eltolja még egyszer a számot egy 0-val, így 0100 kapunk ami a négy. Ez addig fut, még a gépünk szó hosszán (az int méretén) kívül nem tolja az 1-et. Ekkor az a értékben csak 0 fog szerepelni, azaz az "a" értéke 0 lesz, a while ciklus befejeződik, és kiírjuk hányat lépett a ciklus, és ez a szám adja meg, hogy hány bites a szóhossz.

```
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/2.csonag$ gcc szo.c -o szoh
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/2.csonag$ ./szoh
32 bites a szohossz a gepen
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog1/2.csonag$
```

2.6. Helló, Google!

A PageRank egy keresőmotor amit a Google használ. A programot két fiatal írta meg 1998-ban. Nevét az egyik kitalálója után kapta.

A következőben, egy 4 lapból álló PageRank-at fogunk megnézni. A lapok PageRank-ét az alapján nézzük, hogy hány oldal osztotta meg a saját honlapján az oldal hiperlinkjét.

```
#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;
}
```

Forrás:https://progpatet.blog.hu/2011/02/13/bearazzuk_a_masodik_labort

Kezdjük az új headerrel, ez a math.h. Ez tartalmazza a matematikai számításokhoz szükséges függvényeket. A main() függvényben először is létrehozunk egy mátrixot, ami a lapok összeköttetését adja meg. Ha az érték 0 akkor a lap nincs összekötve az adott lappal és persze önmagával sincs. Ahol 1/2 vagy 1/3 az érték az azt jelzi, hogy az oldal hány oldallal van összekötve, például az 1/2: Az oldal 2 oldallal van összekötve és abból az egyik kapcsolatot jelzi (az 1).

A PR tömb fogja a PageRank értéket tárolni. A PRv tömb pedig a mátrixal való számításokhoz kell. A következő lépés egy végtelen ciklus. Ez majd a számítások végén a break paranccsal lép ki, ha a megadott feltétel teljesül. A for ciklusban van maga a PageRank számítása ami majd a távolság függvényt is meghívja, ami egy részsámolást tartalmaz. A végtelen cikluson belül lévő ciklusok azért 4 ig mennek mert 4 oldalt nézünk. A ciklusból való kilépés a "break" paranccsal történik majd ha a távolság függvényben kapott eredmény kisebb mint 0.00000001. A végén a kiir függvény megkapja a PR értékeket és az oldalak számát és kiírja azokat.

2.7. 100 éves a Brun tétel

A tételt Viggo Brun bizonyította 1919-ben. Ezért is nevezték el róla. A tétel kimondja hogy az ikerprímek reciprokösszege a Brun konstanthoz konvergál, ami egy véges érték.

Brun tétel R szimulációban:

```
library(matlab)

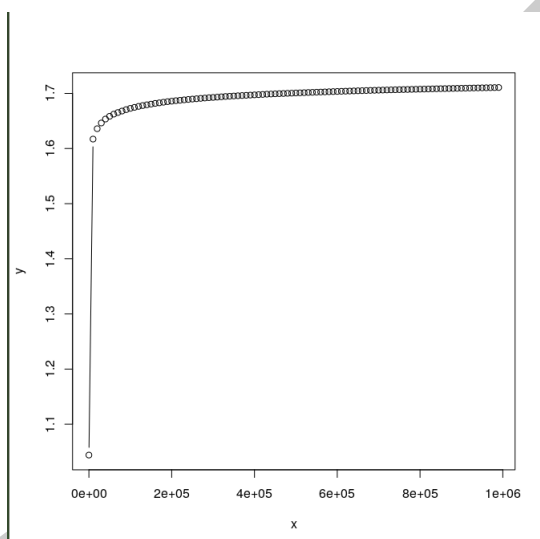
stp <- function(x){

    primes = primes(x)
    diff = primes[2:length(primes)]-primes[1:length(primes)-1]
    idx = which(diff==2)
    t1primes = primes[idx]
    t2primes = primes[idx]+2
    rt1plust2 = 1/t1primes+1/t2primes
    return(sum(rt1plust2))
}

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

A számoláshoz először is kell egy matlab könyvtár. A program fő része az `stp` függvény. egy a függvény megkapja `x`-et. `X` egy szám lesz ami megmondja meddig kell a prímeket számolni. Ehez a `primes` függvényt használjuk. `primes(x)` kiírja `x`-ig a prímeket. A `diff` vektorban eltároljuk a `primes` vektorban tárolt egymás melletti prímek különbségét. A számítást úgy végezzük, hogy a 2 prímtől indulva kivonjuk a prímből az előtte lévő prímet. Az `idx` el vizsgáljuk meg, hogy mely prímek különbsége 2 és ezek hol vannak (a helyüket a `which` függvény adja meg). a `t1primes` vektorban elhelyezzük ezeket a prímeket. A `t2primes` vektorba pedig ami ezeknél a prímeknél kettővel nagyobb (azaz ikerprímek). `rt1plust2` vektorban végezzük a reciprokképzést és a pár reciprokát összeadjuk. A `return`ban pedig a `sum` függvénnyel vissza adjuk ezek summázott összegét. Végezetül a `plot` függvénnyel lerajzoljuk grafikusan.

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



2.8. A Monty Hall probléma

Tutorált: Földesi Zoltán

Ez egy valószínűségi paradoxon. A kérdés egy vetélkedő játékból indul. Van 3 ajtó és az egyik mögött egy értékes nyeremény van, a másik kettő mögött semmi. A versenyzőnek a 3 ajtó közül választania kell egyet. Miután választott, a műsorvezető kinyit egy ajtót, ami mögött nincs a nyeremény. Felteszi a kérdést, hogy akarunk-e változtatni a választásunkon. Itt jön a felvetés, hogy megéri-e változtatni, vagy nem.

Megoldás: Első ránézésre mi is, és szinte mindenki azt mondaná, hogy nem számít, hogy vált-e mert 50-50% az esélye, hogy melyik ajtó mögött van a nyeremény. Mivel már nem 3 hanem 2 ajtó közül lehet választani, így már figyelembe se vesszük azt a harmadik ajtót. De a megoldás az, hogy igen, nagyobb az esélyünk akkor ha az előző döntésünket megváltoztatjuk és a másik ajtót választjuk.

Magyarázat: Kezdetben 3 ajtóból 1 ajtót kell választanunk, azaz $\frac{1}{3}$ az esélye, hogy eltaláljuk a jó megoldást és $\frac{2}{3}$ hogy nem. Ezek után a műsorvezető kinyit egy ajtót ami mögött nincs a nyeremény. Ez a kezdeti valószínűsége nem változtat, ugyanúgy $\frac{1}{3}$ eséllyel választottuk azt az ajtót ami mögött a nyeremény van. Viszont azok az ajtók közül ami mögött nincs semmi, azokból már csak az egyik van csukva. Biztosra tudjuk, hogy a nyeremény a maradék két ajtó közül valamelyik mögött van. Tehát $\frac{2}{3}$ az esélye annak, hogy a másik ajtó mögött van a nyeremény.

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MontyHall_R/mh.r

Szimuláció:

```
kiserletek_szama=10000
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)
```

Most a kísérletet 10000x fogjuk szimulálni. a kísérlet vektorban 1 és 3 "ajtó" közül választunk 10000x. A replace=T-val tesszük lehetővé, hogy egy eredmény többször is kijöhesse. A játékos választásait a jatekos vektornál ugyan így meghatározzuk. A sample() függvénnyel végezzük a kiválasztást. A musorvezeto vektort a length függvénnyel a kísérletek számával tesszük egyenlővé. Következik a for ciklus ami i=1 től a kísérletek számáig fut (10000). A ciklusban egy feltétel vizsgálat következik. az if-fel megvizsgáljuk, hogy a játékos által választott ajtó megegyezik-e a kísérletben szereplő ajtóval. Ha a feltétel igaz egy mibol vektorba beletesszük azokat az ajtókat amiket a játékos nem választott, az else ágon pedig ha a feltétel nem igaz, akkor azt az ajtót eltároljuk amit nem a választott és a nyereményt rejtő ajtót. A musorvezeto vektor-

ban pedig azt az ajtót amit ki fog nyitni. A nemváltoztat es nyer vektorban azok az esetek vannak amikor a játékos azt az ajtót választotta elsőre ami mögött az ajtó van és nem változtat a döntésén. A változtat vektorban pedig azt mikor megváltoztatja a döntését és így nyer ezt egy forciklussal vizsgáljuk. A legvégén kiíratjuk az eredményeket, hogy melyik esetben hányszor nyert.

DRAFT

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Forrás:<https://slideplayer.hu/slide/2108567/>

Magát a gép fogalmát 1936 -ban Alan Turing alkotta meg. A gép decimális számrendszerből unáris számrendszerbe írja át a számot. Az unáris számrendszer másnéven egyes számrendszer. A lényege, hogy 1 eseket írunk csak, ha az 1 számot akarjuk unárisba átváltani, az értéke egy, ha a 2-őt akkor az értéke 11, a tíz pedig 1111111111, Az a program c++ ban a következő:

```
#include <iostream>

using namespace std;

int main()
{
    int a;
    int tiz=0, szaz=0;
    cout<<"Decimalis szam:\n";
    cin>>a;
    cout<<"A szam unarisban:\n";
    for (int i=0; i<a; i++){
        cout<<"1";
        ++tiz;
        if (tiz==10) {cout<<" "; tiz=0;}
        if (szaz==100){cout<<"\n";szaz=0;}
    }
    return 0;
}
```

A kód egyszerű. Bekérünk egy decimális számot "a"-ba, és egy for ciklus segítségével addig írunk mindig egy 1-est amíg i(ami kezdetben 0 és mindig egyel növeljük) kisebb mint a. Én hogy a kimenet szebb legyen 2 változót használtam, 10 db 1 es után egy szóközt teszünk, míg 100 db után egy sortörést.

Magyarázat az Állapotmenet grafikájának:

A gép beolvassa a memóriaszalag számjegyeit, (Az ábrán a szám a 10) ha elér az "=" ig, az előtte lévő számmal kezd el dolgozni, még az 0 nem lesz. Az első elem egy 0, de mivel a következő nem nulla, hanem

1 ezért ebből kivon 1-et, azaz hátulról a második elemet 0-ra állítja. A kezdő elem ami 0 volt, az pedig 9 lesz, és ebből mindig kivon egyet, még 0 nem lesz, (8,7,6,5,4,3,2,1,0), minden kivonásnál kiírat sorban 1-et, így annyi 1 lesz, mint a decimális szám értéke. (Ha 100 lenne a szám akkor az 100 után 099 lenne aztán 098,097....089,088 és így megy 000-ig, és a kimeneten 100db 1-es lesz.)

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

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

A generatív nyelvek kidolgozását Noam Chomsky nevéhez fűzzük. A nyelveket osztályba rendezzük. Vannak erősebb és gyengébb osztályok. Az erősebb osztály képes létrehozni gyengébb osztályt.

Négy darab alapon fekszik a generatív nyelvtan:

1. Terminális szimbólumok. Azaz a konstansok.
2. Nem terminális jelek. Ezek a változók.
3. Kezdőszimbólum. Egy kijelölt szimbólum.
4. Helyettesítési szabályok. Ezzel szavakat értelmezzük majd.

Forrás: <https://slideplayer.hu/slide/2108567/>

1. nyelv

S, X, Y
a, b, c

Az S, X, Y lesznek a változóink. Az a,b,c pedig a konstansok

$S \rightarrow abc$, $S \rightarrow aXbc$, $Xb \rightarrow bX$, $Xc \rightarrow Ybcc$, $bY \rightarrow Yb$, $aY \rightarrow aaX$, $aY \rightarrow aa$

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

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

Azt láthatjuk, hogy egészen addig alkalmazzuk a helyettesítési szabályokat még csak konstansaink lesznek. Azaz mindig alsóbb osztályt hozunk létre.

2. Itt a változók az A.B.C és a konstansok a,b,c.

A, B, C legyenek változók
a, b, c legyenek konstansok

$A \rightarrow aAB$, $A \rightarrow aC$, $CB \rightarrow bCc$, $cB \rightarrow Bc$, $C \rightarrow bc$

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

de lehet így is:

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

3.3. Hivatkozási nyelv

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

Ahogy a beszélt nyelv, úgy a programozási nyelv is fejlődik. Ennek a bemutatására az alábbi programot fogjuk használni:

```
#include <stdio.h>

int main(){
    for(int i=0;i<1;i++){
        printf("Lefut");
    }
}
```

Itt ami lényeges, nem a kódban lesz, hanem a fordításnál. Megvizsgáljuk, hogy a C89 es nyelvten és a C99-es szerint hogyan fordítja le a programot a fordító. Ha a C89 es nyelvtannal fordítom: "gcc -std=gnu89 fajlnev.c -o fajlnev". A program hibát fog írni a for ciklusnál. Most ha a fordításnál átírjuk "gcc -std=gnu99 fajlnev.c -o fajlnev"-re (Azaz a fordító a 99 nyelvten lesz) ,akkor láthatjuk, hogy lemegy a fordítás és a program működik. A kódon belül, a for ciklusban deklaráltuk az int i-t.

magyarázat: Az okot a kódon belül, a for ciklusban kell keresni,ugyanis az "i" -t a for cikluson belül deklaráltuk. A C89 nyelvtenban ez még nem volt megengedett, így a fordító hibát írt, de a C99-ben már igen, ezért nem jelez hibát.

3.4. Saját lexikális elemző

A program a bemeneten megjelenő valós számokat összeszámolja.

A lexikális elemző kódja:

```
%{
#include <string.h>
int szamok=0;
}%
%%
[0-9]+      {++szamok;}
%%

int
main()
{
    yylex();
    printf("%d szam",szamok);
    return 0;
}
```

A szamok változóval számoljuk hányszor fordul elő szám a bemenetben. A programot a % - jelekkel osztjuk fel részekre. a

```
[0-9]+      {++szamok;}
```

Ez a sor adja azt, hogy 0-9 vagy nagyobb számot talál akkor növelje a "szamok" változót. A printf el pedig csak kiíratjuk hogy hány szám volt a bemenetben(ez az elemzés). A yylex() a lexikális elemző

a fordítás a következő:

```
flex program.l
```

ez készít egy "lex.cc.y" fájlt. ezt az alábbi módon futtatjuk.

```
cc lex.yy.c -o program_neve -lfl
```

A futtatáshoz pedig hozzá kell csatolni a vizsgált szöveget.

3.5. l33t.l

Tutor: Földesi Zoltán

Lexelj össze egy l33t ciphert!

```
%{
#include <string.h>
int szamok=0;
}%
%%
"0" {printf("o");}
"1" {printf("i");}
"3" {printf("e");}
"4" {printf("a");}
"5" {printf("s");}
"7" {printf("t");}

"o" {printf("0");}
"i" {printf("1");}
"e" {printf("3");}
"a" {printf("4");}
"s" {printf("5");}
"t" {printf("7");}

%%

int
main()
{
    yylex();
    printf("%d szam",szamok);
    return 0;
}
```

Ez a program lefordítja a l33t nyelven írt titkos szöveget vagy a rendes szöveget írja át a l33t nyelvre.

A program működése az előzővel majdnem megegyezik, csak annyiban tér el, hogy valós számok helyett, itt most a megadott számokat keresi a bemenetben és azok a számok helyett a l33t nyelvben való megfelelő betűket írja a helyére. Ha pedig a l33t nyelvre akarjuk fordítani, akkor a betűket vizsgálja és a megfelelő számot írja be.

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

**Bugok**

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

i.

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

A példában szereplő kód részlet ellentetjé, azaz ha a SIGINT jel kezelése nem lett figyelmen kívül hagyva akkor, a jelkezelő függvény kezelje.

ii.

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

Ez egy for ciklus, benne az i értéket 0 állítjuk, és amíg az i értéke kisebb mint 5 addig a ciklus újra és újra lefut. A 3 argumentumban növeljük az i értékét.

iii.

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

A ciklus majdnem ugyan az mint az előző. az eltérés az i érték növelésében van, nem tűnik nagy eltérésnek, de fontos. Az előzőbe ++i míg itt i++. A különbség az, hogy a ++i-nél először növeli az i értékét, aztán az i értékét átadja, még az i++ először átadja az i értékét és aztán növeli az i értékét. Ez a for ciklusban úgy van ++i-nél hogy megnöveli egyel az i-t és utána hajtja végre újra a lefutást. Az i++ nál pedig először végrehajtja aztán növeli az i értékét.

iv.

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

Ez a for ciklus egy tombot feltölt az i értékével. a tomb ezek után úgy fog kinézni, hogy tomb[5]={0,1,2,3,4}, mivel i++, ezért előbb átadja az értéket és utána növeli az i értékét. Bug: A programba máshogy viselkedik, mivel az első érték mindig egy memóriaszemét lesz. A megoldás, hogy a for cikluson belül adjuk hozzá a tombhoz az értéket for(){ ezen belül }.

v.

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

Itt a for ciklusunk második argumentumába az i kisebb mint n feltétel mellett van egy másik feltétel. A for ciklus csak akkor fut le ha mind a 2 feltétel teljesül. A második feltétel az, hogy az s és a d mutató egyenlő (minden ciklusnál növeljük az értékeket). A feltételt az és operátorral kötjük össze. Bug: A hiba, hogy a második feltétel nem logikai feltétel. Ezt a feltételt is egy if el a for cikluson belül kéne vizsgálnunk.

vi.

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

A printf függvénnyel kiíratunk valamit. Ebben az esetben két egész típusú változót. A printf-en belül az f függvénnyel határozzuk meg a számot. Bug: Rossz a sorrend, ezért hibát kapunk.

vii.

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

A printf függvénnyel kiíratunk két egész számot, az első számot az f függvény adja (az f függvény az "a"-t kapja meg), míg a másik az a változó értéke.

viii.

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

A printf függvénnyel kiíratunk két egész számot. Az előzőnél annival másabb, hogy a függvény az a memória címét kapja meg.

3.7. Logikus

Tutorált: Földesi Zoltán

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

Minden x esetén létezik olyan y ahol $x < y$ és y prím. Ez azt jelenti, hogy a ←
prímek száma végtelen.

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

Minden x esetén létezik olyan y ahol $x < y$ és y prím és az $\exists z (z < y) \wedge (z \text{ prim})$ ←
lefordítva azt jelenti, hogy az ikerprímek száma végtelen.

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

Létezik olyan y, minden x számra, hogy ha x prím akkor $x < y$, lefordítva a ←
prímek száma véges.

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

Létezik, olyan y ami minden x számra $y < x$ akkor hax nem prím, lefordítva ←
ugyan azt jelenti mint az előző, csak tagadással megfogalmazva.

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

A megoldáshoz tudnunk kell mi mit jelent, vannak a logikai összekötőjelek, mint az \wedge =és, \neg =nem, \vee =vagy, \supset =implikáció A kiíratást a \text el végezzük. Vannak kvantorok a "létezik"= \exists és a "minden"= \forall . Az "S" értéknövelés.

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

A program:

```
#include <iostream>

int main(){
    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);
}
```

Mit vezetnek be a programba a következő nevek?

- `int a;`

Egy egész típusú változót deklarál.

- `int *b = &a;`

Egy int típusú mutatót deklarál, ami képes egy változó memóriacímét tárolni. "b" mutató "a" ra mutat.

- `int &r = a;`

Egy egész típusú referenciát deklarál, ami hasonló a mutatóhoz, de nem ugyan az, a referencia úgymond egy állnév, pontosabban egy már létező változóhoz egy másik név.

- `int c[5];`

Ez egy egész típusú 5 elemű tömb.

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

Ez egy referenciája a "c" 5 elemű tömbnek (Az összes elemnek).

- ```
int *d[5];
```

A d tömbben minden egyes tag egy mutató.

- ```
int *h ();
```

Az int típusú változó visszatérési típusát tartalmazó függvény.

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

Egy egész típusra mutató mutatót visszaadó függvény.

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

Egy egész típusút afo és két egész típusút kapó függvényre mutató mutatót visszaadó, egész típust kapó függvény

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

Egy egész típust visszaadó és két egész típust kapó függvényre mutató mutatót visszaadó, egész típust kapó függvényre

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

A következő programban egy alsó háromszögmátrixot hozunk létre.

Forrás: https://gitlab.com/nbatfai/bhax/blob/master/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/-Caesar/tm.c

a kód:

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

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

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

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

    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;
*(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;
}
```

Magyarázat: Szokás szerint includoljuk a szükséges include-kat. A fő függvényben az első sora az "int nr=5" itt adjuk meg, hogy 5 sorunk legyen a kimeneten. A "double **tm", sorral foglalunk le tárhelyet a memóriában. Az első ifben megtaláljuk a malloc függvényt ami dinamikus memória foglaló, ezzel nr számú double ** mutatót foglalunk le, ha null értéket ad vissza az azt jelzi ,hogy nincs elég hely a foglaláshoz. A következő if lefoglalja a mátrix sorait, az első sornak egy double * mutatót foglal le, a másodiknak 2 , a harmadiknak 3 , nr ig. A 3. for ciklussal megadjuk a mátrix elemeit. Az "i" a matrix sorai, a "j" pedig a benne lévő mutatók. a "tm[i][j]=i*(i+1)/2+j; érjük el azt, hogy az elemek mindig egyel nőjenek. A 4. for ciklus pedig a kírátás. Ezek után már csak annyit csinálunk, hogy a 3 sort megváltoztatjuk, mert így is ki lehet írni. A legvégén pedig a free()-vel felszabadítjuk a lefoglalt memóriát, ezzel megelőzve a memóriafolyást.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

A feladat lényege , hogy egy szöveget titkosítsunk Exor-ral(XOR). Az XOR a "kizáró vagy". A szöveget az alábbi módon titkosítjuk: Az eredeti szöveg bájtjaihoz rendelünk titkosító kulcs bájtjokat. Aztán XOR-t műveletet végzünk rajta. Az XOR-t művelet úgy működik, hogy ha a bitek azonosak (1,1;0,0) akkor 0 ad vissza értéknek, ha pedig különbözőek (1,0;0,1) akkor 1-et ad vissza, és így minden bitpáron elvégezve ezt megkapunk egy titkosított szöveget.

Forrás:https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_paruhuzamos_prog_linux/ch05s02.htm

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);

    }

}
```

A kód magyarázata: Először is beinclude-oljuk szükséges include-kat. Aztán két állandó változót definiálunk a #define parancsal. Ezeknek az értéke nem változik. Az első állandó a MAX_KULCS az értéke 100. A második pedig a BUFFER_MERET 256, ez nekünk a beolvasásnál fog kelleni. A fő függvényben egy-egy char típusú tömb méreteivé tesszük a 2 állandót. Ezek után 2 változót hozunk be, a kulcs_index, ami a kulcsunk aktuális elemét tárolja, és az olvasott_bajtok ami a beolvasott bajtok összegét tárolja. A kulcs_merete változóban a kulcs méretét adjuk meg a "strlen()" függvény segítségével, amit mi adunk meg egyik argumentumként. Az strncpy függvény pedig a kulcs kezeléséhez kell. Ezután a while ciklusban beolvassuk a buffer tömbbe a bemenetet, a while ciklus addig fut, ameddig van mit beolvasni. A read függvénnyel lépünk ki a ciklusból. A while cikluson belül a forciklusban végig megyünk az összes bajton és végre hajtjuk a titkosítást.

A futtatás a következő: A fordítás: gcc fajlnev.c -o fajlnev miután lefut, utánna futtatjuk: ./fajlnev 56789012 (ez a kulcs) titkositando.txt (ide írjuk a titkosítandó txt fájl nevét, relíciós jelek között) > titkos.szoveg (titkosított fajlneve). A titkos szöveget a more titkos.szoveg parancsal nézhetjük meg.

4.3. Java EXOR titkosító

Tutor: Mózes Nóra.

A forrást a tutortól származik. A feladatban az előző feladatot fogjuk megírni Java-ban. A könyvben most találkozunk először a Java nyelvvel. A Java egy objektumorientált programozási nyelv, azaz a nyelv objektumokból, osztályokból áll. A Sun Microsystems informatikai cég alkotta meg. Maga a nyelv a C és a C++ nyelvekhez hasonló, azonban sokkal egyszerűbb (az említett objektumorientáltság miatt). A kezdéshez beszéljünk kicsit az osztályokról, azaz a "Class"-okról. A Classok egy függvények csoportja. Van public és private része, a publikus függvényeket a programból bármi meghívhatja, míg a private függvényeket vagy változókat csak az osztályon belüli vagy barát függvények hívhatják meg.

```
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();
        }
    }
}
```

Az ExorTitkosito() függvény, kapja meg a bekért argumentumokat. Ha rosszul kapja meg, a throw() hibát ad vissza. A függvény beldejében történik a titkosítás XOR-al. Ez ugyan úgy működik, mint a fenti C kódban. Ami érdekes lehet számunka az a byte típus, ez 8-bit. Byte típusú lesz a kulcs és a buffer tömb is,

ezek tárolják a kulcsot és a beolvasott szöveget.

Vizsgáljuk meg a "main". A Java nyelvben a main az osztály egyik függvénye (eltér a C++ -tól, ahol a main egy különálló fő függvény az osztálytól.) Az alábbi sor "public static void main(String[] args)" a main függvény fejléce. A "public" mutatja, hogy publikus, azaz elérhető. A "static"-al jelöljük, hogy része az osztálynak. A void típust meg már ismerjük az előzőkből. A main-be képesek vagyunk argumentumokat bekérni a terminálból. A main-en belül láthatjuk a try() és a catch() függvényt, ezekkel a függvényekkel C++ -ban, A try() a hiba üzenetet küldi még a catch() ezt elkapja és kiírja nekünk.

A fordításhoz java fordító kell. Ehez most a "javac"-t fogjuk használni. Ha ez nincs fent a számítógépünkön, akkor a gép jelezni fogja, hogyan kell telepítenünk. Fordítani és futtatni az alábbi módon fogjuk:

```
//Fordítás:
javac ExorTitkosító.java
//Futtatás:
java ExorTitkosító titkosítandó.szöveg > titkosított.szöveg
```

4.4. C EXOR törő

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

Az alábbi feladatban a 3.2 feladatban lévő titkosítóhoz írunk egy programot ami feltöri a titkosított szöveget. A program alapműködése ugyan azon az elven alapszik, mint a 3.2 mivel ugyan így XOR- al alakítjuk vissza a szöveget. A lényeg, hogy a kulcsot amivel titkosítottunk azt ismerjük, mert ezzel a kulccsal tudjuk feltörni. Úgy működik, hogy a titkosított bájtokat össze exoráljuk a kulccsal, és így újra az eredeti bájtokat kapjuk. A feladatban a 3.2 ben titkosított szöveget és a kulcsot fogjuk használni, ugyanis erre épül a program.

Kód:

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

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

int tiszta_lehet(const char titkos[], int titkos_meret)
{
    return strcasestr(titkos,"hogy") && strcasestr(titkos,"nem") && ←
        strcasestr(titkos,"az") && strcasestr(titkos,"ha");
}

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

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

int exor_tores(const char kulcs[], int kulcs_meret, char titkos[], int ←
    titkos_meret)
{
    exor(kulcs, kulcs_meret, titkos, titkos_meret);
    return tiszta_lehet(titkos, titkos_meret);
}

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

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

    for(int i=0; i<MAX_TITKOS-(p-titkos);++i)
        titkos[p-titkos+i]='\0';

    for(int ii='0';ii<='9';++ii)
        for(int ji='0';ji<='9';++ji)
            for(int ki='0';ki<='9';++ki)
                for(int li='0';li<='9';++li)
                    for(int mi='0';mi<='9';++mi)
                        for(int ni='0';ni<='9';++ni)
                            for(int oi='0';oi<='9';++oi)
                                for(int pi='0';pi<='9';++pi)
                                {
                                    kulcs[0]=ii;
                                    kulcs[1]=ji;
                                    kulcs[2]=ki;
                                    kulcs[3]=li;
                                    kulcs[4]=mi;
                                    kulcs[5]=ni;
                                    kulcs[6]=oi;
                                    kulcs[7]=pi;
```

```
if(exor_tores(kulcs,KULCS_MERET,titkos,p-titkos))
    printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",ii,ji,ki,li ←
        ,mi,ni,oi,pi,titkos);
exor(kulcs,KULCS_MERET,titkos,p-titkos);
}
return 0;
}
```

Forrás:https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_parhuzamos_prog_linux/ch05s02.htm

Elsőnek is definiáljuk az állandókat és az include-kat. Az állandók közül most is a buffer a beolvasáshoz szükséges, a kulcs mérete megint a kulcsot tartalmazó tömbhöz kell ami az előzőleg használt kód miatt 8. A fő függvény előtt találunk függvényeket. Az átlagos szóhossz és a tiszta lehet függvény a törés gyorsaságát segítik elő. Az átlagos szóhossz megadja az szóhossz átlagát még a tiszta lehet pedig a gyakori magyar szavak figyeli. A void exor () függvény megkap egy kulcsot, a méretét, a titkos szövegetnek a tömbjét és annak a méretét. És itt a forciklusban a kulcsot össze exortálja a titkos szöveggel. Az exor_tores függvény meghívja az exor függvényt is vissza adja a tiszta szöveget. A fő függvényben láthatjuk deklarációk után a titkos szöveg beolvasását. Utánna a program megnézi az összes lehetséges permutációt és a megoldást kírátja a kimenetre, ezzel a kóddal a 3.2 programot használva fel tudjuk törni a szöveget.

4.5. Neurális OR, AND és EXOR kapu

R

A feladatban egy Neurális hálózatot fogunk írni R nyelvben. A nevét a neuron-ról kapta, ami egy ideg-sejt, Ezekből épül fel az idegrendszer. Ez egy ingerlékeny sejt, ami ingerület fel és leadásával továbbít információt, amit fel is dolgoz.

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

```
library(neuralnet)

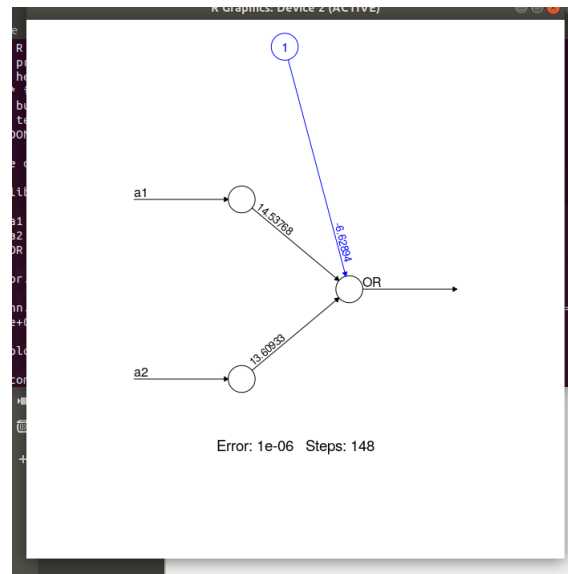
a1    <- c(0,1,0,1)
a2    <- c(0,0,1,1)
OR    <- c(0,1,1,1)

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])
```



A program elején meghívjuk a neuralnet könyvtárat ami tartalmazza a nekünk szükséges függvényeket. A bemenet az a1 és az a2 lesz, A gép most a logikai vagyot, azaz az OR -t fogja megtanulni. Ha a1 és a2 bemenet 0 ad, Az OR értéke is 0 lesz, minden más esetben az OR értéke 1. Ezeket az or.data-ban tárolja el a program, úgy mond "megtanulja". Az nn.or értékét pedig a neuralnet() függvénnyel határozzuk meg. A függvény első argumentumában a megtanulandó érték van, azaz hogy az OR értéke 0 legyen vagy 1. A második argumentumban adjuk meg az or.data ami alapján tanulja meg a program. A harmadik argumentumban rejtett neuronok száma van. A stepmax a lépésszámot adja. A plot függvénnyel kirajzolunk (lásd a képen) a tanulás folyamatának egyik esetét.

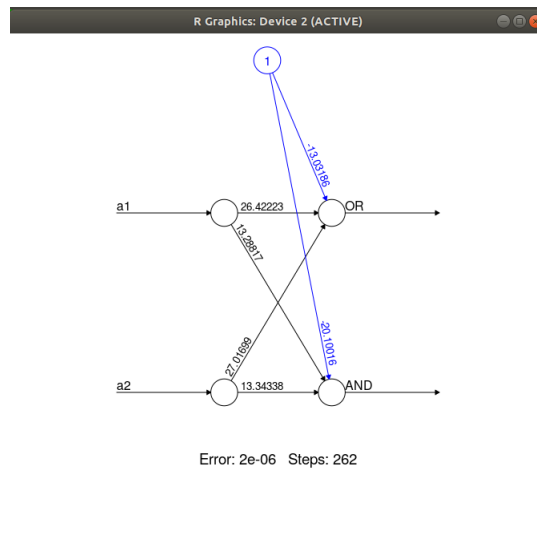
```
library(neuralnet)
a1 <- c(0,1,0,1)
a2 <- c(0,0,1,1)
OR <- c(0,1,1,1)
AND <- c(0,0,0,1)

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])
```



A programunk azzal bővül, hogy megtanítjuk a programnak az OR és az AND -et fogja megtanulni a program. A különbség az előzőtől annyi, hogy az AND csak akkor kap 1 értéket, ha a1 és a2 értéke is 1, különben az AND értéke 0. A tanulás folyamat ugyan olyan mint az előző. A tanulás módját az orand.data-ba mentjük.

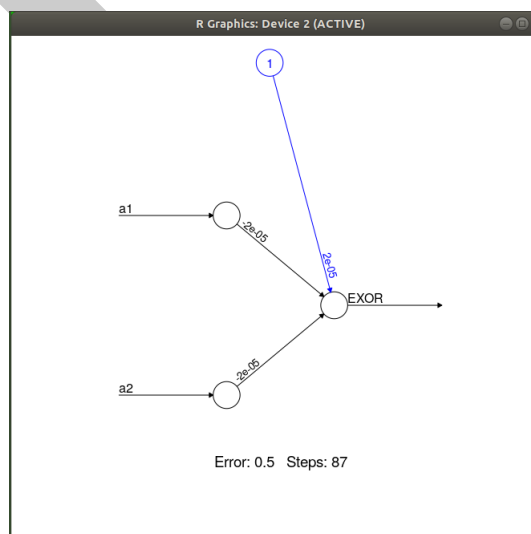
```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)

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])
```



Itt pedig az EXOR-t tanítjuk meg a programmal. Az EXOR-nál az EXOR értéke akkor 1, ha az a1 és a2

értéke 1,0 vagy 0,1 . Ha mind akét érték 0,0 vagy 1,1 akkor az EXOR értéke 0 lesz. Ezt a tanulási mintát az exor.data-ban mentjük el. És a tanulás pont ugyanúgy van mint a fentiekben. A képen láthatjuk, hogy a program nem tanulta meg amit kell, ugyanis az eredmények hibásak. A kulcs abban van, hogy a rejtett neutronok értéke 0. A következőben nézzük meg a megoldását.

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

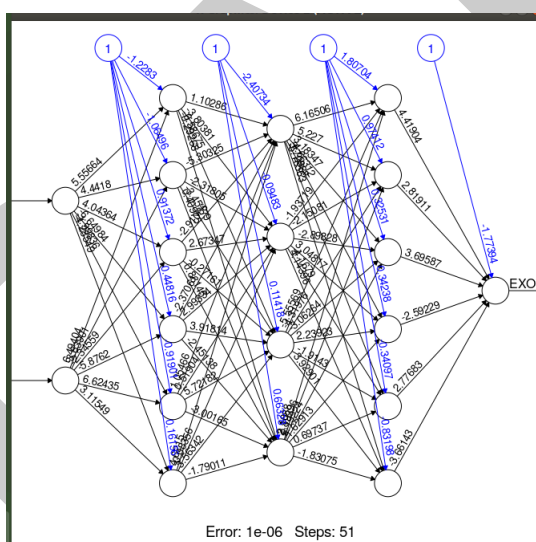
exor.data <- data.frame(a1, a2, EXOR)

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])
```

Itt anyiban változtattunk, hogy a rejtett neutronoknak létrehoztunk 3 réteget , a rétegek értékei 6,4,6. Ahogy a képen is látszik, az eredmény így jó.



4.6. Hiba-visszaterjesztéses perceptron

C++

A perceptron a mesterséges intelligenciának olyan, mint az agynak a neuron. A program képes feldolgozni és megtanulni a bemenetet, ami 0,1ből áll.

Forrás: <https://youtu.be/XpBnR31BRJY>

Kód:

```
#include <iostream>
#include "mlp.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;
}
```

A kód magyarázata: két headere van szükségünk az "mlp.hpp" és a "png++/png.hpp" -re, ezek a megjeleníté miatt kellenek nekünk és ebbe van a perceptron elve is. A fő függvényünk elején lefoglaljuk a tárhelyet a képnek és megadjuk a méreteit. Következik a perceptron létrehozása és a megfelelő értékek hozzá adása. A "double* image = new double[size];" sorral a végélétrehozunk egy size méretű képet és utánna feltöltjük a megadott képpel. a delete parancsokkal töröljük a perceptront és a képet.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Mandelbrot halmaz egy halmaz a komplex számsíkon. Nevét Benoit Mandelbrot kapta, aki megfogalmazta a fraktálok fogalmát (A fraktálok komplex alakzatok).

Forrás: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsocpp/mandelbrot/mandelbrot.cpp>

Kód:

```
include "png++-0.2.9/png.hpp"

#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35

void GeneratePNG( int tomb[N][M] )
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y] ←
            );
        }
    }
    image.write("kimenet.png");
}

struct Komplex
{
    double re, im;
```

```
};

int main()
{
    int tomb[N][M];

    int i, j, k;

    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;

    struct Komplex C, Z, Zuj;

    int iteracio;

    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            C.re = MINX + j * dx;
            C.im = MAXY - i * dy;

            Z.re = 0;
            Z.im = 0;
            iteracio = 0;

            while (Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < 255)
            {
                Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
                Zuj.im = 2 * Z.re * Z.im + C.im;
                Z.re = Zuj.re;
                Z.im = Zuj.im;
            }

            tomb[i][j] = 256 - iteracio;
        }
    }

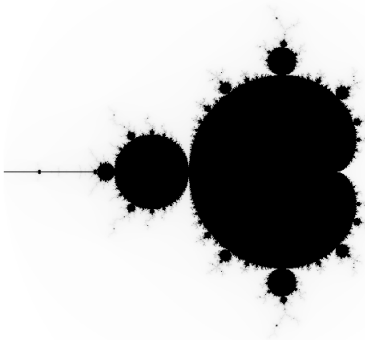
    GeneratePNG(tomb);

    return 0;
}
```

A kód magyarázata: Az include-ról kicsit lejjebb, bővebben kifejtve beszélek majd. A kódot állandók definiálásával kezdjük, ilyen lesz a kép maximum szélessége, magassága. Az első függvény fogja nekünk legenerálni a képet. A "png" csomagot használjuk ehhez. Létrehozunk egy üres pngt ami 500x500 pixel ((500X500 as mátrix)). A forcikluson belül rgb színkóddal határozzuk meg a színes pixeleket. és a "image.write" a képet kiküldjük a kimenetre egy adott névvel. ez a függvény a fő függvény legalján lesz meghívva. A következő egy struktúra amiben 2 double típusú változót deklarálunk , ez a komplex szá-

moknak a struktúrája. Ezután a fő függvényben létrehozunk egy tömböt ami 500x500 elemű. Ezekhez az állandókat használjuk. 3 egész típusú deklaráció után 2 double változót deklarálunk a "dx" és "dy" amivel a pixeleket fogunk meghatározni. A következő sorban lefoglaljuk a helyet c, z, zuj változoknak, utána elvégezzük a számításokat és beletesszük azokat a tömbbe és meghívjuk a függvényt amivel legeneráljuk.

Most nézzük meg a headert. A png++ headerre van szükségünk ahhoz hogy png-t tudjunk kezelni. Ez alpból nincs meg a gépen, ezért először is le kell töltenünk az internetről egy fájlt ami tartalmazza a headert. Miután ezt letöltöttük, még telepíteni kell a libpng könyvtárat az alábbi módon: "sudo apt-get install libpng++-dev".



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

Itt a feladat ugyan az mint az előző. A különbség az, hogy itt most használhatjuk a complex headert. Ez a header már alpból tartalmaz komplex számokat, így az előző feladatban létrehozott struktúra itt már nem fog kelleni.

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

```
#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 )
double dy = ( d - c )
double reC, imC, reZ, imZ;
int iteracio = 0;

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

for ( int j = 0; j < magassag; ++j )
{

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

        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;
        }

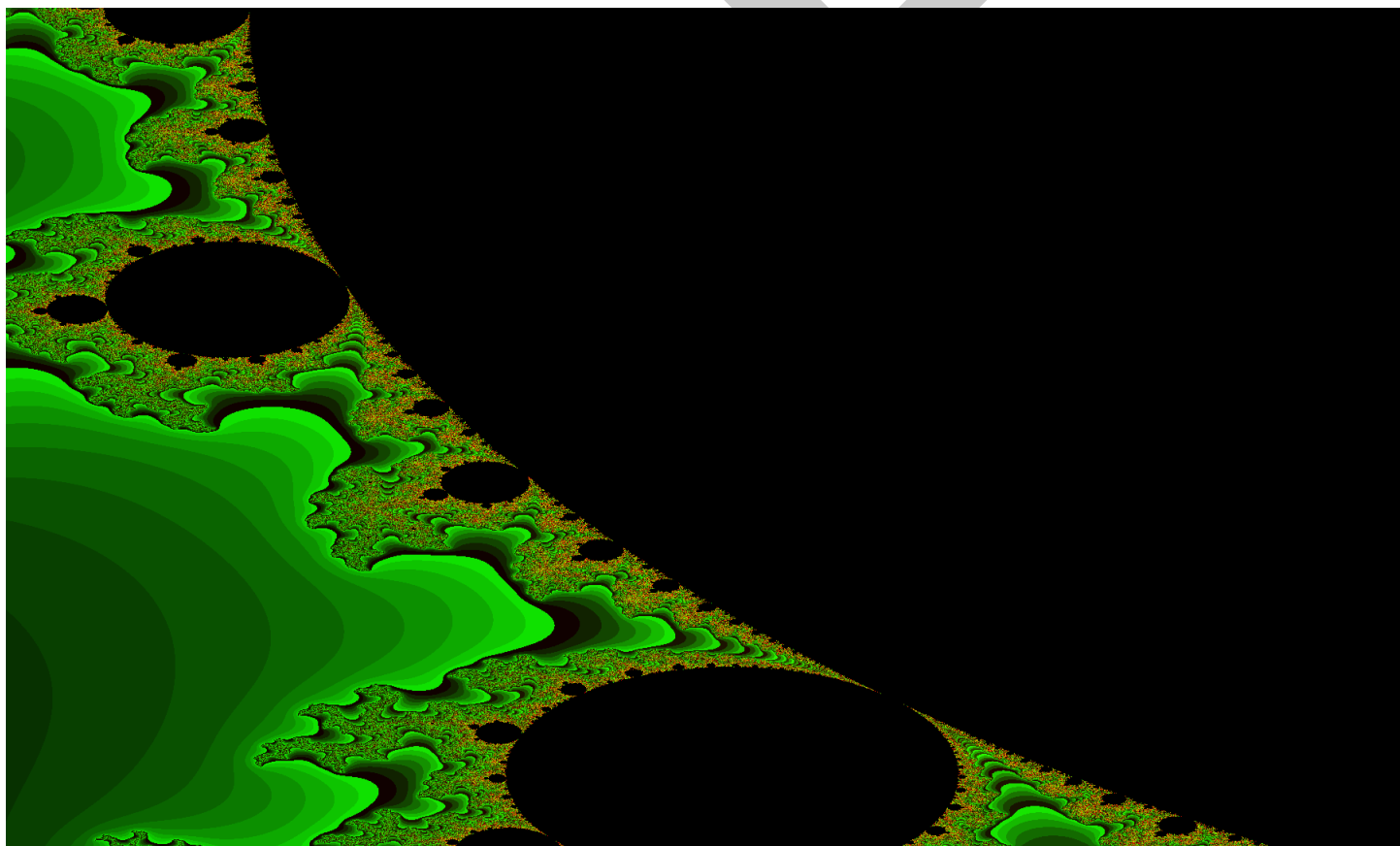
        kep.set_pixel ( k, j,
                        png::rgb_pixel ( iteracio%255, (iteracio*iteracio ↵
                            )%255, 0 ) );
    }

    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;  
  
}
```

Kód magyarázata: A headereket már megbeszéltük. a fő függvényben deklarálunk 2 változót, ha argumentumként jól adjuk meg ezeket, akkor ezeket átadja a változóknak, ha nem jól adjuk meg, akkor kiírjuk, hogy kell helyesen használni. Ezek után megadjuk a szélességet és a magasságot, ami ebbe az esetben FullHD. és az iterációs határt. továbbá deklarálunk változókat amik a kép elkészítéséhez kellenek majd. Az if függvény vizsgálja meg, hogy jól adtuk-e meg az argumentumot. és itt adja át az előbb említett értékeket. Az else ág a rossz esetén, a segítséget írja ki. Ezek után lefoglaljuk a helyet a képnek. A dx, dy-hez hozzárendeljük a megfelelő változókat. A forciklusban végig megyünk minden elemén és megadjuk a c változó értékét. Ekkor használjuk a complex-et, while ciklusban végezzük a számításokat, utána rgb kóddal a pixeleket kiszínezzük.

A futtatáshoz szükségünk lesz a -lpng kapcsolóra.



5.3. Biomorfok

A biomorf program a mandelbrot programkódját vesszük alapul. A mandelbrot halmaz tartalmazza az összes ilyen halmazt. A program ugyanúgy bekéri a megfelelő bemeneteket, ha nem jó akkor kiírja. Ha jó, akkor a megfelelő változók megkapják a megfelelő értékeket. Ezután történik a kép létrehozása. Ugyanúgy megkapja a dx és dy az értéket. Aztán pedig a komplex számokat gozzuk létre. Megint végig megy a

program minden ponton és ahol kell használjuk az rgb kódos színezést. A legvégén pedig kiküldjük a képet a kimenetre.

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

Kód:

```
#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";

for ( int y = 0; y < magassag; ++y )
{

    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)
        {

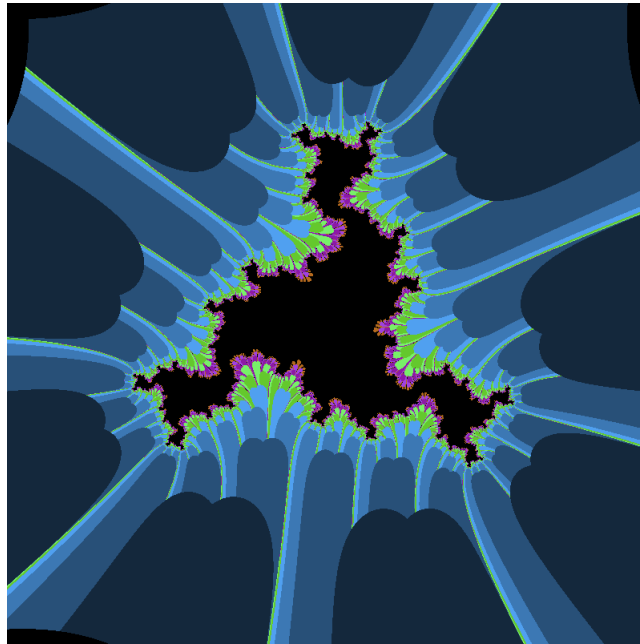
            z_n = std::pow(z_n, 3) + cc;

            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;
}
```



5.4. A Mandelbrot halmaz CUDA megvalósítása

Továbbra is a mandelbrot halmazzal foglalkozunk, de mi is az a cuda? A CUDA az Nvidia videokártyáknak egy párhuzamos számításokat segítő technológia. Használható C és C++ nyelveknél is. Ezen technika segítségével fogjuk felgyorsítani a kép létrehozását. Ehet szükségünk lesz egy Nvidia videokártyára ami rendelkezik CUDA-val. Továbbá telepítenünk kell. A kód kiterjesztése ".cu"

Megoldás forrása: https://progpater.blog.hu/2011/03/27/a_parhuzamossag_gyonyorkodtet

Kód:

```
#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)
{
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
```



```
int iteracio = 0;

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

reZ = 0.0;
imZ = 0.0;
iteracio = 0;

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;
}
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 / 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[])
{

clock_t delta = clock ();

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)
{
    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;

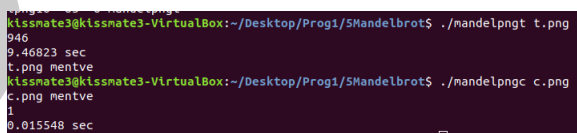
    delta = clock () - delta;
    std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;

}

```

Nézzük a kódot. Az include-k alatt két állandót definiálunk, a kép méretét és az iterációs határt. A következő lépés a Mandelbrot halmaz létrehozása. ezt egy függvénnyel hozzuk létre. A függvény előtt jelezzük, hogy a számításokat Cudával végezzük majd a fordításnál. A függvényen belül deklarálunk float típusú változókat a számításokhoz. A matematikai számítás ugyan az mint az 5.1 feladatban, szóval ezt nem fejtem ki most. A következő függvény előtt nem "__device__" jelzés van hanem "__global__". Ezzel szintén azt jelezzük, hogy a Cuda fogja végezni a számítást. A "threadIdx" jelzi az aktuális szálat és a "blockIdx" hogy, melyik blokkban folyik a számítás. A kép értékeit a j és a k változóknak tároljuk el. Ezt a két értéket fogja kapni az előző függvény. A következő függvény a cudamandel(). Ez egy Méret x Méret azaz 600x600-as tömböt kap. Deklarálunk egy mutatót és a Malloc segítségével lefoglaljuk a megfelelő tárhelyet és a mutató ide fog mutatni. Itt hozzuk létre a megfelelő blokkokat. A végén a tárhelyet felszabadítjuk. A fő függvényünkben sem történik nagy változás. Egyből egy idő méréssel kezdünk. Lemérjük mennyi időbe telik a gépnek, hogy megalkossa a képet. Utánna deklaráljuk a tömböt, meghívjuk a cudamandel() függvényt és már az ismert módon létrehozuk a képet.

A kódot az "nvcc" fordítóval fordítjuk, le kell tölteni, ehhez a gép ad segítséget. A következőléppen fordítjuk: "nvcc mandelpngc_60x60_100.cu -lpng16 -O3 -o mandelpngc ". Miután fordítottuk utánna futtatjuk. Ha egymás mellé tesszük a Cudas és a nem kudas képkalkotást, láthatjuk, hogy a kép elkészítési ideje a cudásnál sokkal gyorsabb.



```

kissmate3@kissmate3-VirtualBox:~/Desktop/Progi/5Mandelbrot$ ./mandelpngt t.png
9.46823 sec
t.png mentve
kissmate3@kissmate3-VirtualBox:~/Desktop/Progi/5Mandelbrot$ ./mandelpngc c.png
c.png mentve
0.015548 sec

```

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!

A program azt fogja csinálni, hogy létrejön nekünk egy mandelbrot halmaz és az egérrel képesek vagyunk belenagyítani akár a végtelenségig a halmazba.

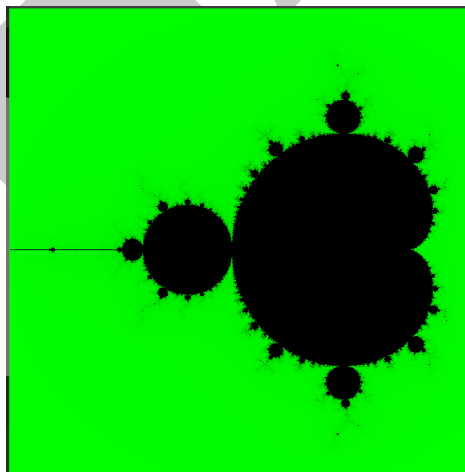
Kód:

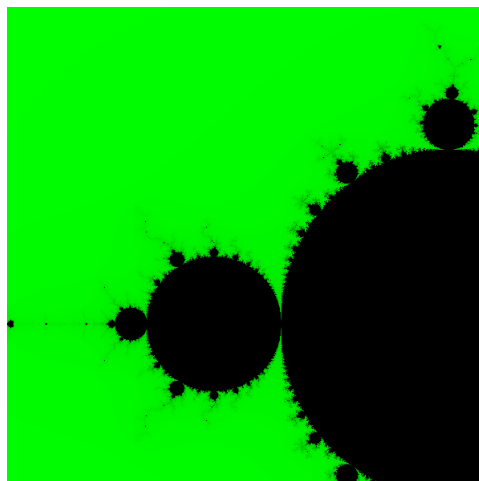
```
#include<QApplication>
#include "frakablak.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Frakablak w1,
    w2(-.08292191725019529, -.082921917244591272,
        -.9662079988595939, -.9662079988551172, 1200, 3000),
    w3(-.08292191724880625, -.0829219172470933,
        -.9662079988581493, -.9662079988563615, 1200, 4000),
    w4(.14388310361318304, .14388310362702217,
        .6523089200729396, .6523089200854384, 1200, 38655);
    w1.show();
    w2.show();
    w3.show();
    w4.show();
    return a.exec();
}
```

Ez a program nem elég önmagában, több forrásra van szükségünk. Ilyen például a `frakablak.h` header. Egy mappába össze kell szednünk az összes forrást, és telepítenünk kell ezt: `"sudo apt-get install libqt4-dev"`. A `qmake -project` paranccsal létrehozunk egy `.pro` fájlt. ebbe meg kell adnunk a `QT+=Widgets` parancsot a megfelelő helyre. Ez létrehoz egy fájlokat `.o` kiterjesztéssel és egy `makefile`t, ezek után `make` paranccsal létrehozzuk a nagyítót. Ezek után kész is a programunk. A `frakszal.cpp`-ben készül el az ábránk amit majd nagyítani fogunk. Az `rgb` pixel színezést azonban már a `frakablak` végzi.

Forrás: https://progpater.blog.hu/2011/03/26/kepess_egypercsek





5.6. Mandelbrot nagyító és utazó Java nyelven

Ebben a feladatban az 5.5 feladatot fogjuk megírni Java-ban. A program lényege itt is az, hogy a mandelbrothalmazba belenagyítunk.

A program elején létrehozuk a Mandelbrot halmazt. Ehez az `extends` szóval hozzá kapcsoljuk a Mandelbrothalmazt építő java kódunkat. A `mousePressed()` függvényel megadjuk a programnak az egér által kijelölt kordinátákat. Ezután A kijelölt területen újraszámoljuk a halmazt. Majd feldolgozza a létre jött kép szélét és magasságát. A `pillanatfelvétel()` függvényel egy pillanatfelvételt készítünk. A függvényen belül elnevezzük a tartomány szerint és egy png formátumú képet készítünk a pillanatfelvételtől. A nagyítás során láthatunk egy segítő négyzetet, ezt a `paint()` függvényel hozzuk létre.

A kód forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html>

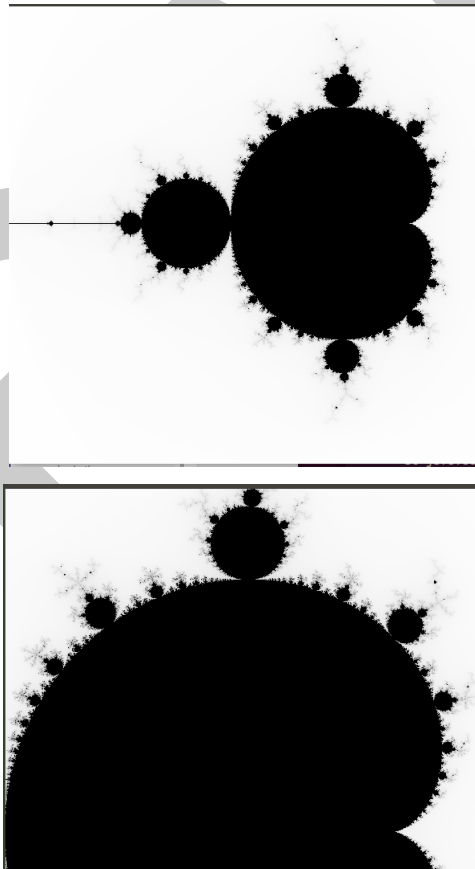
```
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    private int x, y;
    private int mx, my;
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
        int szélesség, int iterációsHatár) {
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        addMouseListener(new java.awt.event.MouseAdapter() {
            public void mousePressed(java.awt.event.MouseEvent m) {
                x = m.getX();
                y = m.getY();
                mx = 0;
                my = 0;
                repaint();
            }
        });
        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;
```

```
        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);
    }
});
addMouseListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent m) {
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
}
public void pillanatfelvétel() {
    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();
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrotHalmazNagyitas_");
    sb.append(++pillanatfelvételSzámláló);
    sb.append("_");
    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
    sb.append("_");
    sb.append(d);
    sb.append(".png");
    try {
```

```
        javax.imageio.ImageIO.write(mentKép, "png",
                                     new java.io.File(sb.toString()));
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
}

public void paint(java.awt.Graphics g) {
    g.drawImage(kép, 0, 0, this);
    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);
}

public static void main(String[] args) {
    new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
}
```



6. fejezet

Helló, Welch!

6.1. Első osztályom

Tutor: Ignéczi Tíbor, a kód tőle származik.

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

C++:

```
class PolarGen
{
public:
    PolarGen()
    {
        nincsTarolt = true;
        std::srand (std::time(NULL));
    }
    ~PolarGen()
    {
    }
    double kovetkezo();
private:
    bool nincsTarolt;
    double tarolt;
};
double PolarGen::kovetkezo ()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1= std::rand() / (RAND_MAX +1.0);
            u2= std::rand() / (RAND_MAX +1.0);
```



```
        v1=2*u1-1;
        v2=2*u1-1;
        w=v1*v1+v2*v2;
    }
    while (w>1);
    double r =std::sqrt ((-2 * std::log(w)) /w);
    tarolt=r*v2;
    nincsTarolt =!nincsTarolt;
    return r* v1;
}
else
{
    nincsTarolt =!nincsTarolt;
    return tarolt;
}
}
int main (int argc, char **argv)
{
    PolarGen pg;
    for (int i= 0; i<10;++i)
        std::cout<<pg.kovetkezo ()<< std::endl;
    return 0;
}
```

Ez a program véletlenszerűen fog számokat generálni nekünk. Ezt egy osztályon belül fogjuk kivitelezni. Az osztályunk neve a PolarGen-t kapta. Két részre tudjuk most bontani. Van egy nyilvános és egy privát rész. A nyilvánoshoz hozzá tudunk férni viszont a privátban, csak az osztályon belül tudjuk meghívni a benne lévő változókat és függvényeket. Az osztály elején egyből ott van a konstruktor ezt onnan tudjuk felismerni, hogy ugyan úgy hívjuk ahogyan az osztályt is. ebben kezdő értékeket tudunk adni és egy objektum létrehozásával egyből lefut. Esetünkben most a "nincsTarolt" privát változó értékét fogja "True"-ra állítani és az srand is itt lesz ami a véletlen szám generálásához kell. Utánna van a destruktork ami ugyan úgy néz ki mint a konstruktor csak előtte van '~' jel. Ez a program végén fog lefutni. Ebbe szoktak felszabadítani a memóriát. A privát részben létrehozunk egy logikai és egy double típusú változót. A kovetkezo() függvény az amiben a random számokat fogjuk létrehozni. Azt hogy ezt hogyan végezzük matematikailag, azt most figyelmen kívül hagyjuk. A main függvényben meghívunk egy osztálytípusú változót. Ez fogja beindítani a konstruktort. Utánna pedig egy for ciklusban tíz véletlen száot íratunk ki.

Java:

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

    public PolarGenerator()
    {
        nincsTarolt = true;
    }

    public double kovetkezo()
    {
```

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

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

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

Ez az előző program csak javában megírva. A program felépítése sokkal átláthatóbb és egyszerűbb lett. Alapjaiban ugyan úgy működik mint a C++ megfelelője

6.2. LZW

A program a bemeneti adatokból egy bináris fát épít. Bináris fa, ha mindegyik csomópontnak maximum 2 gyermeke van. (2 elágazása). A fa 0 és 1 számokból épül fel. A kitüntetett elem a gyökér. Innen minden elemet el tudunk érni. A következőben megnézzük, hogy is működik ez. Az eltérés, hogy itt nem fő függvény van, hanem minden egy osztály része.

A kód forrása: https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat?p5MQomtcQIdfTeZvPInhgRxu-CCsxGOx453MSrGk

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

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

typedef struct binfa
{
    int ertek;
    struct binfa *bal_nulla;
    struct binfa *jobb_egy;
} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

    if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
        perror ("memoria");
        exit (EXIT_FAILURE);
    }
    return p;
}

extern void kiir (BINFA_PTR elem);
extern void ratlag (BINFA_PTR elem);
extern void rszoras (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);

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

    BINFA_PTR gyoker = uj_elem ();
    gyoker->ertek = '/';
    gyoker->bal_nulla = gyoker->jobb_egy = NULL;
    BINFA_PTR fa = gyoker;

    while (read (0, (void *) &b, 1))
    {
        if (b == '0')
        {
            if (fa->bal_nulla == NULL)
            {
                fa->bal_nulla = uj_elem ();
                fa->bal_nulla->ertek = 0;
                fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
                fa = gyoker;
            }
        }
    }
}
```

```
        else
        {
            fa = fa->bal_nulla;
        }
    }
    else
    {
        if (fa->jobb_egy == NULL)
        {
            fa->jobb_egy = uj_elem ();
            fa->jobb_egy->ertek = 1;
            fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
            fa = gyoker;
        }
        else
        {
            fa = fa->jobb_egy;
        }
    }
}

printf ("\n");
kiir (gyoker);

extern int max_melyseg, atlagosszeg, melyseg, atlagdb;
extern double szorasosszeg, atlag;

printf ("melyseg=%d\n", max_melyseg-1);

atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
ratlag (gyoker);
atlag = ((double)atlagosszeg) / atlagdb;

atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
szorasosszeg = 0.0;

rszoras (gyoker);

double szoras = 0.0;

if (atlagdb - 1 > 0)
    szoras = sqrt( szorasosszeg / (atlagdb - 1));
else
    szoras = sqrt (szorasosszeg);

printf ("atlag=%f\nszoras=%f\n", atlag, szoras);*/
```

```
    szabadit (gyoker);
}

int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

void
ratlag (BINFA_PTR fa)
{
    if (fa != NULL)
    {
        ++melyseg;
        ratlag (fa->jobb_egy);
        ratlag (fa->bal_nulla);
        --melyseg;

        if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

double szorasosszeg = 0.0, atlag = 0.0;

void
rszoras (BINFA_PTR fa)
{
    if (fa != NULL)
    {
        ++melyseg;
        rszoras (fa->jobb_egy);
        rszoras (fa->bal_nulla);
        --melyseg;

        if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

int max_melyseg = 0;

void
```

```
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
        kiir (elem->jobb_egy);
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
            ,
            melyseg-1);
        kiir (elem->bal_nulla);
        --melyseg;
    }
}

void
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal_nulla);
        free (elem);
    }
}
```

Magyarázat: Elsőnek is a szükséges headereket inculdeáljuk. Ezek után deklaráljuk a Binfánk struktúráját. A struktúra egy egészet tartalmaz aminek a neve "ertek" és 2 mutatóval fog rendelkezni, amikben bal és jobb gyermekeket tároljuk majd. Az 1 érték jobbra fog kerülni, még a 0 balra. A "typedef" el adunk neki egy nevet, amivel a programon belül fogjuk hívni. Ezután az uj_elem függvény következik. ez fogja nekünk lefoglalni a tárhelyet a memóriában amit "NULL" kezdőértékkel fog rendelkezni. Ha nincs memória, akkor hibát dob ki. A végén vissza adja a lefoglalt mutatót. Utánna függvény prototipusokat kapunk, ezek közül a feladatnak megfelelően csak a kiir() és a szabadit() függvényeket fogjuk megvizsgálni. Ugorjunk a main fő függvényre. AZ első egy char típusú változó, ebben fogjuk tárolni ideiglenesen a beolvasott karaktert. Aztán létrehozuk a gyökérelemet és értékül adunk neki egy karaktert, jelen esetben '/'. A while ciklusban fog zajlani a faépítés. Először is megvizsgálja a beolvasott karakter. Mindig a gyökér elemtől indul. Ha a beolvasott karakter értéke 0, akkor először megvizsgálja, hogy a gyökérnek vagy az adott csomópontnak van e bal_nullas gyermeke, ha van, akkor rálép a csomópontra, ha viszont nincs akkor a gyökérnek vagy az adott csomópontnak létrehoz egy bal_nullas gyermeket. Ha a beolvasott karakter értéke 1, akkor a program ugyan ezen az elven mint a 0 -ás értéknél végig vizsgálja csak a jobb_egyest gyermekkel. Most következik a kiir és a szabadit függvény. A szabadit() függvény egy rekurzív függvény. Törli a memóriából az eltárolt elemeket. A kiir() függvény is rekurzív függvény. Bejárja a fa elemeit.

6.3. Fabejárás

Tutor:Földesi Zoltán

A fabejárásnak 3 típusa van, preorder, inorder és postorder. Azt hogy hogyan járja be a fát már a neve is rejti. Kezdjük a preorder fabejárással. Itt mindig a gyökérrel kezdi a program a vizsgálatot ó, aztán a bal oldalt járja be és legvégezetül pedig a jobb oldalt fogja bejárni. Az

Kezdjük a preorder fabejárással. Itt mindig a gyökérrel kezdi a program a vizsgálatot ó, aztán a bal oldalt járja be és legvégezetül pedig a jobb oldalt fogja bejárni.

```
//preorder fabejárás:
void kiir(BINFA_PTR elem)
{
    if (elem !=NULL)
    {
        ++melyseg
        if (melyseg>max_melyseg)
            max melyseg = melyseg;
        for (int i=0; i<melyseg;i++)
            printf("----");
        printf("%c(%d)\n",elem->ertek<2 ? '0' + elem->ertek : elem-> ←
            ertek, melyseg-1);
        kiir(elem->bal_nulla);
        kiir(elem->jobb_egy);
        --melyseg
    }
}
```

Inorder fabejárás: Az inorder fabejárásnál először a bal oldalt vizsgáljuk meg, utánna jön a gyökér és legvégül pedig a jobb oldalt nézzük. Erre példa az előző programban lévő kiir függvény, ahol inorder fabejárás van.

A postorder fabejárás: Itt először a fa bal oldalát fogja átvizsgálni aztán a jobb oldalt és legutoljára pedig a gyökeret vizsgáljuk.

```
//postorder fabejárás:
void kiir(BINFA_PTR elem)
{
    if (elem !=NULL)
    {
        ++melyseg
        if (melyseg>max_melyseg)
            max melyseg = melyseg;
        kiir(elem->bal_nulla);
        kiir(elem->jobb_egy);
        for (int i=0; i<melyseg;i++)
            printf("----");
        printf("%c(%d)\n",elem->ertek<2 ? '0' + elem->ertek : elem-> ←
            ertek, melyseg-1);
        --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: https://progater.blog.hu/2011/03/31/imadni_fogjatok_a_c_t_egy_emberkent_tiszta_szivbol

Az alábbi program a fenti C változatnak lesz úgymond a C++ változata. Az első lépés, hogy ami C-ben struktúra volt, azt átírjuk C++-ban egy osztályba, mivel a C++ -ban megtehetjük. Ez az alábbi módon fog kinézni:

```
class LZWBinFa
{
public:
    LZWBinFa (char b = '/'):betu (b), balNulla (NULL), jobbEgy (NULL) ←
    {};
    ~LZWBinFa () {};
    void operator<<(char b)
{
    if (b == '0')
    {
        // van '0'-s gyermeke az aktuális csomópontnak?
        if (!fa->nullasGyermeke ()) // ha nincs, csinálunk
        {
            Csomopont *uj = new Csomopont ('0');
            fa->ujNullasGyermeke (uj);
            fa = &gyoker;
        }
        else // ha van, arra lépünk
        {
            fa = fa->nullasGyermeke ();
        }
    }
    else
    {
        if (!fa->egyenesGyermeke ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyenesGyermeke (uj);
            fa = &gyoker;
        }
        else
        {
            fa = fa->egyenesGyermeke ();
        }
    }
}
```


Ezen belül fogjuk a gyökeret létrehozni és értékül adni neki a '/'-t. A mutatóinak értékét 0-ra állítjuk. Ezek után jön a faépítés a már fentiekben elmagyarázott módon. A program megnézi, hogy 1-est vagy 0 érkezik a bemenetről. itt azt látjuk, hogy a betétel a << operátorral történik, ez annyiban különbözik a C-ben írt programtól, hogy ez egyből beleteszi a fába a beérkezett karaktert. Egy új csomópontot a "new" szóval tudunk létrehozni, ha szükséges. Ez azért lehetséges mert van egy Csomópont osztályunk (Lásd a lenti programban). A Class csomóponton belül az egyesGyermek() és a nullasGyermek() függvények a gyermekükre mutató pointereket fogják tartalmazni. Az ujNullasGyermek és au ujEgyesGyermek-nek pedig adunk egy gyermeket és arra fogja állítani a mutatót. A private részben fogjuk ezeket deklarálni, ez azt jelenti, hogy csak az osztályon belül használhatóak ezek a változók. A legvégén jön a main főfüggvény. Itt deklaráljuk a char típusú változót amibe beolvasunk és innen kerül az osztályokhoz. Végül meghívjuk a kiir és a szabadit függvényeket amire példát az előző programokban találunk. Ugye a kiir()-al kiíratjuk az eredmény és a szabadit()-al pedig felszabadítjuk a lefoglalt memóriát.

```
class Csomopont
{
public:
    Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
    ~Csomopont () {};
    Csomopont *nullasGyermek () {
        return balNulla;
    }
    Csomopont *egyesGyermek ()
    {
        return jobbEgy;
    }
    void ujNullasGyermek (Csomopont * gy)
    {
        balNulla = gy;
    }
    void ujEgyesGyermek (Csomopont * gy)
    {
        jobbEgy = gy;
    }
private:
    friend class LZWBinFa;
    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &);
    Csomopont & operator=(const Csomopont &);
};

int main ()
{
    char b;
    LZWBinFa binFa;
    while (std::cin >> b)
    {
```

```
        binFa << b;
    }
    binFa.kiir ();
    binFa.szabadit ();
    return 0;
}
```

6.5. Mutató a gyökér

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

Vegyük alapul a C++ ban megírt LZWBinf-t. első dolgunk, hogy a fában a gyökér elemet átalakítjuk egy mutatóvá. Azt az alábbi módon fogjuk megcsinálni: A gyökér elem a protected részén van az osztálynak. Itt az eredeti "Csomopont gyoker;" az alábbi módon átírunk:

```
protected:
    Csomopont *gyoker;
    int maxMelyseg;
    double atlag, szoras;
```

Ugye C++ ban a mutatót egy '*'-al jelöljük. Ha most futtatnánk a programot, akkor számtalan hibába ütköznénk. Ezeket ki kell javítanunk. A programban így nem a gyökér memóriacímét kell átadnunk (Töröljük az összes referenciajelet a gyökerek előtt) és mivel mutató lett a gyökér így nem '.'-al hiavatkozunk hanem '->'-al. Itt láthatunk példát arra, hogy hogyan:

```
//előtte:
fa=&gyoker;
//utánna:
fa=gyoker;

//előtte:
szabadit (gyoker.egyenesGyermekek ());
szabadit (gyoker.nullasGyermekek ());
//utánna:
szabadit (gyoker->egyenesGyermekek ());
szabadit (gyoker->nullasGyermekek ());

}
```

Ha mindezek után lefuttatjuk a programunkat az lefordul, azonban futtatáskor szegmentális hibába ütkö-zünk. Ez azért van, ugyanis a gyökér memóriacíme nincs lefoglalva. Ennek a megoldását a konstruktorban és a destruktorban fogjuk megalkotni. A konstruktorban foglaljuk le és a destruktorba fogjuk törölni a lefoglalt memóriát. Lásd:

```
LZWBinFa ()
{
    gyoker= new Csomopont ('/');
    fa = gyoker;
}
~LZWBinFa ()
{
    szabadit (gyoker->egyesGyermekek ());
    szabadit (gyoker->nullasGyermekek ());
    delete(gyoker);
}
```

6.6. Mozgató szemantika

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

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

Ebben a feladatban egy QT programban fogjuk szimulálni a hangyák mozgását. A való életben a hangyák sose zavarodnak össze, sose torlódnak fel, sőt minél többen vannak annál jobban és gyorsabban mozognak. Továbbá tudjuk, hogy a látásuk nem a legjobb. Tehát a kulcs az egymás közötti kommunikáció, ezt feromonokkal érik el. Ebben a szimulációban mi is úgymond feromonokkal fogjuk a hangyák közötti kommunikációt elérni. A szabály, hogy mindig a legerősebb feromonú hangya felé lépünk, a programban látszik, hogy a hangyák feromon csíkot hagynak maguk után, ami idő elteltével egyre gyengül míg el nem tűnik.

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

Kód forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Myrmecologist

Kód:(csak a main.cpp van itt a többi kód a forrásnál található)

```
#include <QApplication>
#include <QDesktopWidget>
#include <QDebug>
#include <QDateTime>
#include <QCommandLineOption>
#include <QCommandLineParser>

#include "antwin.h"

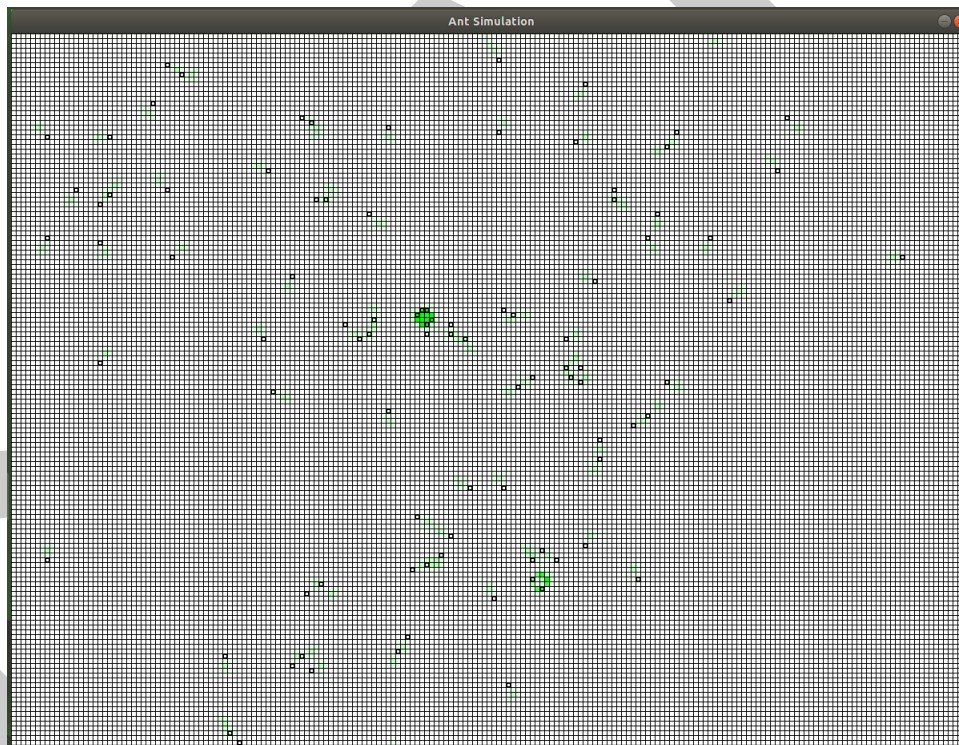
int main ( int argc, char *argv[] )
{
    QApplication a ( argc, argv );

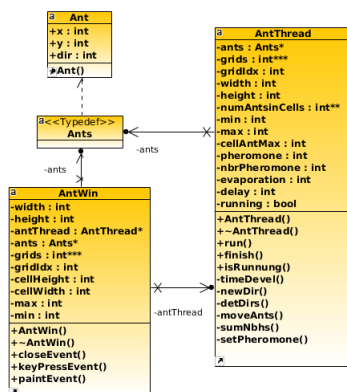
    QCommandLineOption szeles_opt ( {"w", "szelesseg"}, "Oszlopok (cellakban ←
        ) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( {"m", "magassag"}, "Sorok (cellakban) ←
        szama.", "magassag", "150" );
```

```
QCommandLineOption hangyaszam_opt ( {"n","hangyaszam"}, "Hangyak szama. ↵  
", "hangyaszam", "100" );  
QCommandLineOption sebesseg_opt ( {"t","sebesseg"}, "2 lepes kozotti ↵  
ido (millisec-ben).", "sebesseg", "100" );  
QCommandLineOption parolgas_opt ( {"p","parolgas"}, "A parolgas erteke. ↵  
", "parolgas", "8" );  
QCommandLineOption feromon_opt ( {"f","feromon"}, "A hagyott nyom ↵  
erteke.", "feromon", "11" );  
QCommandLineOption szomszed_opt ( {"s","szomszed"}, "A hagyott nyom ↵  
erteke a szomszedokban.", "szomszed", "3" );  
QCommandLineOption alapertek_opt ( {"d","alapertek"}, "Indulo ertek a ↵  
cellakban.", "alapertek", "1" );  
QCommandLineOption maxcella_opt ( {"a","maxcella"}, "Cella max erteke." ↵  
, "maxcella", "50" );  
QCommandLineOption mincella_opt ( {"i","mincella"}, "Cella min erteke." ↵  
, "mincella", "2" );  
QCommandLineOption cellamerete_opt ( {"c","cellameret"}, "Hany hangya ↵  
fer egy cellaba.", "cellameret", "4" );  
QCommandLineParser parser;  
  
parser.addHelpOption();  
parser.addVersionOption();  
parser.addOption ( szeles_opt );  
parser.addOption ( magas_opt );  
parser.addOption ( hangyaszam_opt );  
parser.addOption ( sebesseg_opt );  
parser.addOption ( parolgas_opt );  
parser.addOption ( feromon_opt );  
parser.addOption ( szomszed_opt );  
parser.addOption ( alapertek_opt );  
parser.addOption ( maxcella_opt );  
parser.addOption ( mincella_opt );  
parser.addOption ( cellamerete_opt );  
  
parser.process ( a );  
  
QString szeles = parser.value ( szeles_opt );  
QString magas = parser.value ( magas_opt );  
QString n = parser.value ( hangyaszam_opt );  
QString t = parser.value ( sebesseg_opt );  
QString parolgas = parser.value ( parolgas_opt );  
QString feromon = parser.value ( feromon_opt );  
QString szomszed = parser.value ( szomszed_opt );  
QString alapertek = parser.value ( alapertek_opt );  
QString maxcella = parser.value ( maxcella_opt );  
QString mincella = parser.value ( mincella_opt );  
QString cellameret = parser.value ( cellamerete_opt );  
  
qsrand ( QDateTime::currentMSecsSinceEpoch() );
```

```
AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), feromon ←  
    .toInt(), szomszed.toInt(), parolgas.toInt(),  
        alapertek.toInt(), mincella.toInt(), maxcella.toInt(),  
        cellameret.toInt() );  
  
w.show();  
  
return a.exec();  
}  
}
```

Kezdjük az ant.h tartalmazza a hangya tulajdonságait. Hol van az x és y tengelyen és hogy merre mutat az iránya, merre megy. Az antwin-ban van pedig a hangyaboly(ants). Továbbá ezen belül adjuk meg, hogy egy cella hány pixelből álljon és hogy az ablak szélessége és magassága hány. Forciklus-okkal felépítjük cellákból az ablakot és elhelyezzük benne a hangyákat(azt ant-ből). Új és új hangyák jelennek meg. Ezek megváltoztatják a régebbi hangyák irányát. Az antwin.h tartalmazza a billentyűzet parancsait, például a p vel megállítjuk a folyamatot. Az antheard.cpp tartalmazza a mozgáshoz, törléshez, az új irány megadásához, a hangyák számának eltárolásához szükséges függvényeket. Itt vizsgálja azt is hogy a hangyák száma nő e vagy csökken az idő múlásával.





A kép, a megoldás videóból származik.

7.2. Java életjáték

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

Az életjátékt azaz sejtautómatákat először Naumen János vetette fel. A felvetés a gép önreprodukciójának matematikai modellalkotást tartalmazta. A legismertebb modell a John Horton Conway-féle életjáték. Maga a "játék" egy négyzetrácsos mezőn zajlik amin mozognak a sejtek. A sejtek "élete" szabályokhoz van kötve. Megvan adva hogy mi a feltétele hogy egy sejt létrejöjjön, életbenmadardjon vagy elpusztuljon. Conway erre 3 feltételt szabott meg:

- 1.szabály: Egy sejt csak úgy éli túl, ha kettő vagy három szomszédja van.
- 2.szabály: Egy sejt akkor pusztul el, ha kettőnél kevesebb szomszédja van. Ezt elszigetelődésnek hívjuk. A másik eset hogy akkor pusztul el ha háromnál több szomszédja van. Ezt túlnépesedésnek hívjuk.
- 3.szabály: A harmadik szabály a születésre vonatkozik, és akkor történik meg ha egy cellának a körzetében 3 sejt található.

Ezen a 3 szabály meghatározásával kapunk egy önműködő sejtautómatát. Beleszolásunk csak kezdetben van, utána a szabályok szerint önállóan működik a program. Mi most külön a sikló-kilövőt fogjuk vizsgálni. Hogy ezt elérjük, rögzítenünk kell adott cellákban sejteket, így létre jön egy "sikló ágyú", ez időközönként "siklókat" fog lőni.

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apb.html?fbclid=IwAR0Gc-Q7v3531_qDrqAd4LflhWrzTwYnnsTZ5wTpBAhQjwZ63pl2moebOpY

Kód:

```

public class Sejtautomata extends java.awt.Frame implements Runnable {
    public static final boolean ÉLŐ = true;
    public static final boolean HALOTT = false;
    protected boolean [][][] rácsek = new boolean [2][][];
    protected boolean [][] rác;
    protected int rácIndex = 0;
    protected int cellaSzélesség = 20;
    protected int cellaMagasság = 20;
    protected int szélesség = 20;
    protected int magasság = 10;
}
  
```

```
protected int várakozás = 1000;
private java.awt.Robot robot;
private boolean pillanatfelvétel = false;
private static int pillanatfelvételSzámláló = 0;
public Sejtautomata(int szélesség, int magasság) {
    this.szélesség = szélesség;
    this.magasság = magasság;
    rácsok[0] = new boolean[magasság][szélesség];
    rácsok[1] = new boolean[magasság][szélesség];
    rácsIndex = 0;
    rács = rácsok[rácsIndex];
    for(int i=0; i<rács.length; ++i)
        for(int j=0; j<rács[0].length; ++j)
            rács[i][j] = HALOTT;
    siklóKilövő(rács, 5, 60);
    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
            setVisible(false);
            System.exit(0);
        }
    });
    addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyPressed(java.awt.event.KeyEvent e) {
            if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
                cellaSzélesség /= 2;
                cellaMagasság /= 2;
                setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
                validate();
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                cellaSzélesség *= 2;
                cellaMagasság *= 2;
                setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
                validate();
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
                pillanatfelvétel = !pillanatfelvétel;
            else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
                várakozás /= 2;
            else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
                várakozás *= 2;
            repaint();
        }
    });
    addMouseListener(new java.awt.event.MouseAdapter() {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    })
}
```



```
});  
addMouseListener(new java.awt.event.MouseMotionAdapter() {  
    // Vonszolással jelöljük ki a négyzetet:  
    public void mouseDragged(java.awt.event.MouseEvent m) {  
        int x = m.getX()/cellaSzélesség;  
        int y = m.getY()/cellaMagasság;  
        rácsok[rácsIndex][y][x] = ÉLŐ;  
        repaint();  
    }  
});  
cellaSzélesség = 10;  
cellaMagasság = 10;  
try {  
    robot = new java.awt.Robot(  
        java.awt.GraphicsEnvironment.  
            getLocalGraphicsEnvironment().  
            getDefaultScreenDevice());  
} catch(java.awt.AWTException e) {  
    e.printStackTrace();  
}  
setTitle("Sejtautomata");  
setResizable(false);  
setSize(szélesség*cellaSzélesség,  
        magasság*cellaMagasság);  
setVisible(true);  
new Thread(this).start();  
}  
public void paint(java.awt.Graphics g) {  
    boolean [][] rács = rácsok[rácsIndex];  
    for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon  
        for(int j=0; j<rács[0].length; ++j) { // s az oszlopok  
            if(rács[i][j] == ÉLŐ)  
                g.setColor(java.awt.Color.BLACK);  
            else  
                g.setColor(java.awt.Color.WHITE);  
            g.fillRect(j*cellaSzélesség, i*cellaMagasság,  
                cellaSzélesség, cellaMagasság);  
            g.setColor(java.awt.Color.LIGHT_GRAY);  
            g.drawRect(j*cellaSzélesség, i*cellaMagasság,  
                cellaSzélesség, cellaMagasság);  
        }  
    }  
}  
if(pillanatfelvétel) {  
    pillanatfelvétel = false;  
    pillanatfelvétel(robot.createScreenCapture  
        (new java.awt.Rectangle  
            (getLocation().x, getLocation().y,  
            szélesség*cellaSzélesség,  
            magasság*cellaMagasság)));  
}
```

```
}
public int szomszédokSzama(boolean [][] rács,
    int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 0;
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            if(!((i==0) && (j==0))) {
                int o = oszlop + j;
                if(o < 0)
                    o = szélesség-1;
                else if(o >= szélesség)
                    o = 0;

                int s = sor + i;
                if(s < 0)
                    s = magasság-1;
                else if(s >= magasság)
                    s = 0;

                if(rács[s][o] == állapot)
                    ++állapotúSzomszéd;
            }

    return állapotúSzomszéd;
}

public void időFejlődés() {

    boolean [][] rácsElőtte = rácsok[rácsIndex];
    boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];

    for(int i=0; i<rácsElőtte.length; ++i) { // sorok
        for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok

            int élők = szomszédokSzama(rácsElőtte, i, j, ÉLŐ);

            if(rácsElőtte[i][j] == ÉLŐ) {
                if(élők==2 || élők==3)
                    rácsUtána[i][j] = ÉLŐ;
                else
                    rácsUtána[i][j] = HALOTT;
            } else {
                if(élők==3)
                    rácsUtána[i][j] = ÉLŐ;
                else
                    rácsUtána[i][j] = HALOTT;
            }
        }
    }

    rácsIndex = (rácsIndex+1)%2;
}
```

```
public void run() {  
  
    while(true) {  
        try {  
            Thread.sleep(várákozás);  
        } catch (InterruptedException e) {}  
  
        időFejlődés();  
        repaint();  
    }  
}  
public void sikló(boolean [][] rács, int x, int y) {  
  
    rács[y+ 0][x+ 2] = ÉLŐ;  
    rács[y+ 1][x+ 1] = ÉLŐ;  
    rács[y+ 2][x+ 1] = ÉLŐ;  
    rács[y+ 2][x+ 2] = ÉLŐ;  
    rács[y+ 2][x+ 3] = ÉLŐ;  
  
}  
public void siklóKilövő(boolean [][] rács, int x, int y) {  
  
    rács[y+ 6][x+ 0] = ÉLŐ;  
    rács[y+ 6][x+ 1] = ÉLŐ;  
    rács[y+ 7][x+ 0] = ÉLŐ;  
    rács[y+ 7][x+ 1] = ÉLŐ;  
  
    rács[y+ 3][x+ 13] = ÉLŐ;  
  
    rács[y+ 4][x+ 12] = ÉLŐ;  
    rács[y+ 4][x+ 14] = ÉLŐ;  
  
    rács[y+ 5][x+ 11] = ÉLŐ;  
    rács[y+ 5][x+ 15] = ÉLŐ;  
    rács[y+ 5][x+ 16] = ÉLŐ;  
    rács[y+ 5][x+ 25] = ÉLŐ;  
  
    rács[y+ 6][x+ 11] = ÉLŐ;  
    rács[y+ 6][x+ 15] = ÉLŐ;  
    rács[y+ 6][x+ 16] = ÉLŐ;  
    rács[y+ 6][x+ 22] = ÉLŐ;  
    rács[y+ 6][x+ 23] = ÉLŐ;  
    rács[y+ 6][x+ 24] = ÉLŐ;  
    rács[y+ 6][x+ 25] = ÉLŐ;  
  
    rács[y+ 7][x+ 11] = ÉLŐ;  
    rács[y+ 7][x+ 15] = ÉLŐ;  
    rács[y+ 7][x+ 16] = ÉLŐ;  
    rács[y+ 7][x+ 21] = ÉLŐ;  
    rács[y+ 7][x+ 22] = ÉLŐ;
```

```
rács[y+ 7][x+ 23] = ÉLŐ;
rács[y+ 7][x+ 24] = ÉLŐ;

rács[y+ 8][x+ 12] = ÉLŐ;
rács[y+ 8][x+ 14] = ÉLŐ;
rács[y+ 8][x+ 21] = ÉLŐ;
rács[y+ 8][x+ 24] = ÉLŐ;
rács[y+ 8][x+ 34] = ÉLŐ;
rács[y+ 8][x+ 35] = ÉLŐ;

rács[y+ 9][x+ 13] = ÉLŐ;
rács[y+ 9][x+ 21] = ÉLŐ;
rács[y+ 9][x+ 22] = ÉLŐ;
rács[y+ 9][x+ 23] = ÉLŐ;
rács[y+ 9][x+ 24] = ÉLŐ;
rács[y+ 9][x+ 34] = ÉLŐ;
rács[y+ 9][x+ 35] = ÉLŐ;

rács[y+ 10][x+ 22] = ÉLŐ;
rács[y+ 10][x+ 23] = ÉLŐ;
rács[y+ 10][x+ 24] = ÉLŐ;
rács[y+ 10][x+ 25] = ÉLŐ;

rács[y+ 11][x+ 25] = ÉLŐ;

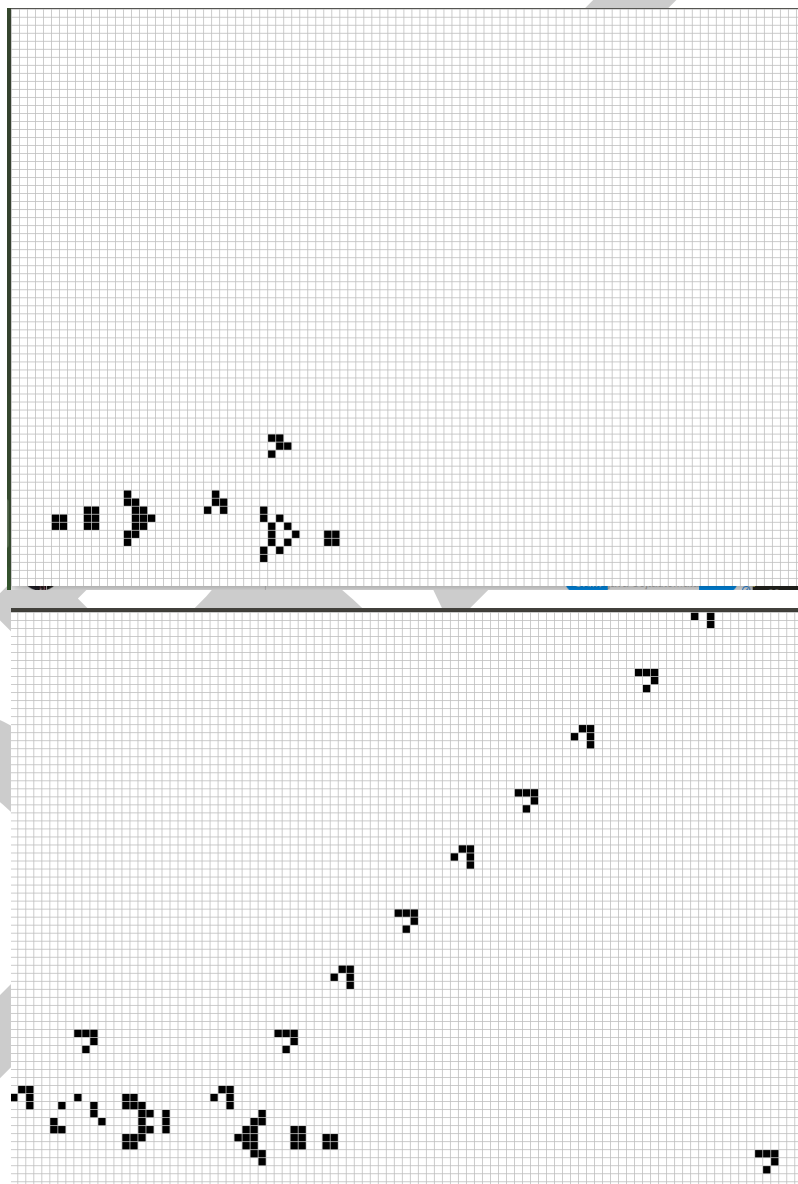
}

public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    // A pillanatfelvétel kép fájlneve
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételSzámláló);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
}

public void update(java.awt.Graphics g) {
    paint(g);
}

public static void main(String[] args) {
    new Sejtautomata(100, 75);
}
}
```

A program elején megadjuk, hogy egy sejt lehet élő vagy halott. A feladatban 2 rácsfelét használunk, az egyik rács a sejt állapotát fogja tárolni míg a második az egy másdopercel későbbi tulajdonságait. Meghatározzuk az aktuális rácsot a `rácsIndex`-el. Utánna pedig egy cella magasságát és szélességét, ezt követően pedig, hogy hány cellából álljon a "játék". A következő hogy a az állapotok között mennyi idő teljen el. A függvények közül az első függvény megkapja a méreteket és létrehozza az ablakot. Itt készíti el a 2 rácsot is és az indexet is elindítja. Kezdetben minden rács HALOTT. Ezen belül lesz meghívva a siklólovó aminek a kód végén minden kordinátája megvan adva. Vannak billentyűről beérkezőparancsaink is, különböző feladatokkal ellátva pl a "g" betűvel, a két állapot közötti időt csökkentjük. Ugyan így vannak az egérrel történő információk feldolgozására szolgáló függvények. Külön a kattintásra és a mozgásra. Külön tudunk készíteni pillanatfelvételt az aktuális állapotról az "s" gomb segítségével. A programban a sejtér rajzolását a `paint()` függvénnyel végezzük. A `szomszédokSzama()` függvényben vizsgáljuk a szabályokat és aszerint történik a sejtek viselkedése.



7.3. Qt C++ életjáték

Most alkossuk meg az életjátékot Qt C++-ban is. A program lényege itt is ugyan az mint a java-ban, ugyan azok a szabályok. Ha kettő vagy gárom szomszédja van, akkor életben marad, ha 2 nél kevesebb kipustul, ha 3 nál több, akkor túlnépesedés miatt pusztul ki. Itt is a siklóágyú lesz a fő célunk.

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

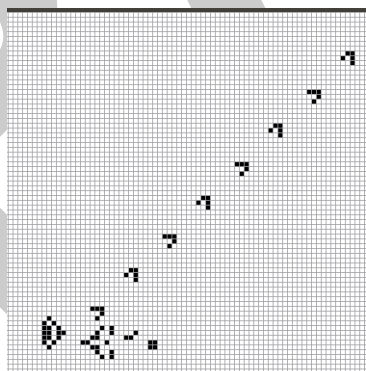
Kód:(ez csak a main.cpp a többi szükséges fájl a forrásnál található.)

```
#include <QApplication>
#include "sejtablak.h"
#include <QDesktopWidget>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    SejtAblak w(100, 75);
    w.show();

    return a.exec();
}
```

A forrásfájljaink a Sejtablak.cpp , sejszal.h, sejszal.cpp és a sejtablak.h . Ezek az átláthatóság miatt vannak külön. A sejtablak.h és .cpp tartalmazza a függvényeket amivel majd a kirajzolás fog történni és ebben van a sikló lövés is, úgy mint a java-s párjánál, külön minden egyes cellát megadunk amiben sejt van. A sejszal.h és .c pedig az életjátékhoz szükséges szabályokat . Ezen belül vannak a függvények melyek az adott állapotokat vizsgálják és a szabályok szerint alakítják a programot.



7.4. BrainB Benchmark

Tutorált: Földesi Zoltán

Elsőnek is a Benchmark jelentését nézzük meg. A benchmark egy elemzés, tesztfeladat. Egy bizonyos tesztet végez el és azt az elért pontszám alapján összehasonlítja a tesztet elvégzők között. Ilyen például telefonok teljesítményét végző benchmark , vagy az esetünkben az agy teljesítményét vizsgáló Benchmark. Ennek segítségével tudunk egy viszonyítási alapot venni egy adott feladatban. Például a telefonoknál, hogy minél több pontot ér el annál jobb a teljesítménye és össze tudjuk hasonlítani más telefonok teljesítményével. Ugyan így a BrainBenchmarkban, a tesztet megoldó emberek közül az adott pontszám megadja

hogy ki teljesített a legjobban és egymáshoz is tudjuk viszonyítani őket. Az adott programunk az egyének figyelemképpességét és koncentrációját fogja vizsgálni. Adott egy karakter, ami a mi karakterünk és azon kell tartanunk az egér kurzort. A program azt vizsgálja, hogy mennyi ideig vagyunk képesek a kurzort a mi karakterünkön tartani, azaz meddig nem veszítjük el azt. Persze nem ilyen egyszerű, mert közben rengeteg új karakter jelenik meg a monitoron befolyásolva ezzel minket, hogy el ne veszítsük a karakterünket. A program arra is reagál, ha elveszítjük a karakterünk.

Az adott programban a mi karakterünk Samu lesz. Samut figyelemmel kell tartani. Ahogy fent említettem ezt a kurzorral fogjuk megtenni. Minél tovább tartjuk Samun a kurzort annál több másik karakter lesz a képernyőn, A feladat 10 percig tart és annál jobb vagy ha minél több kis karakter között is megtudod tartani a saját karaktered. Ha elveszítenéd abban az esetben belassul az új karakterek megjelenése míg meg nem találod. Annál jobban teljesítettél, minél több pontod van a 10 perc végén.

Megoldás forrása: <https://github.com/nbatfai/esport-talent-search>

Kód:

```
eturn a.exec();
#include <QApplication>
#include <QTextStream>
#include <QtWidgets>
#include "BrainBWin.h"

int main ( int argc, char **argv )
{
    QApplication app ( argc, argv );

    QTextStream qout ( stdout );
    qout.setCodec ( "UTF-8" );

    qout << "\n" << BrainBWin::appName << QString::fromUtf8 ( " ←
    Copyright (C) 2017, 2018 Norbert Bátfai" ) << endl;

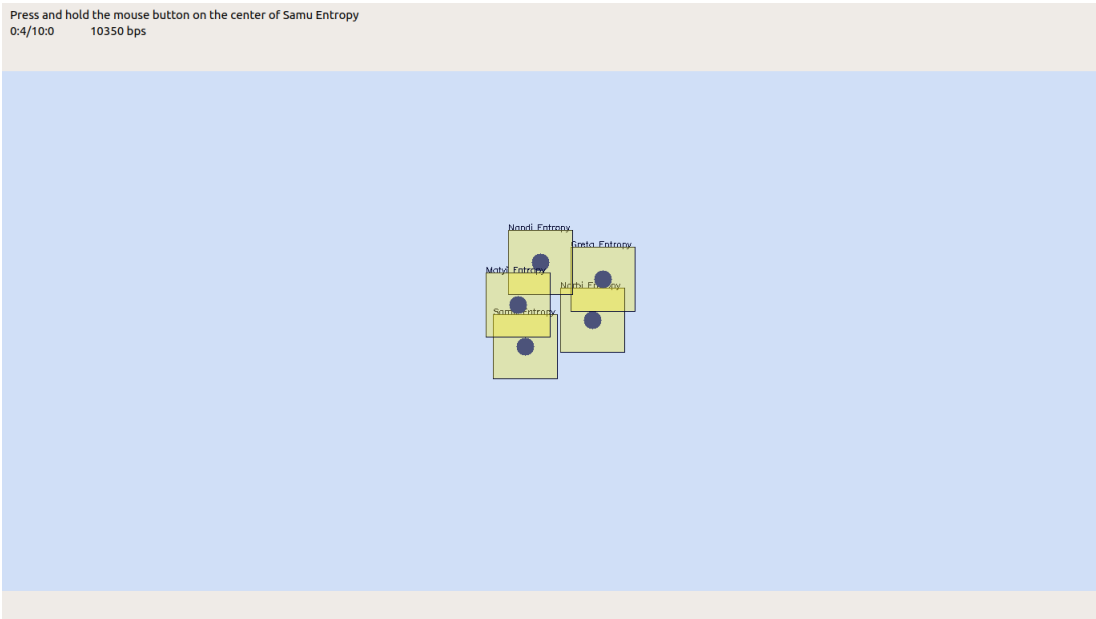
    qout << "This program is free software: you can redistribute it and ←
    /or modify it under" << endl;
    qout << "the terms of the GNU General Public License as published ←
    by the Free Software" << endl;
    qout << "Foundation, either version 3 of the License, or (at your ←
    option) any later" << endl;
    qout << "version.\n" << endl;

    qout << "This program is distributed in the hope that it will be ←
    useful, but WITHOUT" << endl;
    qout << "ANY WARRANTY; without even the implied warranty of ←
    MERCHANTABILITY or FITNESS" << endl;
    qout << "FOR A PARTICULAR PURPOSE. See the GNU General Public ←
    License for more details.\n" << endl;

    qout << QString::fromUtf8 ( "Ez a program szabad szoftver; ←
    terjeszthető illetve módosítható a Free Software" ) << endl;
    qout << QString::fromUtf8 ( "Foundation által kiadott GNU General ←
    Public License dokumentumában leírtak;" ) << endl;
```

```
qout << QString::fromUtf8 ( "akár a licenc 3-as, akár (tetszőleges) ←  
    későbbi változata szerint.\n" ) << endl;  
  
qout << QString::fromUtf8 ( "Ez a program abban a reményben kerül ←  
    közreadásra, hogy hasznos lesz, de minden" ) << endl;  
qout << QString::fromUtf8 ( "egyéb GARANCIA NÉLKÜL, az ←  
    ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ" ) << endl;  
qout << QString::fromUtf8 ( "ALKALMAZHATÓSÁGRA való származtatott ←  
    garanciát is beleértve. További" ) << endl;  
qout << QString::fromUtf8 ( "részleteket a GNU General Public ←  
    License tartalmaz.\n" ) << endl;  
  
qout << "http://gnu.hu/gplv3.html" << endl;  
  
QRect rect = QApplication::desktop()->availableGeometry();  
BrainBWin brainBWin ( rect.width(), rect.height() );  
brainBWin.setWindowState ( brainBWin.windowState() ^ Qt:: ←  
    WindowFullScreen );  
brainBWin.show();  
return app.exec();  
}
```

Ez ismét egy QT program, az összes szükséges fájl megtalálható a forrásnál. Fent csak a main.cpp látható de itt a többről is beszélünk. Először nézzük a BraintBTheard.cpp-t. Itt áll elő a kezdő pozíció. Létrejön a mi karakterünk és 4 másik karakter. Ezek úgy vannak mindig elhelyezve, hogy mindig egymás közelébe legyenek, hogy a feladatunk ne legyen túl könnyű. A run függvény a teszt indításáért felel. Itt méri az időt is, azaz a program addig fut amíg az idő a megaditt időnl(10 perc) kisebb. Itt található még a pause függvény aminek a neve a válasz. A következő BraintBTheard.cpp található a karakterek kinézete, ezeknek a neve programon belül "hero" azaz hős. Itt adjuk meg a nevét, az elhelyezkedését, színét, és a mozgásának a gyorsaságát. Továbbá a többi karakter születése, gyorsasága és további információkat róluk itt adunk meg. A BrainBWin.cpp -ben megadjuk a program nevét a verzió számát. majd az UpdateHeroes függvényben zajlik a a kurzorunk menetének vizsgálata, itt figyelni hogy rajta vagyunk e az egerrel a karakteren, hányszor vesztettük el a karakterünk, vagy hogy éppen fut e a teszt. Ezek utána a következő függvény kirajzolja az ablakot. Itt van továbbra az óra megjelenítése vagy a pontszámunkké. Ezek után jön az eger funkciói. Például ha lenyomjuk akkor elindul. Vagy az elmozdítás követése, És a billentyűzetről bevitt karaktereket is itt dolgozza fel.



8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Először is a Python nyelvet mutatjuk be. A nyelv egy magas szintű programozási nyelv. 1991 ben került nyilvánosság elé, amit Guido holland származású programozó fejlesztett. A python interpreteres nyelv.

A Minst kézzel írott számok adatbázisa (6000 kép). Ez az alapja azoknak a programoknak ami képről ismeri fel a tárgyakat. A kézel beírt számokról a program el fogja dönteni hogy milyen szám, ez azért érdekes, mert a kézzel írott írás szinte mindenkinél másabb, viszont a programnak mindig tudnia kell, hogy melyik számot kell felismernie. A programhoz a TensorFlowot használjuk. A TensorFlow a Google által alkotott gépi tanulási rendszer. Sok helyen használják, az egyik leghasználtabb, az a google mapsban található utcakép. Ez neutrális háló helyett itt transzformációs gráfok találhatóak. A TensorFlow nyílt forráskódu, le kell töltenünk a használathoz. Nézzük a kódot. Először is a könyvtárakat amik kellenek, a from kulcsszóval fogjuk ezeket hozzáadni a programhoz. Utánna beimportáljuk a Tensorflow könyvtárt. Ezután következik a main. Ebben van a kiíratás felépítése, hogy hogyan küldjük ki az eredményeket (a képen látszik.). A sess azzaz egy session segítségével fogjuk a tanítást végezni, hasonlóan mint a neurális hálózathoz. Az alaposság kedvéért, hogy minél pontosabb eredményt kapjunk ezerszer futtatjuk a ciklust. Zárunk kiíratjuk mennyire lett pontos az eredmény. A programban először felugrik maga a kép ami a kézel írott számot tartalmazza (ehhez a matplotlib-et használjuk, hogy meg tudjuk rajzolni), ha ezt bezárjuk jön a következő kép, a képeket az aktuális mappában lementjük. A programnak elég nagy a pontossága, jól felismeri. Az eredményeket egy tömbben tárolja el a program.

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

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse

# Import data
from tensorflow.examples.tutorials.mnist import input_data
```

```
import tensorflow as tf

import matplotlib.pyplot

FLAGS = None

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

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

    # Define loss and optimizer
    y_ = tf.placeholder(tf.float32, [None, 10])

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

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

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

    print("-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a tovaabblepeshez csukd be az ablakat")

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

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

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

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

print("-- A saját kezi 8-asom felismerese, mutatom a szamot, a ↵
    tovabblepeshez csukd be az ablakat")

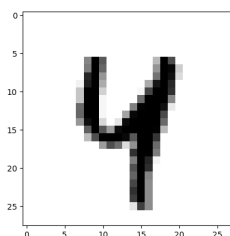
img = reading()
image = img.eval()
image = image.reshape(28*28)

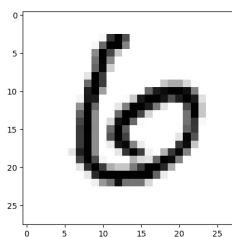
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm ↵
    .binary)
matplotlib.pyplot.savefig("8.png")
matplotlib.pyplot.show()

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

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

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





```
-----  
-- A hálózat tesztelése  
-- Pontosság: 0.9184  
-----  
-- A MNIST 42. tesztkepek felismerése, mutatom a számot, a továbblépéshez csak  
d be az ablakot  
-- Ezt a hálózat ennek ismeri fel: 4  
-----  
-- A MNIST 11. tesztkepek felismerése, mutatom a számot, a továbblépéshez csak  
d be az ablakot  
-- Ezt a hálózat ennek ismeri fel: 6  
-----  
(venv) k1ssmate3@k1ssmate3-VirtualBox:~$
```

8.2. Mély MNIST

Python

Itt használom fel az első passzolási lehetőséget a SMNISTforHumansExp3 ban elért és közzé tett eredmény miatt.

8.3. Minecraft-MALMÖ

Megoldás forrása: https://bhaxor.blog.hu/9999/12/31/minecraft_steve_szemuvege

Kezdjük azzal, hogy mi is az a Malmö. Ez egy Microsoft projekt a neve Project Malmö. A lényege pedig, hogy a mesterséges intelligencia segítségével feldolgozzuk az adott környezetet. A kísérletet az ismert játékban a Minecraftban végezzük. A feladat lényege, hogy mesterséges intelligenciával övezzük fel a karaktert és elindítjuk utjára, a MI feladata, hogy a környezetet érzékelje a karakter és kikerülje az akadályokat, ne akadjon el. A szükséges fájlok megtalálhatók a Microsoft Githubján. A feladat megoldása a következő. Mindig adott egy kocka amin a karakterünk épp áll, ez a kocka körüli 26 kockából álló területet fogja a program vizsgálni. Azaz összesen 27 kockát vizsgálunk. Ezeket a kockákat megkülönböztetjük. Ugye a játékban lehet fű (fold is az), levegő, magasfű, levél. Ezeket vizsgálva kell a programnak eldönteni hogy a karakter merre mozogjon. Ehez szükséges még parancsokat létrehoznunk, hogy a karakter hogyan mozogjon, mit tegyen ha akadályba ütközik, pl ha magas blokk, akkor ugorjon, vagy ha fa van előtte kerülje ki. Nézzen körbe merre tud menni.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Tutor:Földesi Zoltán

A lisp programozási nyelv a mesterséges intelligenciát kutatók kedvelt nyelve, eredetileg nem ez volt a célja, de hamar az MI kutatásban lett használatos. Neve jelentése a listafeldolgozás, mivel a programok felépítése láncolt lista(zárojelekkel választjuk el a listákat). A lisp egy kifejezésorientált nyelv.

Nézzük meg elsőnek Iteratív módon.

```
(defun faktorialisi (n)
  (do
    ((i 1 (+ 1 i))
     (prod 1 (* i prod)))
    ((equal i (+ n 1)) prod)))
```

Az elején a defun-al adjuk meg a függvény nevét és a változót amibe majd az érték érkezik. A do egy konstrukció, ezt iteratív programoknál használjuk mint ez is(iteratív struktúra). a (+ 1 i)-ben növeli az i értékét 1 el utána végzi el a szorzást aztán növeli az n et, amíg szükséges.

Rekurzív módon:

```
(defun faktorialisr(n)
  (if (= n 1)
      1
      (* n (faktorialisr (- n 1)))))
```

Az előző program rekurzív változata. Az elején a defunnal adjuk meg a nevet és a paramétert. Aztán if el vizsgáljuk hogy n egyenlő e 1 el, ha egyenlő akkor meghívja önmagát n-1 re és összeszorozza az n el.

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



```
FAKTORIALISI
Break 2 [6]> (faktorialisi 5)
120
Break 2 [6]> (faktorialisr 5)
120
Break 2 [6]> 
```

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Itt használom fel a második passzolási lehetőséget a SMNISTforHumansExp3 ban elért és közzé tett eredmény miatt.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

A könyvben legelőször az alapfogalmakkal fogunk foglalkozni. Először is nézzük a programozási nyelveket. ezeknek 3 szintjét különböztetjük meg. Ez a három a gépi nyelv, assembly nyelv és a magas szintű nyelv (C++, C ...stb). Minket a magas szintű programozási nyelv fog érintetni. Az ebben írt programokat forrásprogramoknak nevezzük. Ezt a gépnek értelmeznie kell és feldolgoznia gépi nyelvre. Ezt egy fordító program segítségével jahtjuk végre, Ez először tárgyprogramot hoz létre majd ebből lesz a gépi nyelv. Az átfordításnak 4 lépése van. 1. lexikális elemzés. 2. szintaktikai elemzés. 3. szemantikai elemzés. 4.kód-generálás. Az első lépésnél a forrást lexikális egyégekre bontjuk. A 2 lépésnél a szintaktika helyességét vizsgálja, hogy megfelel e a szintaktika szabályainak. Mivel, ha nem helyes a szintaktika, nem lehet gépi nyelvet ekőállítani, ugyanis nem fogja megérteni. A 3 lépés a program megértése szemantikailag. A 4. lépés pedig legenerálni a kódot. Ezt kapja meg a vezérlő operációs rendszer. A másik technika az interpreteres technika, az első 3 lépés itt megegyezik a fordító programokkal, a különbség, hogy itt nem készül tárgyprogramot. Sorba veszi az utasításokat és sorban értelmezi azokat majd végrehajtja. Minden nyelvnek saját hivatkozási nyelve van. Ebben van a nyelv szemantikai és szintaktikai szabályai definiálva. A szemantikai részt emberi nyelven szokták megadni, a legelterjettebb az angol nyelv ezen a téren. A következő pontban a nyelvek osztályozásába kapunk betekintést. 2 fő osztályra bontjuk őket. Imperatív nyelvek és Deklaratív nyelvek. Az utóbbi algoritmikus nyelv, még az előbbi nem algoritmikus nyelv. A könyvben részletesen ki vannak fejtve. A következő a jelölés rendszer. A szintaktika formális leírásához alkalmazzuk ezeket. Vannak terminális és nem terminális jelölések. A jelölések után következnek a kifejezések. Ezek szintaktikai eszközök. Ezeknek van értéke és típusa. A kifejezések formálisan operandusokból, operátorokból és kerek zárójelekből állnak. Az operadusok a változók vagy függvény meghívása lehet. Az operátorok pedig a műveletek amit az értékekkel elvégzünk. A záró jelek egybe foglalják és a sorrendet befolyásolják. Egy ilyen kifejezésnek 3 alakját tudjuk felírni: Prefix (pl: + 7 2), infix (pl: 7 + 2), postfix (pl: 7 2 +). Az infix alakban az operátorok nem azonos erősségűek. HA egyértelmű infix kifejezést akarunk létrehozni akkor teljesen záróljeleznünk kell azt. Ez felül írja a precedencia táblázatot. A könyvben továbbá zt vizsgáljuk milyen nyelv , hogyan használja és hogyan értékeli ki a kifejezéseket. A C egy kifejezés orientált programozási nyelv. Típuskényszerítés elvét használja. A C precedencia táblázatát is megismerjük a könyv 51 oldalán. Például: [] ez a tömb operátor, . minősítő operátor, -> mutatóval minősítő operátor, ++ és -- értéknövelő és csökkentő operandusok, sizeof() operátor típus vagy kifejezés hosszát adja meg. Vannak a matematikai operátorok, mint a szorzás, osztás, összeadás, kivonás. Hasonlító operátorok mint az egyenlő,

nem egyenlő, nagyobb vagy egyenlő...stb. A logikai operátorok: "vagy", "kizáró vagy", "és". A könyvben ezekhez találunk leírást, hogy hogyan használjuk. Az adattípusokkal is foglalkozunk a 2.4-es pontban. Az adattípus megadja, hogy a változó milyen típusú, azaz minden típusnév egy azonosító. 3 dolog határoz meg egy adattípust: tartomány, műveletek, reprezentáció. A programozási nyelvekben lehetőség van definiálni típusokat. Lehet létrehozni is de minden nyelvben vannak beépített típusok. Ilyen például az int azaz az egész típus, ennek vannak variánsai például short vagy long int, a bool a logikai típus, float vagy a double amik lebegőpontos számokat tudnak tárolni. A könyvben megnézzük az egyszerű és összetett típusokat. A könyvben nagy sújt kap a saját típus létrehozása. Szót kell még ejteni a tömbökről, ezeknek is van típusa ugyan úgy mint a változóknak, a tömbök olyan típusú változókat tartalmaznak amilyen típusú a tömb, ezeken belül a tömb indexével tudunk mozogni. A mutató típust is átvesszük. A nevesített konstans egy olyan eszköz a programozásban aminek 3 része van: Név, Típus, Érték. Ezt a 3-at mindig deklarálni kell. Ugye a konstans jelentése állandó, tehát ez a programban mindig a deklarálásnál megadott nevet, értéket és típust fogja tartalmazni. Ezeket érdemes beszélő nevekkal ellátni, és olyan értékeket adni amiket sokszor használunk. C-ben ezt az alábbi módon kell: `#define név literál`. A változó a konstant "testvére" úgy mond. Neki 4 komponense van, Név, attribútum, cím, érték. A változó ahogy a neve is mondja nem állandó. A név az azonosítója. ezzel hivatkozunk rá a programban. Az attribútumok közül a legfontosabb hogy milyen típusú, a típusokról már fentebb beszéltünk, amilyen típusú olyan értéket tud tárolni. És tudunk címet adni neki, azaz hol helyezkedjen el a tárolóban. de az elhelyezést a gép végzi, de hogy mikor hozza létre az attól függ hogy hol deklaráljuk. Nézzünk egy példát: `int a=5`. ez egy egész típusú változó kezdőértéke 5 a neve pedig "a". Most nézzük meg a C nyelv alapelemeit. Vannak integrált típusok (int, char...stb). Származtatott típusok (tömb, függvény, union, mutató). Ezek az aritmetikai típusok. A tömböt az alábbi módon hozzuk létre: `int a[elemszám]`. megadjuk a típusát a nevét és hogy hány elemű. Az utasítások a következő rész. Az utasítások alkotják a programot. Ezekből épül fel egy algoritmus, ciklus, és szinte mindent, az utasításokat fordítja le a fordítóprogram tárgyprogramra. Kér nagyobb részre tudjuk bontani, deklarációs utasítás és végrehajtható utasítás. A deklarációs utasítás a fordítóprogram miatt vannak. tőle kérnek szolgáltatást vagy egy üzemmemóriába való lépést. Befolyásolja a tárgykódot de nem kerül lefordításra. A végrehajtható utasítás, ahogy a nevében is benne van, bővégre lehet hajtani. Ezekből lesz generálva a tárgykód. Ezeket tudjuk csoportosítani például értékeadó utasítás, üres utasítás, ugró utasítás, elágazó utasítás, hívó utasítás, I/O utasítások ...stb. Ezekről a könyvben részletes leírást kapunk. például az értékeadó utasításban értéket adunk vagy módosítunk egy változón vagy több változón. Az elágazó utasítások is szinte kihagyhatatlanok egy programból, ugye ezek az if és az else if feltételes utasítások. Ezek lehetővé teszik a programban a több irányú elágazást. A ciklusszervező utasításokat is ismerjük már ha nem kezdők vagyunk. Ugye ezek a for, while, do while ciklusok. A végtelen ciklus volt az első feladatunk, azt már ismerjük, Vannak feltételes ciklusok, ugye itt addig fut a ciklus még a feltétel igaz. van kezdő feltételes és végfeltételes ciklus. Ezeknek a működését a könyvben megismertük. Van az előírt lépésszámú ciklus (a for()). Annyiszor fut le a ciklus, ahányszor előírtuk neki. Vannak összetett ciklusok Ez az előzők kombinációja. A működésük nagyon bonyolult. 3 vezérlő utasítás van C-ben. 1. Return: Ez szabályosan befejezi a függvényt és vissza adja azt a vezérlést hívónak, legtöbbször értékkel tér vissza. 2. BREAK: ezt a cikluson belül alkalmazzuk, ha életbe lép akkor kilép a ciklusból és az utána lévő cikluson belüli utasításokat nem hajtja végre. Ezt iffel szoktuk alkalmazni. 3. Continue: ez is a ciklus magában van. ez is kilép, vagy újabb cikluslépésbe kezd, vagy megvizsgálja újra az ismétlődés feltételeit. Az 5. fejezetben a programok szerkezetét ismerjük meg, hogy milyen részekből áll. Itt ismerjük meg az alprogram feladata egy bemeneti adatcsoporthoz leképezése vagy kimeneti adatcsoporthoz képezése, egy megadott specifikáció szerint. Ebben a fejezetben ismerjük meg a blokkokat. a blokk egy programegység, ami utasításokat foglal magában. A kezdetét és végét speciális karakter jelzi. A blokk bárhol elhelyezhető a programban. A 13. I/O fogjuk venni. Ez egy olyan rész ahol a program nyelvek nagyban eltérnek egymástól. Állományuk a függvények szerint 3 lehet. lehet input, output és input-output állomány. Az I/O során a programban az adatok a tároló és a periféria kö-

zött mozognak. Ezeknek van egy bizonyos ébrázolási módja. Az adatátvitelnek 3 fajtáját alkalmazzuk: formátumos módú, szerkesztett módú és listázott módú adatátvitel. Ha állományokat alkalmazunk azt a következőképpen kell csinálnunk. 1. deklaráció, 2. összerendelés, 3. állomány megnyitása, 4. feldolgozás, 5. lezárás. A C-nyelvnek nem része az I/O erre egy standart könyvtárat használunk.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

A könyvben a C nyelv alapjaival fogunk megismerkedni az alapoktól. A könyv programok segítségével segít elsajátítani a C nyelv ismeretét. Ha eddig egyáltalán nem találkoztunk a C nyelvvel, a könyv akkor is nagyon hasznos lehet, hiszen nagyon részletesen, mindenre odafigyelve adja át a tudást. Először kezdjük a változók típusaival és a program belüli alkalmazásával. A programban különböző típusú ezeken belül különböző méretű változók vannak. Típusok: int -integer, azaz egész típusú számot deklarálunk ezzel, mérete a gép egészétől függ. char -Ez egy karaktert tud tárolni, mérete 1 bájt. float -egyszeresen pontos lebegőpontos számot tud tárolni (törtet). double- kétszeresen pontos lebegőpontos számot tud tárolni, ezt hatalmas számokkal való számoláshoz használjuk. Az int típusnak vannak fajtái még, ilyen a short int vagy a long int, van továbbá még unsigned int amivel a negatív számok helyett, csak pozitív értékeket tudunk tárolni, de abból többet mint a sima int-nél. Továbbá ami még lényeges lehet, az lokális és globális változó, ahogy a nevük is jelzi, a globális az egész programban használható, még a lokális csak a megadott helyen. A könyvben a helyes alkalmazásról rengeteg példát láthatunk. Vannak az állandók, ezeknek megadunk egy értéket, és ez az érték állandó lesz a programban, nem változik. A könyv következő fejezetében a Vezérlési szerkezetet vesszük. Ez határozza meg, hogy milyen sorrendben hajtsanak végre a műveletek. Először az utasításokat és a blokkokat nézzük. Fontos szabály, hogy minden utasítást ";" -al zárunk le. Így válik egy sor utasítássá. Ha az utasítások {}-el vannak körbe zárva, akkor azt egy bloknak nevezzük és 1 db utasításlént kezeljük. A blokkok közé soroljuk a ciklusokat és a az elágazásokat. Először az if utasítással ismerkedünk meg. Ha a megadott feltétel igaz, akkor a megadott utasítások fognak lefutni, ha nem igaz akkor az utasításokat figyelmen kívül hagyja. Ennek a bővítése az else-if. Itt meg tudjuk adni, hogy a program milyen utasításokat hajtson végre, ha a feltétel nem igaz. A következő a switch utasítás. A lényege hogy megadunk neki mintákat, és ha a vizsgált elem megegyezik valamelyik mintával, a mintánál megadott utasítások fognak lefutni. ezek után vegyünk a ciklusokat. A ciklus egy olyan blok ami a feltételig újra és újra lefut. Kezdjük a while ciklussal, a while jelentése amíg, tehát amíg a kifejezés igaz addig lefut a ciklus, itt az lesz fontos, hogy először vizsgálja meg, hogy a feltétel igaz e, aztán fog lefutni az utasítás vagy utasítások. Ezzel megegyezik a for utasítás, a lényege és működési elve ugyan ez. A felépítése különbözik. A do while ciklus először végrehajtja az utasítást és aztán vizsgálja meg a feltételt, ha nem igaz akkor kilép a ciklusból. A break utasítással a futás közben is kitudunk lépni egy ciklusból ezt a ciklusokban lévő if-ekben szoktuk alkalmazni. A goto -val pedig egy adott címére ugorhatunk. De ezt nem sűrűn alkalmazzák, sokan nem is szeretik. Rendezni tudjuk ezeket a futási sorrend alapján. Van a sima ami sorban fut le, de van a címkézett utasítás blokk az előbb említett goto-val, itt az adott feltételektől függ, milyen sorrendben fut le a program. A Függelékben az utasításoknál csoportosítjuk az utasítások fajtáit. A címkézett utasításokhoz előtagként megadott címke kapcsolódik. A kifejezésutasítás, a nevéből adódóan kifejezésekből épül fel. Az összetett utasítás megszünteti a korlátozást ahol a fordító csak egyetlen utasítást fogad el. A kiválasztó utasítások, a lehetséges esetek közül választ a feltételnek megfelelően (if, switch). A vezérlés átadó utasítások, amár fentiekben említett goto, continuum break, return parancsokat alkalmazzák. Az Iterációs utasítások a ciklusokat alkalmazzák (while, do, for).

10.3. Programozás

[BMECPP]

A könyv témája a szoftverfejlesztés C++-ban. A legelső fejezetben azokat az új dolgokat fogjuk megnézni amiket C-ben nem tudtunk de C++ ban már lehetséges. Elsőre függvényparaméterek és visszatérési értékek vesszük. A C ben üres paraméterlistával definiált függvényt itt C++ ban void paraméter megadásával oldjuk meg. A visszatérési típusnál is eltér a 2 nyelv. Míg a C nyelvél int, addig C++ nem támogatja az alaphelyeztetett típust. Az int main fő függvénynek is 2 típusa van C++ -ban. Az első mikor nem adunk meg semmit, a másik mikor argumentumokat adunk a mainnak a parancssorból. A C++ nyelvben jelent meg először a bool azaz a logikai típus aminek értéke true vagy false. A feladatok között van egy ahol az volt a hiba, mikor utasításon belül deklaráltunk. C++ ban már ez is lehetséges. A példa erre, hogy az i értéket a forcikluson belül deklaráljuk: "for(int i=0; i kisebb mint 10; i++)". Hasznos lehet, ha a deklarációt a felhasználás előtt végezzük. C++ ban a függvényeket a nevük és a megadott argumentumokkal azonosítjuk. Azonos nevű csak akkor lehet, ha más az argumentum lista. C-ben a paraméterátadás érték szerint történik. A megkapott érték klonózva lesz és a másolatra fogunk hivatkozni, azaz a visszatérési érték nem befolyásolja a programunkat. Ez az adott példában a könyv jól szemlélteti. Ennek C++ ban a megoldása hogy az eredeti változót fogjuk átadni mégpedig a memória címével. Ezt a referencia jellel fogjuk teljesíteni. Így a változtatások, az eredeti változón játszódnak végbe, és az új értéket kapja vissza a program. Ez a referencia C-ben nem létezik. A harmadik fejezetben megismerjük az Objektumokat és osztályokat. A C++ nyelv egy objektumorientált nyelv, ennek alapelveivel megismerkedünk. A lényege hogy a függvényeket osztályokba foglaljuk az osztályok példányait nevezzük objektumoknak. (objektumokkal hívjuk meg az osztályok függvényeit.) A 3.2 bekezdésben egy példán keresztül láthatjuk, az egységbezáras előnyeit. A struktúrának így lesznek tagváltozói, és tagfüggvényei is. A tagfüggvényeket kétféle képpen adhatjuk meg. Osztálydefinícióban vagy struktúradefiníción kívül. Fontos még az adatrejtésről is beszélnünk. Így csak az osztályon belül férhetünk hozzá, külső osztály nem férhet hozzá, az értéket függvényel tudjuk kiadni. függvény és változó is lehet ilyen, ezeket a private részbe írjuk, azaz ezeket priváttá tesszük. A konstruktor onnan ismerjük fel, hogy ugyan az a neve mint az osztálynak és nem adunk típust neki, ez akkor fut le, ha meghívjuk az adott osztályt amiben benne van, egy objektummal, ezt gyakran inicializálásra használjuk. A destruktort annyiban különbözik szintaktikailag, hogy a név előtt egy '~' jel található, és automatikusan meghívódik egy objektum befejeződéskor, ezt a memória felszabadításra szoktuk használni főként. A c-ben a dinamikus memória foglалás malloc-al vagy a free kulcsszavakkal történtek. A C++ ban már operátorral tudunk lefoglalni memóriát, ez a new operátor, példa: "int *pelda; pelda= new int;" később ezt a változót fel kell majd szabadítanunk, vagy memóriaszivárgás lép fel. A dinamikus adattámogatás miatt szükséges sokszor, hogy megadjuk a mutató állapotát. Ez a NULL kulcsszó. Ha az állapot NULL akkor nincsen lefoglalt adat, ha ezt nem adjuk meg akkor a new szócskával lefoglalt területre mutat. A 3.5.3 fejezetben ismerkedünk meg a másoló konstruktorral. A másolókonstruktor egy referenciát kap, amely megegyezik az osztálynak a típusával. A másolókonstruktor is rendelkezik mindennel, amivel egy egyszerű konstruktor. Ugyan úgy tudunk inicializálni vele objektumokat. Amiben viszont több hogy érték szerint adunk egy függvényparamétert neki, akkor a megadott változó lemásolódik, és ezt használjuk a függvény törzsében. Az osztályainknak lehetnek friend függvényei vagy osztályai. Ez azt jelenti, hogy jogot adunk egy függvénynek vagy egy osztálynak, hogy hozzá férjen az adott osztály védett, azaz private és protected változóihoz és függvényeihez. Ezt a friend kulcsszóval tesszük lehetővé. A feljogosítandó függvény vagy osztály elé írjuk és ezzel fel is jogosítjuk. A tagváltozóknál az értékadás és az inicializálás nem ugyan az. Az inicializálás azaz alaphelyzetbe álltjuk. A létrehozásnál megadunk neki egy alap értéket. Az értékadás az az "=" jelel történik a programtól függően, ez a programon belül bárhol megtörténhet ahol a szabályok engedik. Említenünk kell a statikus tagokat, ezek a tagok az osztályhoz tartoznak és nem az osztályob-

jektumhoz. Továbbá lehetőségünk van struktúráknak, osztályoknak típusdefinícióra, a typedef szóval. A hatodik fejezetben vesszük az operátorokat és az operátortúlterhelést. A C nyelvben az operátorok műveleteket végeznek az argumentumokon. Az operátorok kiértékelési sorrendjét egy speciális szabályrendszer határozza meg, ezeket ugyan úgy mint matematikában, zárójelekkel írhatunk felül. A C++ rendelkezik új operátorokkal a C -hez képest. Ilyen a hatókör operátor aminek jelölése a "::" ezt az osztályok hatólörének megadásánál használjuk. Ismerjük már a "*" operátor, ugye ezzel jelöljük a mutatókat. A "->" operátorral pedig mutató esetén hivatkozunk. Fontos különbség a C és a C++ között a függvénymellékhata, A C nem képes erre, de a C++ igen. Ráadásul mivel az operátor speciális függvény ezért különböző argumentumok esetén túl tudjuk őket terhelni. A 10 es fejezet a kivitelezésről szól. Ilyen a hagyományos hibák kezelése. De mi is az a kivitelezés? A kivitelezés egy mechanizmus, amely ha hibát fedez fel, akkor a hibakezelő ágra ugrik. Ilyen a try_catch blokk. A throw kulccsal küldünk egy kidobást. a try-catch pedig elkapja, ha egy catch ág megegyezik az elkapott típussal. Ha nem kapja el, azt kezeletlen kivitelnek nevezzük. Van egymásba ágyazott try-catch blokk, így tudjuk a kidobásokat külön szinten kezelni

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, Berners-Lee!

11.1. Bevezetés a mobilprogramozásba.

Az alábbi szövegben a Python programozási nyelvel fogunk foglalkozni a megadott könyv alapján. Kezdetben a nyelv jellemzőiről olvashatunk. A Python nyelv, más programozási nyelvekkel (C++, Java, C) ellentétbe elég csak a forrást megadni, ugyanis a fordítási fázisra itt nincs szükség. A Python használható a neves platformokon, mint például Unix, Windows, MacOS ...stb. A nyelv alkalmas prototípus alkalmazások elkészítésére, hiszen sokkal kevesebb erőfeszítéssel lehet benne dolgozni mint például a C++-ban vagy a Javában. A Python nyelv egy magas szintű programozási nyelv, mégis egyszerűsége hasonlítható az awk vagy Perl nyelvekhez. A nyelvben a Python kódkönyvtárat használjuk. Az ebben lévő modulok, gyorsabbá teszik a programok fejlesztését. A modulok használhatóak rendszerhívásokra, hálózatkérésre és fájlkezelésre is. A nyelvvel könnyen olvasható alkalmazást készíthetünk. Ennek oka az, hogy az adattípusok engedik, hogy összetett kifejezéseket röviden tudjunk leírni. Továbbá a más nyelvekkel ellentétbe a kódnak a csoportosításának tagolása tabulátorral vagy új sorral történik és nem kell definiálnunk a változókat vagy az argumentumokat. A Python nyelv szintaxisa behúzás alapú. Nem szükséges kapcsos zárójel vagy kulcsszavak használata. Egy blokk végét egy behúzással végezzük, ez lehet akár üres sor. Behúzással nem lehet kezdeni egy szkriptet. Minden utasítás a sor végéig tart, így nem szükséges a C, C++ vagy a Java-ban ismert ";" használatára. Ha túl hosszú lenne egy sor, akkor "\" jellel lehet ezt jelölni. Egy behúzás nem érvényes a folytatósorokra. A sorokat a nyelvben tokenekre bontja a nyelv, ezek között lehet tetszőleges üres karakter. A token fajták: azonosító, operátor, kulcsszó ...stb. A nyelvben megkülönböztetjük a kis és nagy betűket. Pythonban objektumokkal reprezentálunk minden adatot és az ehez kapcsolatos műveleteket az objektum határozza meg. Az adattípusok hasonlóan a többi nyelvhez itt is lehetnek, sztringek, számok, ennesek, szótárak és listák. A szám típuson belül lehet egész szám, ami lehet lebegő pontos és komplex szám ezen belül is decimális, oktális vagy hexadecimális. A lebegőpontos szám a C++-ban ismert double-nek felel meg. A pythonba megtalálható a szekvencia is, ez egy nem negatív egész számokkal indexelt gyűjtő. A sztringet két féleképpen lehet megadni: idézőjelek között "példa" vagy aposztrófok között 'példa'. Az ennes típusok vesszővel elválasztott gyűjteményei az objektumoknak. A lista lehet több különböző típusú elemekből. Az elemeket szögletes zárójelek között kell felsorolni. A szótár pedig kulcsokkal azonosított rendezetlen halmaza az elemeknek. A változók Pythonban objektumokra mutató referenciák. A változóknak itt nincs típusa, így különböző típusú objektumokra lehet hivatkozni. A változóknak a "=" egyenlőség jellel lehet értéket adni. A "del" kulcsszóval törölhetünk egy változó hozzárendelést.

11.2. Java és C++

A következőbe a C++ és a Java programozási nyelvet fogjuk összehasonlítani. Mind a két nyelv egy magas szintű programozási nyelv. A Java nagyban hasonlít a C++ nyelvre, mivel sokmindent átvett attól, ilyenek például a jelölések. A forrásszöveget olvasva látható, hogy a Java nyelv szintaxisa a C,C++ nyelvből fejlődött. A típusok megegyeznek, ugyan úgy int az egész, string a szöveg, char a karakter és így tovább. Ugyan úgy kell deklarálni, az utasítások és a ciklusok is megegyeznek. Így egy C++ programozó is könnyen meéríti a Java programokat és fordítva. Fontos különbség van a nyelvek fordítása között. A C++ nak szüksége van egy fordító programra, ami a programozó által megírt kódot lefordítja gépi kódra, a Javanál erre nincs szükség, ugyanis platformfüggetlen. A fordításhoz először létrejön egy bájtkód ami a Java Virtuális Gépre értelmezhető kód. Mind a két nyelv objektumorientált, de még a Java nyelv teljesen objektumorientált nyelv, addig a C++ egy többelvű programozási nyelv (azaz több programizációs módszert támogat). Az objektum orientált nyelv az mikor a programban az adatokat és műveleteket objektumokban foglaljuk egybe. A Java teljesen osztályokból épül fel, de ugyan úgy tudjuk a C++ ban is használni az osztályokat. Egy osztályt a "Class" kulcsszóval hozunk létre. Az osztályon belül vannak az objektumok, ezeket tetszőleges sorrendbe megadhatjuk. Ezek mellett itt adjuk meg a metódusokat. (A metódusok a változók, eljárások és a függvények). A megadott elemekre megadhatunk biztonsági szabályozást. Lehet public, ilyenkor nyilvános lesz az elem, mindenki számára látható. A private akkor csak az adott osztály láthatja, ha pedig protected akkor az leszármazott. Ha nem adunk meg semmit, csak a forrásban lesz látható. Egy objektumon belüli elemre úgy lehet hivatkozni, hogy az objektum nevét és az elem nevét egy pontal össze kapcsoljuk. Ha egy elem több részes, akkor ugyan így kell hivatkozni rá. Ha egy elem elé írjuk, hogy "static", akkor az az elem vagy elemek nem egy objektumhoz fognak tartozni, hanem az osztályhoz. A "new" szó egy operátor, ezzel tudjuk lefoglalni az új objektumunk számára a szükséges helyet, továbbá inicializálja is azt. Egy osztályban a metódusokat az osztályhoz hasonlóan a "class" kulcsszavakkal létrehozott szerkezeten belül kell megadni. A metódus deklarációja módosítókkal adható meg, ilyen a visszatérési érték, a metódus paraméterei, a metódus neve és a metódus törzsének a leírása. Nézzünk egy példát egy metódusra és annak meghívására:

```
System.out.println(pelda.toString());
```

Ez a metódus egy sztringet ad vissza az objektumból. A metódusneveket túl lehetett terhelni. Azaz egy osztály több metódusát el lehet nevezni ugyan azon a néven, de csak akkor ha a szignatúrájuk nem azonos, tehát különböző. Ezt az takarja, hogy a formális paramétereik száma nem egyezik. A java fordító ezek alapján tudja, hogy melyik metódust kell meghívnia. Visszatérve az objektumokra a Java megkülönbözteti magát az objektumot amely valahol a tárhelyben a változóival mztat...

12. fejezet

Helló, Arroway!

12.1. 00 szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! <https://arato.inf.unideb.hu/batfai.norbert/UDPROC> (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: [source/labor/polargen](#))

A feladatot kezdjük először azzal, hogy mit csinál a program. A program le fog generálni 10 számot véletlenszerűen. Ezt polártranszformációval fogjuk elérni. A polártranszformációt használja a Java szám-generátora a véletlen szerű számok generálására.

Nézzük a kódot:

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

Az első sorban létrehozzuk a publikus osztályunkat ami a Polárgenerátor nevet kapja. Ebben az osztályban két változót fogunk deklarálni. Az első egy boolean logikai változó, a másik egy double típusú változó. A logikai változó neve "nincsTarolt", ezzel azt fogjuk nézni, hogy van e eltárolt változó, ezt a program elején True-val adjuk meg, mert az elején nem tárol változót. A "tarolt" változóval pedig a kiszámolt változónak az értékét fogjuk eltárolni.

```
public PolárGenerátor() {  
    nincsTárolt = true;  
}
```

Ez az osztály konstruktor, ezt onnan tudjuk legkönnyebben hogy, ugyan az a neve mint az osztálynak. Ezzel végezzük a "nincsTarolt" változó inicializálását. Itt kap True értéket.

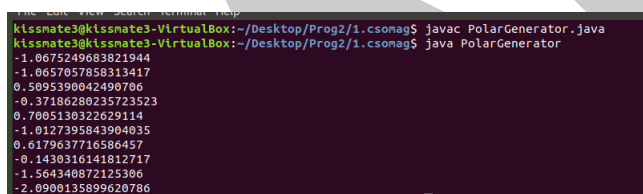
```
public double következő()
```

Ez egy metódus. Itt történik a matematikai számítás. Először egy if elágazással megnézzük, hogy van-e tárolt elem, vagy nincs. Ha nincs tárolt elem (azaz a "nincsTarolt" értéke igaz.), akkor elvégzi a számításokat. Két darab értéket fogunk kapni, az első értéket vissza adjuk return -el, míg a másikat eltároljuk a double típusú "tarolt" változóba. Ha van tárolva érték, akkor rálépünk az else ágra. Abban az esetben az eltárolt értéket fogjuk vissza adni.

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

A program végén pedig lássuk a main függvényt. Ez a fő függvény, ez fog lefutni a legelőször. A "new" kulcsszóval létrehozunk egy objektumot a PolarGenerátor osztálynak. Az objektum meghívásával fog lefutni a konstruktorunk, ami True értéket állít be, a logikai változónknak. Ezután jön a for ciklus, ez 10 alkalommal fog lefutni és minden alkalommal meghívjuk a public double következő() metódust. Így fogunk 10 darab véletlenszerű számot legenerálni.

A kódot 'PolárGenerátor.java' néven mentjük el, és futtatjuk az alábbi módon: 'javac PolárGenerátor.java'.



```
kissnate3@kissnate3-VirtualBox:~/Desktop/Prog2/1.csonag$ javac PolárGenerátor.java  
kissnate3@kissnate3-VirtualBox:~/Desktop/Prog2/1.csonag$ java PolárGenerátor  
-1.0675249683821944  
-1.0657657858313417  
0.5895390842490706  
-0.37186280235723523  
0.7005130322629114  
-1.0127395843904035  
0.6179637716586457  
-0.1430316141812717  
-1.564340872125306  
-2.0900135899620786
```

Most lássuk ezt C++ nyelven:

```
class PolarGen  
{  
public:  
    PolarGen()  
    {  
        nincsTarolt = true;  
        std::srand (std::time(NULL));  
    }  
    ~PolarGen()  
    {  
    }  
    double kovetkezo();  
private:  
    bool nincsTarolt;  
    double tarolt;  
};
```

Ugyan úgy, ahogy a Javában, itt is egy osztállyal kezdünk. Az osztályon belül van, publikus és privát rész. A public részben találjuk a konstruktort ami inicializálja a "nincsTarolt" változót. Továbbá itt található az srand() függvény, amivel C++ ban véletlenszerű számokat generálunk. A másik a destruktork, ezt a '~' előtagból tudjuk. Ezt általában memóriafelszabadításra használjuk. Ez akkor fut le, ha töröljük az

objektumot, vagy lezárjuk. A private részben pedig a két változónk van, amiknek feladatait már a Java-s kódban leírtuk. A private és public között az a különbség, hogy a public az osztályon kívül is elérhető még a private csak osztályon belül használható. A PolarGen osztályban találjuk még a `kovetkezo()` függvényt, ahogy a Java-s kódban itt is ez a függvény fogja végezni a számítást. Ez ugyan úgy ha nincs tárolt érték akkor egy értéket vissza ad, egyet pedig eltárol, ha pedig van tárolt érték, akkor azt adja vissza.

```
int main (int argc, char **argv)
{
    PolarGen pg;
    for (int i= 0; i<10;++i)
        std::cout<<pg.kovetkezo ()<< std::endl;
    return 0;
}
```

A fő függvényünkben itt is az objektum található és a for ciklus, ami 10 alkalommal fog lefutni. A két kódsor között a két nyelv különbségei, szintaxisa de a két nyelv hasonlóságai is megfigyelhető.

12.2. Homokozó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiírtani és minden máris működik (erre utal a feladat neve, hogy Java-ban minden referencia, nincs választás, hogy mondjuk egy attribútum pointer, referencia vagy tagként tartalmazott legyen). Miután már áttettük Java nyelvre, tegyük be egy Java Servletbe és a böngészőből GET-es kéréssel (például a böngésző címsorából) kapja meg azt a mintát, amelynek kiszámolja az LZW binfáját!

Tutor: Tóth Attila

A könyvünkben Welch fejezetében már foglalkoztunk az LZWBinfával, akkor C++ nyelven írtuk meg. Most át fogjuk írni Java nyelven. A működése ugyan az lesz. A program bemeneti adatokból fog felépíteni egy bináris fát. Egy fa akkor bináris, ha a fa minden csomópontjának maximum 2 gyermeke van. A bináris fa 0 és 1-es számokból épül fel. Fontos még tudni, hogy van egy kitüntetett elem, amit gyökérnek hívunk, innen tudunk elérni minden elemet, amit a fa tartalmaz. (A fa sok tulajdonsággal rendelkezik, van magassága, mélysége, elemszáma, szórása, ágak hossza, ágak száma).

A kód átírása után a java kódon (lásd a feladat alján) látszik a hasonlósága a C++ -os kóddal, nem sok változást vehetünk észre. Az első lépésünk az volt, hogy töröltük a pointereket és a referenciákat a C++ -os forrásból. Erre azért van szükség mert a Java nem tűri a pointereket és a referenciákat. Ezeket az objektumok referenciáival helyettesítjük. Lényeges különbség még a destruktort eltűnése. A destruktort a lefoglalt memória felszabadítására használtuk, de Javában nem lesz erre szükségünk, ugyanis a Garbage Collector azaz a szemétyűjtő megteszi ezt helyette. A szemétyűjtő automatikusan felszabadítja a már nem használt memóriát. Különbség még, hogy Javában nincs oerátortúlterhelés, így ezeket függvényekre cseréljük. Ezen kívül pedig a C++ osztályokon belül lévő public és privát részt is elhagyjuk, itt külön külön kell elé írni a függvény, változó elé, hogy milyen a jogosultsága.

Most nézzük, hogyan is tudjuk a böngésző címsorából átadni az adatokat a binfának. Ehez Java Servletre lesz szükségünk. Első dolgunk telepíteni az apache tomcat az alábbi módon: https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-8-on-ubuntu-16-04?fbclid=IwAR0juqF-I5y9SxYBk-_1-_lgNMchwPq_WEeyssS5PH5ldOdMlaFJpUvMXP4 Miután ezzel kész vagyunk, a böngészőn keresztül megadjuk a bemenetet. Ezt "?text=" módon tesszük meg (pl ?text=01010011101010)

```

File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/1.csomag
Get:2 http://hu.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcurl4 amd64 7.58.0-2ubuntu3.8 [214 kB]
Get:3 http://hu.archive.ubuntu.com/ubuntu bionic-updates/main amd64 curl amd64 7.58.0-2ubuntu3.8 [159 kB]
Fetched 1.673 kB in 1s (1.229 kB/s)
Preconfiguring packages ...
(Reading database ... 278679 files and directories currently installed.)
Preparing to unpack .../libssl1.1_1.1.1-1ubuntu2.1-18.04.4_amd64.deb ...
Unpacking libssl1.1:amd64 (1.1.1-1ubuntu2.1-18.04.4) over (1.1.0g-2ubuntu4.3) ...
Preparing to unpack .../libcurl4_7.58.0-2ubuntu3.8_amd64.deb ...
Unpacking libcurl4:amd64 (7.58.0-2ubuntu3.8) over (7.58.0-2ubuntu3.6) ...
Selecting previously unselected package curl.
Preparing to unpack .../curl_7.58.0-2ubuntu3.8_amd64.deb ...
Unpacking curl (7.58.0-2ubuntu3.8) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Setting up libssl1.1:amd64 (1.1.1-1ubuntu2.1-18.04.4) ...
Checking for services that may need to be restarted...done.
Checking for services that may need to be restarted...done.
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up libcurl4:amd64 (7.58.0-2ubuntu3.8) ...
Setting up curl (7.58.0-2ubuntu3.8) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
kissmate3@kissmate3-VirtualBox: /tmp$ curl -O http://apache.mirrors.tonfish.org/tomcat/tomcat-8/v8.5.5/bin/apache-tomcat-8.5.5.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left   Speed
100 327 100 327  0     0  1368      0  0:00:00  0:00:00  0:00:00 1368
kissmate3@kissmate3-VirtualBox: /tmp$ sudo mkdir /opt/tomcat
kissmate3@kissmate3-VirtualBox: /tmp$ sudo tar xzvf apache-tomcat-8.tar.gz -C /opt/tomcat --strip-components=1
gzip: stdin: not in gzip format
tar: Child returned status 1
tar: Error is not recoverable: exiting now

```

A kimeneten láthatjuk, hogy a Java és a C++ LZWBinfa kimenetje megegyezik:

```

kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/1.csomag
Open ▾  Open ▾  kimenet.txt
~/Desktop/Prog2/1.csomag
Save  ▢  ▢  ▢  ▢

-----1(3)
-----1(2)
-----1(4)
-----0(5)
-----0(3)
-----1(1)
-----1(3)
-----0(4)
-----0(2)
-----0(3)
-----1(5)
-----0(4)
---/(0)
-----1(3)
-----1(2)
-----1(4)
-----0(3)
-----1(5)
-----0(6)
-----0(4)
-----0(5)
-----0(1)
-----1(4)
-----1(3)
-----0(4)
-----0(2)
depth = 6
mean = 4.3
var = 0.9486832980505138

-----1(3)
-----1(2)
-----1(4)
-----0(3)
-----1(5)
-----0(6)
-----0(4)
-----0(5)
-----0(1)
-----1(4)
-----1(3)
-----0(4)
-----0(2)
depth = 6
mean = 4.3
var = 0.948683

Plain Text ▾  Tab Width: 8 ▾  Ln 23, Col 17  ▾  INS

```

A kód: <https://github.com/kissmate3/Forr-sok/blob/master/LZWBinFa.java>

12.3. Gagyi

Az ismert formális

```
while (x <= t && x >= t && t != x);
```

tesztkérdéstípusra adj a szokásosnál (miszerint x , t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írd Java példaprogramot mely egyszer végtelen ciklus, más x , t értékekkel meg nem! A példát építsd a JDK Integer.java forrására, hogy a 128-nál inkluzív objektum példányokat poolozza!

Javában az Integer típusú számokat a java -128 és 127 között eltárolja a poolba. (Feltételezi a java, hogy a program sokat fog kisebb számokkal dolgozni, és ha Integer típusú szám kell, akkor a poolból kivesz.) Ha

a -128 és 127 közötti intervallumból szeretnénk értéket adni egy változónak, akkor az előre eltárolt pool-ból fog hozzá memória címet rendelni. Ez úgy történik, hogy ebben az esetben meghívódik az `Integer.valueOf` függvény, ami kiosztja a címet. Ha a megadott tartományok kívül rendelünk a változóhoz értéket, például -129 akkor viszont ugyan ez a függvény létrehoz egy új objektumot. Ezek alapján ha `x` és `t` változó értéke egyenlő és ezen értéke az adott tartományon belül vannak, akkor azonos lesz a két változó címe. Viszont ha `x` és `t` értéke azonos, de nem a -128 és 127 tartományból lesz, akkor a két változó címe nem fog megegyezni, mivel két külön objektum fog létrejönni ezeknek a számoknak.

A megadott feltétellel írunk két példa programot: A feltétel a két eset kapcsán egyszer végtelen ciklusba lép, a másik esetben pedig a feltétel nem teljesül.

```
class elsoeset{
    public static void main(String args []){
        Integer x=120;
        Integer t=120;

        while (x <= t && x >= t && t != x)
            System.out.println("A feltétel nem teljesül");
    }
}
```

Ebben az esetben lááthatjuk, hogy az `x` és a `t` értéke is 120, azaz egyenlő, továbbá mind a két érték benne van a tartományba, tehát a poolból ugyan azt a címet fogják megkapni. A feltétel így nem teljesül, tehát a program nem fog végtelen ciklusba lépni.

```
class masodik eset{
    public static void main(String args []){
        Integer x=140;
        Integer t=140;

        while (x <= t && x >= t && t != x)
            System.out.println("Végtelen ciklus");
    }
}
```

Itt is megegyezik, `x` és `t` értéke, de ez az érték már a tartományon kívül lesz, így két különböző objektumból kapják a változók a címeket, így az összehasonlításakor nem lesz azonos a cím, tehát a programunk végtelen ciklusba fog lépni

12.4. Yoda

Írjunk olyan Java programot, ami `java.lang.NullPointerException`-el leáll, ha nem követjük a Yoda conditions-t! https://en.wikipedia.org/wiki/Yoda_conditions

A Yoda egy programozási stílus, pontosabban Yoda feltételeknek hívjuk. A lényege pedig az, hogy az összehasonlító kifejezésben megcseréljük a változót és a konstans. Eredetileg a konstanshoz hasonlítjuk a változót, azaz a konstans a bal még a változó a jobb oldalon helyezkedik el. De a Yoda feltételek szerint pont fordítva, azaz a konstans lesz a jobb oldalon és a változó a bal oldalon. Ez a megszokott szintaxistól

eltérő. A nevét a Csillagok Háborújából ismert Yoda mesterről kapta, mivel furán beszél az angolt, szavakat felcserél, nem a szokványos angol szintaxis szerint beszél. Nézzünk egy példát:

Általánosan:

```
if(érték == 20) {Tortenjen valami}  
// Ez egy hagyományos feltétel. Jelentése: Ha az érték egyenlő 20 al...
```

Yoda feltétel szerint:

```
if(20 == érték) {Tortenjen valami}  
// Ez egy nem hagyományos feltétel. Jelentése: Ha a 20 egyenlő az értékkel ↔  
...
```

Az első példa szerint a változó értéke 20 lesz, még a másik szerint hibát kapnánk a kód fordítása során. Ezért ha a Yoda stílusban írjuk a programot, ezt a hibát kiküszöböltük. További előnye a Yoda stílusnak még, hogy el tudjuk kerülni a nem biztonságos viselkedésű null típusokat:

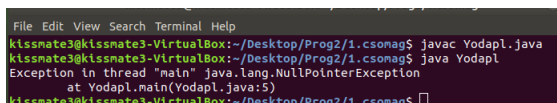
Yoda feltétel szerint:

```
String peldaString = null;  
if(peldaString.equals("foobar"))
```

A yoda feltételeknek vannak hátrányai. Azok akik kritizálják anyelvet, azt mondják, hogy olyan szinten rontja az olvashatóságát a programnak, hogy az meghaladja a Yoda stílus előnyeit. A Swift programozási nyelv és néhány másik nem engedi meg a változó hozzárendeléseket egy feltételen belül. Továbbá amit fent előnynek tüntettünk fel, az lehet hátrány is mivel a null muta hibákat elrejtjük ugyan, de ezek később jelentkezhetnek a megírt programban. Van egy hátrány ami C++ -ban van jelen. A hiba akkor jöhet elő, mikor nem alaptípusokat hasonlítunk össze.

Most pedig írjuk meg a programunkat ami java.lang.NullPointerException-el leáll, ha nem követjük a Yodaconditions-t!

```
class Yodapl{  
  
public static void main(String args[]){  
    String peldastring = null;  
    if (peldastring.equals("pelda")){  
        System.out.println("Pelda kod")  
    }  
}  
}
```



```
File Edit View Search Terminal Help  
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$ javac Yodapl.java  
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$ java Yodapl  
Exception in thread "main" java.lang.NullPointerException  
    at Yodapl.main(Yodapl.java:5)  
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$
```

12.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp- alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását! Ha megakadsz, de csak végső esetben: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Ebben a feladatban a Baliey-Borwien-Plouffe (rövidítve BBP) formulákról fogunk beszélni. Ez a formula a π (pí) képlete. A nevét a képlet felfedezőjéről és a cikket megjelenítő kiadójának vezetőiről nevezték el. A felfedező Simon Plouffe volt 1995-ben. A képlet a következő:

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right].$$

A képlet arra szolgál, hogy egy kiválasztott Pi számjegytől hexadecimális formában tudjunk kiszámítani további Pi számjegyeket. A képletet egy program találta, érdekesség, hogy ez volt az első lényegesebb formula, amit egy program talált.

Most beszéljünk a kódról:

```
public class PiBBP {
    String d16PiHexaJegyek;

    public PiBBP(int d) {

        double d16Pi = 0.0d;

        double d16S1t = d16Sj(d, 1);
        double d16S4t = d16Sj(d, 4);
        double d16S5t = d16Sj(d, 5);
        double d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

        Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};

        while(d16Pi != 0.0d) {

            int jegy = (int)StrictMath.floor(16.0d*d16Pi);

            if(jegy<10)
                sb.append(jegy);
            else
                sb.append(hexaJegyek[jegy-10]);

            d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
        }
    }
}
```

```
        d16PiHexaJegyek = sb.toString();  
    }
```

Elsőnek egy String típusú változót hozunk létre 'd16PiHexaJegyek' néven. Ezt a változót fogjuk használni a hexadecimális jegyek tárolására. Aztán következik a konstruktor, ami megkapja a megadott számot, ahonnan elkezd a számítást. A számításokhoz 5 darab double típusú változót deklarálunk. Ezek után láthatjuk a képletet a programba. A StrictMath.floor() függvényel vissza kapjuk az eredmény egész részét, A 'hexaJegyek' karaktertípusú változóba megadjuk, a hexadecimális jegyekhez szükséges betűket. Aztán egy while cikluson belül elvégezzük a számítást és egy if feltételen belül a megfelelő helyeken össze rakjuk az eredményt. A 'd16PiHexaJegyek = sb.toString();' sorban a kapott eredmény a toString() függvényel String be visszük át és így stringként kapja meg a 'd16PiHexaJegyek' változónk.

```
public double d16Sj(int d, int j) {  
  
    double d16Sj = 0.0d;  
  
    for(int k=0; k<=d; ++k)  
        d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);  
  
    return d16Sj - StrictMath.floor(d16Sj);  
}  
  
public long n16modk(int n, int k) {  
  
    int t = 1;  
    while(t <= n)  
        t *= 2;  
  
    long r = 1;  
  
    while(true) {  
  
        if(n >= t) {  
            r = (16*r) % k;  
            n = n - t;  
        }  
  
        t = t/2;  
  
        if(t < 1)  
            break;  
  
        r = (r*r) % k;  
  
    }  
  
    return r;  
}
```


Ezek után következik egy double típusú függvény ami megkapja paraméterül a 'd' kiválasztott számot és egy egész számot. Ebben a függvényben számításokat fogunk végezni maradékképzéssel. ehhez pedig a hosszú egész típusú n16modk függvényt hívjuk meg.

```
public String toString() {  
  
    return dl6PiHexaJegyek;  
}  
public static void main(String args[]) {  
    System.out.print(new PiBBP(1000000));  
}  
}
```

A toString függvényt már a fentiekben említettük, ez String típusba adja vissza az értéket. Végezetül a fő függvényben kiíratjuk az eredményt.



```
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$ javac PiBBP.java  
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$ java PiBBP  
6C65E5308kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$
```

13. fejezet

Helló, Liskov!

13.1. Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. [source/binom/Batfai-Barki/madarak/](#))

A Liskov féle helyettesíthetőségi alapelv (röviden LSP) egyike a S.O.L.I.D objektum orientált tervezési elveknek. Az elv nevét a kidolgozója után kapta, aki Barbara Liskov volt. A lényege pedig, hogy a leszármazott osztályoknak a példányainak ugyanúgy kell viselkednie mint az őosztályban lévő példányoknak. Ezt úgy tudnánk röviden megmagyarázni, hogyha egy programban az őosztály valamelyik példánya helyett a leszármazott osztály példányát használjuk, akkor a program működésében semmilyen változás nem történhet. Példának vegyük Struc és a Gólya példáját. A Struc és a Gólya is a madár osztályába tartozik. A Madár osztály rendelkezik a repül függvénnyel, mivel tudjuk hogy a legtöbb madár repül. Itt jön majd egy hiba az öröklődésben, mert a struc és a gólya alosztály is megörökli a repül függvényt, pedig a Struc nem tud repülni. Így a program működésében nem lesz hiba, de hibás programtervet eredményez. Hasonlóan tudunk még példákat felhozni, ilyen a kör és elipszis, vagy a négyzet és a téglalap területszámítása.

Most nézzünk egy C++ kódrészletet, amivel megsértjük ezt az elvet.

```
class Madar {
public:
    virtual void repul() {};
};

class Program {
public:
    void fgv ( Madar &madar ) {
        madar.repul();
    }
};

class Golya : public Madar
```

```
{};

class Struc : public Madar
{};

int main ( int argc, char **argv )
{
    Program program;
    Madar madar;
    program.fgv ( madar );

    Golya golya;
    program.fgv ( golya );

    Struc struc;
    program.fgv ( struc );
}
```

Most pedig nézzük meg Java-nyelven:

```
class Madar {
public void repul() {};
};

class Prog {
public void fgv ( Madar madar) {
    madar.repul();
}

};
class Golya extends Madar {};
class Struc extends Madar {};

class Liskov{
    public static void main(String[] args)
    {
        Prog kod = new Prog();
        Madar mad = new Madar ();
        kod.fgv (mad);

        Golya golya = new Golya();
        kod.fgv (golya);

        Struc struc = new Struc();
        kod.fgv(struc);
    }
}
```

A kódban először létrehozuk a Madár osztályt, ez lesz a szülőosztály. Létrehozuk benne a repul() függvényt. Az osztály létrehozása után a Gyólya és a Struc leszármazott osztály. Itt mind a két osztály megkapja

a madar osztályt. A Liskov osztályban pedig példányosítunk.

13.2. Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az őson keresztül csak az ős üzenetei küldhetők! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

A feladatunk az, hogy bebizonyítsuk egy programon keresztül, hogy egy ősosztály csak a saját metódusait tudja használni, a leszármazott osztály metódusait már nem. Azaz például az ősosztály csak azokat a függvényeket fogja tudni használni, ami a saját függvénye, egy új függvényt amit leszármazott osztályban hoztunk létre, azt nem fogja tudni használni.

Elsőnek nézzünk erre példát Java nyelven:

```
class Szulo
{
    public void szulofugv()
    {
        System.out.println("Szulo fuggveny");
    }
};

class Gyermek extends Szulo
{
    public void gyerfugv()
    {
        System.out.println("Gyermek fuggveny");
    }
}

public static void main (String[] args)
{
    Szulo obj1 = new Szulo();
    Gyermek obj2 = new Gyermek();

    obj1.szulofugv();
    obj2.gyer();
    obj2.szulofugv();
    obj1.gyer();
}
```

A kódot megnézve látjuk hogy az ősosztály a Szulo, az extends utasítással leszármazottja lesz a Gyermek osztály, mind a két osztályban létre hoztunk egy saját függvényt. A fő függvényünkhöz mind a két osztálynak létrehozunk egy egy objektumot obj1 és obj2 néven. Ezek után az objektumokkal meghívjuk a függvényeket. Az első három hívás rendesen működik, de az utolsónál mikor a szülővel akarjuk meghívni a gyermek függvényét beleütközünk a fent leírtakba, így bizonyítva van, hogy az őson keresztül csak az ős üzenetei küldhetők.

kissmate3@k

File Edit View Search Terminal Help

kissmate3@kissmate3-VirtualBox:~\$ javac m.java

m.java:25: error: cannot find symbol

obj1.gyerfugv();

^

symbol: method gyerfugv()

location: variable obj1 of type Szulo

1 error

kissmate3@kissmate3-VirtualBox:~\$

Most pedig nézzünk erre egy példát C++ nyelven:

```
#include <iostream>

using namespace std;

class Szulo{

public:
void szfugv(){
    cout<<"Szulo fuggve"<<endl;
}
};

class Gyermek: public Szulo{
public:
void gyfugv(){
    cout<<"Gyermek fuggveny"<<endl;
}
};

int main(){
Szulo* obj1 = new Szulo();
Gyermek* obj2 = new Gyermek();
obj1->szfugv();
obj2->gyfugv();
obj2->szfugv();
obj1->gyfugv();
return 0;}
```

```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~$ g++ m.cpp -o m
m.cpp: In function 'int main()':
m.cpp:26:7: error: 'class Szulo' has no member named 'gyfugv'; did you mean 'szfugv'?
  obj1->gyfugv();
        ^~~~~~
        szfugv
kissmate3@kissmate3-VirtualBox:~$
```

13.3. Anti OO

A BBP algoritmussal 4 a Pi hexadecimális kifejtésének a 0. pozíciótól számított jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanito-k-javat-apas03.html#id561066>

Az első csokorban megbeszélt BBP algoritmust futási idejét fogjuk most összehasonlítani négy programozási nyelven. Ez a négy nyelv a C, C++, C# és a Java. Az algoritmussal a 0 ik pozícióból a 10^6 , 10^7 és a 10^8 darab jegyeket fogjuk meghatározni. A számítások elvégzése után ezt az eredményt kaptam:

	Java	C	C++	C#
10^6	1.578	1.665	2.069	1.776
10^7	17.746	20.084	23.741	18.5
10^8	207.95	231.001	263.882	205.74

Az eredmények alapján arra jutottam, hogy a számítást az első két esetben a Java még a harmadik esetben a C# végezte el, De az első két esetben is nagyon kis különbség volt a Java és a C#. Minden esetben a leglassabb a C++ volt. A C nyelv első esetben a második legjobb volt, de a másik két esetben közepes eredményt ért el. A számításokhoz használt számítógép egy I7 7700HQ nyolcmagos processzor 8gb DDR4 rammal, de mivel Virtuális gépen végeztem a számítást így 4 maggal és 4gb rammal történt a teszt.

```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox: ~/Desktop/Prog2/2.csomag$ g++ PiBBP.cpp -o PiBBP
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./PiBBP
6
2.06969
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ g++ PiBBP.cpp -o PiBBP
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./PiBBP
7
23.7412
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ g++ PiBBP.cpp -o PiBBP
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./PiBBP
12
282.594
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$

File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ gcc pi_bbp_bench.c -o pi_bbp_bench -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
main ()
^~~~~~
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./pi_bbp_bench
6
1.665548
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ gcc pi_bbp_bench.c -o pi_bbp_bench -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
main ()
^~~~~~
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./pi_bbp_bench
7
20.084121
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ gcc pi_bbp_bench.c -o pi_bbp_bench -lm
pi_bbp_bench.c:77:1: warning: return type defaults to 'int' [-Wimplicit-int]
main ()
^~~~~~
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ ./pi_bbp_bench
12
231.001771
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$

File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mono PiBBPBench.exe
6
1.766523
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mcs PiBBPBench.cs
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mono PiBBPBench.exe
7
18.500215
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mcs PiBBPBench.cs
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$ mono PiBBPBench.exe
12
205.748154
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/2.csomag$
```

13.4. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/SMNIST>
Apró módosításokat eszközölj benne, pl. színvilág.

A feladatot az UDPROGban Racs Tamás által megosztott segítséggel tudtam elkészíteni, ezért tutor: Racs Tamás

A feladatunk az volt, hogy színeket változtassuk meg a SMINSTforHumans programban. Én ezt Windows 10 operációs rendszeren végeztem el az Android Studio ingyenes programot használva. Miután a feltelepítés megtörtént, a Studion belül megnyitottam a SMINSTforHumans projektet. A változtatásokat pedig a SMINSTSurfaceView.java állományban végeztem. Elsőnek a bgColor -ban írtam át a hátterek RGB kódjait (lásd az első kép). Majd további módosításokat hajtottam végre a textPaint, msgPaint, dotPaint és broderPaint változókon (lásd a második képen).

```

surfaceView.java
private static final int NUM_OF_DIGITS = 10;
private float[] senValueDots = new float[2 * NUM_OF_DIGITS];

private float[] digitsCoords = new float[2 * NUM_OF_DIGITS];

private int successCnt = 0;
private boolean armed = true;
private long armedTime = 0;

int[] bgColor =
{
    android.graphics.Color.rgb( red: 210, green: 83, blue: 77),
    android.graphics.Color.rgb( red: 255, green: 169, blue: 83)
};
int bgIdx = 0;

private int decisionTimeDelaySum = 0;
private int senValueSum = 0;

private static String msg1 = "How many dots can you see?";
private static String msg2 = "Touch the appropriate number.";

android.graphics.Paint dotPaint = new android.graphics.Paint();
MNISTSurfaceView cinit()

textPaint.setColor(Color.BLACK);
textPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
textPaint.setAntiAlias(true);
textPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
textPaint.setTextSize(50);

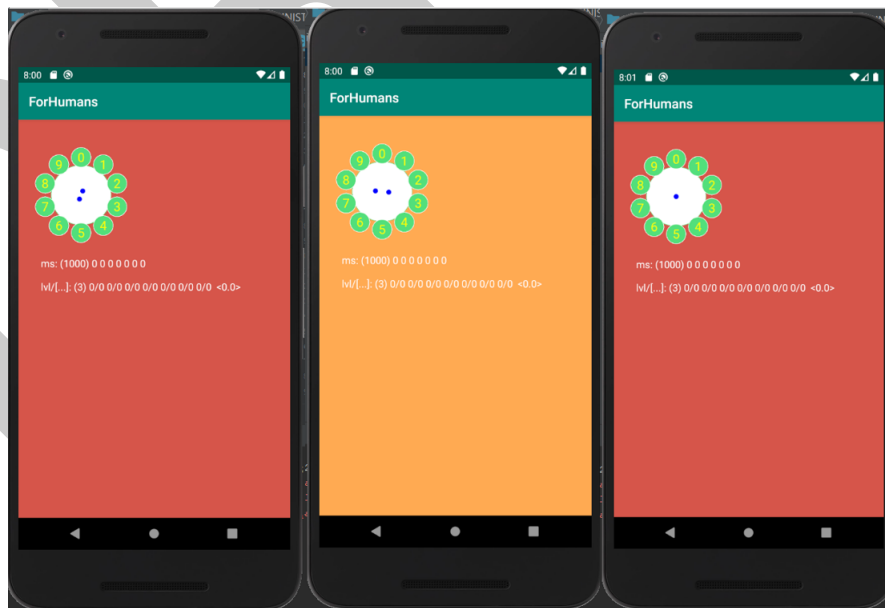
msgPaint.setColor(Color.WHITE);
msgPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
msgPaint.setAntiAlias(true);
msgPaint.setTextAlign(android.graphics.Paint.Align.LEFT);
msgPaint.setTextSize(40);

dotPaint.setColor(Color.BLUE);
dotPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);
dotPaint.setAntiAlias(true);
dotPaint.setTextAlign(android.graphics.Paint.Align.CENTER);
dotPaint.setTextSize(50);

borderPaint.setStrokeWidth(2);
borderPaint.setColor(Color.WHITE);
fillPaint.setStyle(android.graphics.Paint.Style.FILL);
fillPaint.setColor(android.graphics.Color.rgb( red: 45, green: 54, blue: 166));
STSurfaceView cinit()

```

Ezek után a programot lefuttatva a szoftveren belül egy virtuális telefonon megkapjuk az eredményt. A program kinézetét a szöveg alatti képen tekinthetjük meg. Ha a saját telefonunkon akarjuk megnézni, akkor össze tudjuk kapcsolni azt a Studióval, ehhez szükségünk lesz a telefonunk mobile driver-ére. Ezután a telefonunkon még engedélyezni kell a fejlesztői beállításokon belül az USB hibakeresést. Ezek után tudjuk a programmal összhangba hozni a telefonunkat. Jó játékot!



13.5. Ciklomatikus komplexitás

Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását! Lásd a fogalom tekintetében a https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_2.pdf (77-79 fóliát)!

Először azzal kezdünk, hogy mi is az amiről beszélünk. A Ciklomatikus komplexitás egy szoftvermetrika. Sokszor szoktuk McCabe komplexitásnak is nevezni, alkotója után aki Thomas J. McCabe. Ő 1976-ban publikálta. A komplexitást egy konkrét számértékben fogja kifejezni egy meghatározott szoftver forráskódja alapján, a gráfelméleten alapszik a számítás. A ciklomatikus komplexitás értéke az alábbi:

$$M = E - N + 2P$$

A képletben belül az összefüggő komponensek számát, azaz a 'P'-t megszorozzuk kettővel. Az 'E' jelenti a gráf éleinek számát és az 'N' pedig a csúcsok számát. Egy programban ez azt jelenti, hogy egy gráfot egy függvényben megtalálható utasítások alkotnak. Ha egy utasítás után egyből lefut a másik akkor a két utasítás között él található.

A feladatban én az LZWBinfának számoltam ki a komplexitását. Ehez a Lizard programot használtam. A képen a CCN oszlopban látható az LZWBinf függvényeinek a komplexitása:

```
File Edit View Search Terminal Help
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$ lizard LZWBInf.java
=====
NLOC   CCN   token  PARAM  length  location
-----
3       1     9       0       5 LZWBInfFa:LZWBInfFa@5-9@LZWBInfFa.java
22      4    104      1      28 LZWBInfFa:egyBttFeldolgo12-39@LZWBInfFa.java
4       1     26       0       7 LZWBInfFa:killr@42-48@LZWBInfFa.java
4       1     22       1       4 LZWBInfFa:killr@53-56@LZWBInfFa.java
5       1     21       1       5 LZWBInfFa:Csomopont::Csomopont@62-66@LZWBInfFa.java
3       1     8       0       3 LZWBInfFa:Csomopont::nullasGyermeke@70-72@LZWBInfFa.java
3       1     8       0       3 LZWBInfFa:Csomopont::egyesGyermeke@75-77@LZWBInfFa.java
3       1    11       1       3 LZWBInfFa:Csomopont::ujNullasGyermeke@80-82@LZWBInfFa.java
3       1    11       1       3 LZWBInfFa:Csomopont::ujEgyesGyermeke@85-87@LZWBInfFa.java
3       1     8       0       3 LZWBInfFa:Csomopont::getBetu@90-92@LZWBInfFa.java
15      3    107      2      17 LZWBInfFa:killr@107-123@LZWBInfFa.java
5       1    21       0       5 LZWBInfFa:getMelyseg@132-136@LZWBInfFa.java
6       1    32       0       6 LZWBInfFa:getAtlag@138-143@LZWBInfFa.java
12      2    68       0      15 LZWBInfFa:getSzorasz@145-159@LZWBInfFa.java
11      3    51       1      12 LZWBInfFa:rmelyseg@161-172@LZWBInfFa.java
12      4    66       1      12 LZWBInfFa:ratlag@174-185@LZWBInfFa.java
12      4    78       1      12 LZWBInfFa:rszorasz@187-198@LZWBInfFa.java
3       1    14       0       3 LZWBInfFa:usage@202-204@LZWBInfFa.java
64      14   400      1      95 LZWBInfFa:main@207-301@LZWBInfFa.java
1 file analyzed.
=====
NLOC   Avg.NLOC  Avg.CCN  Avg.token  function_cnt  file
-----
207      10.2     2.4     56.1        19      LZWBInfFa.java
=====
No thresholds exceeded (cyclomatic_complexity > 15 or length > 1000 or parameter_count > 100)
=====
Total nloc  Avg.NLOC  Avg.CCN  Avg.token  Fun Cnt  Warning cnt  Fun Rt  nloc Rt
-----
207      10.2     2.4     56.1        19         0         0.00  0.00
kissmate3@kissmate3-VirtualBox:~/Desktop/Prog2/1.csonag$
```

IV. rész

Irodalomjegyzék

DRAFT

13.6. Általános

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

13.7. C

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

13.8. C++

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

13.9. Lisp

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

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

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