

Univerzális programozás

Igy neveled a programozód!

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

Copyright © 2020 Deák Ruben

Copyright (C) 2019, 2020, 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>

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Bátfai, Margaréta, Ács Deák, Ruben	2020. május 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2020-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2020-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2020-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2020-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2020-02-27	Helló, Turing! csomag kész.	deakruben
0.0.6	2020-03-13	Helló, Chomsky! csomag kész.	deakruben
0.0.7	2020-03-20	Helló, Caesar! csomag kész.	deakruben

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.8	2020-03-27	Helló, Mandelbrot! csomag kész.	deakruben
0.0.9	2020-04-03	Helló, Welch! csomag kész.	deakruben
0.1.0	2020-04-10	Helló, Conway! csomag kész.	deakruben
0.1.1	2020-04-24	Helló, Schwarzenegger! csomag kész.	deakruben
0.1.2	2020-05-01	Helló, Chaitin! csomag kész.	deakruben

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, előadásokat nézzek meg, könyveket olvassak el?	3
II. Tematikus feladatok	5
2. Helló, Turing!	7
2.1. Végtelen ciklus	7
2.2. Lefagyott, nem fagyott, akkor most mi van?	8
2.3. Változók értékének felcserélése	10
2.4. Labdapattogás	11
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	13
2.6. Helló, Google!	14
2.7. 100 éves a Brun tétel	16
2.8. A Monty Hall probléma	16
2.9. Minecraft-MALMÖ bevezető	18
2.10. Vörös Pipacs Pokol/csigafolytonos mozgási parancsokkal	18
3. Helló, Chomsky!	20
3.1. Decimálisból unárisba átváltó Turing gép	20
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	21
3.3. Hivatkozási nyelv	22
3.4. Saját lexikális elemző	22

3.5. Leetspeak	23
3.6. A források olvasása	24
3.7. Logikus	27
3.8. Deklaráció	28
3.9. Vörös Pipacs Pokol/csiga diszkrét mozgási parancsokkal	29
4. Helló, Caesar!	31
4.1. double ** háromszögmátrix	31
4.2. C EXOR titkosító	33
4.3. Java EXOR titkosító	35
4.4. C EXOR törő	36
4.5. Neurális OR, AND és EXOR kapu	38
4.6. Hiba-visszaterjesztéses perceptron	42
4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve	43
5. Helló, Mandelbrot!	44
5.1. A Mandelbrot halmaz	44
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	47
5.3. Biomorfok	50
5.4. A Mandelbrot halmaz CUDA megvalósítása	54
5.5. Mandelbrot nagyító és utazó C++ nyelven	57
5.6. Mandelbrot nagyító és utazó Java nyelven	58
5.7. Vörös Pipacs Pokol/fel a láváig és vissza	61
6. Helló, Welch!	62
6.1. Első osztályom	62
6.2. LZW	66
6.3. Fabejárás	68
6.4. Tag a gyökér	71
6.5. Mutató a gyökér	79
6.6. Mozgató szemantika	80
6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid	90

7. Helló, Conway!	91
7.1. Hangyaszimulációk	91
7.2. Java életjáték	109
7.3. Qt C++ életjáték	118
7.4. BrainB Benchmark	118
7.5. Vörös Pipacs Pokol/19 RF	126
8. Helló, Schwarzenegger!	127
8.1. Szoftmax Py MNIST	127
8.2. Mély MNIST	132
8.3. Minecraft-MALMÖ	132
8.4. Vörös Pipacs Pokol/javíts a 19 RF-en	133
9. Helló, Chaitin!	134
9.1. Iteratív és rekurzív faktoriális Lisp-ben	134
9.2. Gimp Scheme Script-fu: króm effekt	135
9.3. Gimp Scheme Script-fu: név mandala	135
9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-edet	135
10. Helló, Gutenberg!	136
10.1. Programozási alapfogalmak	136
10.2. C programozás bevezetés	141
10.3. C++ programozás	142
10.4. Python nyelvi bevezetés	144
III. Második felvonás	146
11. Helló, Arroway!	148
11.1. A BPP algoritmus Java megvalósítása	148
11.2. Java osztályok a Pi-ben	148
IV. Irodalomjegyzék	149
11.3. Általános	150
11.4. C	150
11.5. C++	150
11.6. Lisp	150

Ábrák jegyzéke

4.1. A double ** háromszögmátrix a memóriában	33
5.1. A Mandelbrot halmaz a komplex síkon	44
6.1. Polargenteszt.cpp futtatása	65
6.2. Inorder fa bejárás	69
6.3. Preorder fa bejárás	70
6.4. Postorder fa bejárás	71
7.1. Hangya szimuláció	92
7.2. Hangya szimuláció	109
7.3. Életjáték	117
7.4. Életjáték	117
7.5. BrainB Benchmark	125

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

Magam is ezeken gondolkozok. Szerintem a programozás lesz a jegyünk egy másik világba..., hogy a galaxisunk közepén lévő fekete lyuk eseményhorizontjának felületével ez milyen relációban van, ha egyáltalán, hát az homályos...

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](#), 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 [\[?\]](#) könyv 25-49, kb. 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket, előadásokat nézzek meg, könyveket olvassak el?

A kurzus kultúrájának élvezéséhez érdekes lehet a következő elméletek megismerése, könyvek elolvasása, filmek megnézése.

Elméletek.

- Einstein: A speciális relativitás elmélete.
- Schrödinger: Mi az élet?
- Penrose-Hameroff: Orchestrated objective reduction.
- Julian Jaynes: Breakdown of the Bicameral Mind.

Könyvek.

- Carl Sagan, Kapcsolat.
- Roger Penrose, A császár új elméje.
- Asimov: Én, a robot.
- Arthur C. Clarke: A gyermekkor vége.

Előadások.

- Mariano Sigman: Your words may predict your future mental health, <https://youtu.be/uTL9tm7S1Io>, hihetetlen, de Julian Jaynes kétkamarás tudat elméletének legjobb bizonyítéka információtechnológiai...
- Daphne Bavelier: Your brain on video games, <https://youtu.be/FktsFcooIG8>, az esporttal kapcsolatos sztereotípiák eloszlatására („The video game players of tomorrow are older adults”: 0.40-1:20, „It is not true that Screen time make your eyesight worse”: 5:02).

Filmek.

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.
- Rain Man, <https://www.imdb.com/title/tt0095953/>, az [?] munkát ihlette, melyeket akár az **MNIST**-ek helyett lehet csinálni.
- Kódjátzsma, <https://www.imdb.com/title/tt2084970>, benne a **kódtörő feladat** élménye.
- Interstellar, <https://www.imdb.com/title/tt0816692>.
- Middle Men, <https://www.imdb.com/title/tt1251757/>, mitől fejlődött az internetes fizetés?
- Pixels, <https://www.imdb.com/title/tt2120120/>, mitől fejlődött a PC?

- Gattaca, <https://www.imdb.com/title/tt0119177/>.
- Snowden, <https://www.imdb.com/title/tt3774114/>.
- The Social Network, <https://www.imdb.com/title/tt1285016/>.
- The Last Starfighter, <https://www.imdb.com/title/tt0087597/>.
- What the #\$*! Do We (K)now!?, <https://www.imdb.com/title/tt0399877/>.
- I, Robot, <https://www.imdb.com/title/tt0343818/>.

Sorozatok.

- Childhood's End, <https://www.imdb.com/title/tt4171822/>.
- Westworld, <https://www.imdb.com/title/tt0475784/>, Ford az első évad 3. részében konkrétan meg is nevezi Julian Jaynes kétkamarás tudat elméletét, mint a hosztok programozásának alapját...
- Chernobyl, <https://www.imdb.com/title/tt7366338/>.
- Stargate Universe, <https://www.imdb.com/title/tt1286039/>, a Desteny célja a mikrohullámú háttér struktúrája mögötti rejtély feltárása...
- The 100, <https://www.imdb.com/title/tt2661044/>.
- Genius, <https://www.imdb.com/title/tt5673782/>.

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

A végtelen ciklusban a feltétel mindig teljesül (true) ezért az utasítások újra lefutnak, így nincs olyan feltétel ami miatt kilépne a ciklus. Például: `for(;;)`

Egy mag 0 százalékban:

```
int
#include <stdio.h>
int
main ()
{
    while(true) {
        sleep(100);

        return 0;
    }
}
```

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 ciklusba a feltételt a `()`-ban adjuk meg és amíg ez igaz addig a `{ }`-ban megadott utasítások végrehajtódnak és a ciklus újra és újra lefut. A példában a ciklus feltétele "true" ami azt jelenti, hogy a feltétel igaz, tehát a ciklus mindig újraindul, amíg ki nem lőjük. 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, jelen esetben 100 ms-ig.

Egy mag 100 százalékban:

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

int main() {
```

```
while (true) {  
}  
return 0;  
}
```

Magyarázat: Ez az előző példához hasonló. Az include-ok 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.

Minden mag 100 százalékban:

```
#include <omp.h>  
#include <stdio.h>  
#include <omp.h>  
  
int main(void)  
{  
    #pragma omp parallel  
    {  
        for(;;)  
        {  
  
        }  
    }  
    return 0;  
}
```

A programunk, az előzőhöz képest egy openmp-vel bővült. #pragma omp parallel sor adja azt az utasítást a gépnek, hogy a feladat az összes szálon fusson, vagyis párhuzamosan minden szálon. (Ezért a fordításnál -fopenmp kapcsoló szükséges még a parancsba.)

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

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

Megoldás videó:

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

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

```
    else
        return false;
}

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

    main(Input Q)
    {
        Lefagy2(Q)
    }

}
```

Mit fog kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulság: Ha a T100-t és T1000-t létező programnak tekintjük és T1000-ben meghívjuk saját magát. A T100 alapján ha a programunkba van végtelen ciklus, akkor igaz (true) értéket ad a Lefagy program a Lefagy2 programnak. Tehát az is igaz (true) értéket fog adni, viszont ha a Lefagy hamis (false) értéket ad vissza akkor a Lefagy2 belép egy végtelen ciklusba és a program le fog fagyni. Olyan program tehát mint a T100, nem működik mivel ha egy olyan program érkezik bele amiben van végtelen ciklus, akkor a program leáll mert a ciklus nem áll meg.

2.3. Változók értékének felcserélése

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

A feladat két változónak az értékeinek felcserélése. Az $x = 2$ és $y = 3$ példában ez nem tűnik nehéznek ha egy segédváltozót használunk aminek megadjuk x -értékét aztán x -nek az y -értékét végül y -nak pedig értékül adjuk a segédváltozó értékét. A következő példában egy másfajta változócsere fogunk alkalmazni, amely számolással cseréli fel az x és y értékeit.

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

int main() {

    int x = 2, y = 3;

    printf("%s\n%d %d\n, kulonbseggel:", x, y);

    x -= y;
    y += x;
    x = y-x;

    printf("%d %d\n", x, y);

    return 0;
}
```

Magyarázat: A fejléct már ismerjük a 2.1-ből. A printf() függvény a kiíratást végzi, benne az első argumentum a a kiíratás formátuma, a többi pedig a változók kiíratása. A „%s” azt jelenti, hogy egy szöveget fogunk kiíratni, amit a „%d” követ amely, egész típusú változót jelent, a „\n” pedig a sortörést jelenti. Aztán egy kis matematikai számítás, végül újra egy kiíratás, hogy megmutassuk, hogy a változók felcserélődtek.

Elsősorban egy kis bevezető: $x += y$ egyenlő $x = x + y$ kifejezéssel (ez a formátum csak egy rövidítés, későbbi programozásban hasznos dolog lesz)

A lépések egyszerűbben a következők: $x = 2$, $y = 3$ $x -= 3$ így az x értéke -1 $y += -1$ ez azt jelenti hogy y -hoz hozzáadjuk az x -et (a -1 -t) így $x = -1$ és $y = 2$ $x = y - x$ azaz $x = 2 - (-1) = 2 + 1 = 3$ tehát az x értéke 3 és az y értéke 2 lett. Így kész is van a csere.

2.4. Labdapattogás

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

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

int main ( void )
{
    WINDOW *ablak;
    ablak = initscr ();

    int x = 0;
    int y = 0;

    int xone = 0 , yone = 0;

    int xmax, ymax;

    for(;;)
    {
        getmaxyx ( ablak, ymax, xmax );

        mvprintw ( y, x, "O" );

        refresh();
        usleep (100);

        x = x + xone;
        y = y + yone;

        if ( x >= xmax-1 ) {
            xone *= -1;
        }
        if ( x <= 0 ) {
            xone *= -1;
        }
        if ( y >= yone-1 ) {
            yone *= -1;
        }
        if ( y >= ymax-1 ) {
            yone *= -1;
        }
    }
}
```

```
}  
  
return 0;  
}
```

Magyarázat: Az új dolog ami a fejlécnél feltuúnik az a curses.h header. Ez képernyő kezelési függvényeket tartalmaz, és a program megjelenítéséhez szükségünk van rá. A main() függvényben a "void" kifejezés azt jelenti, hogy csak megjelenítünk a képernyőn valamit.

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

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

A deklarált x-en és y-on lesz a kezdő értékünk. Az xone és yone pedig a lépésközöt mutatja, jelen esetben 1. (lépésenként a koordináta rendszeren xone, yone-al való elmozdulást). Az xmax és ymax lesznek a határértékek, hogy a program csak az ablakon belül mozogjon.

A végtelen ciklus miatt a labda pattogás nem áll ki nem löjünk a programot. A getmaxyx() függvény meghatározza az ablak méretét, a refresh() függvény pedig az ablakot frissíti. A mvprint() függvény az x és y koordináta tengelyen megjeleníti jelen esetben az "O" karaktert. A usleep() függvény altatja a ciklust, azaz mennyi időn belül induljon újra a ciklus, tehát a labda pattogásának sebességét is ezzel megadjuk.

```
x = x + xone;  
y = y + yone;
```

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 így a labda irányt változtat. A fordításnál -lncurses kapcsolót is kell használnunk a fejlécben megjelenő curses.h miatt.

Egy másik megoldás az ""if" logikai feltételek használata nélkül:

```
#include <stdio.h>  
#include <stdlib.h>  
#include <curses.h>  
#include <unistd.h>  
  
int  
main (void)  
{  
    int xj = 0, xk = 0, yj = 0, yk = 0;  
    int mx = 80 * 2, my = 24 * 2;  
  
    WINDOW *ablak;  
    ablak = initscr ();  
    noecho ();  
    cbreak ();  
    nodelay (ablak, true);  
  
    for (;;)   
    {  
        xj = (xj - 1) % mx;
```

```
        xk = (xk + 1) % mx;

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

        clear ();

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

        refresh ();
        usleep (1)
    }
    return 0;
}
```

Magyarázat: A programunk ugyan azt csinálja mint az "if"-es változata. Csak ugye most logikai kifejezés, utasítás nélkül. A megoldáshoz szükségünk van matematikai számításokra, ehhez deklarálunk egész típusú változókat. A számításokat egy végtelen ciklusban számoljuk és mvprintw-val írjuk ki a képernyőre. A clear()-el minden egyes számítás előtt letisztítjuk az ablakot, az első kettő mvprintw-val a felső és alsó határokat rajzoljuk ki, a harmadikkal pedig a "Labdát". Az usleep() függvény itt is a pattogás sebességét határozza meg.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

```
#include <stdio.h>

int main()
{

    int x = 1;
    int bit = 0;

    do
    bit++;
    while (a<=1);
```



```
printf("%d %s\n", bit, "bites a szohossz.");  
}
```

A fejléct ismerjük, a main() függvény a fő függvényünk, amelyben megadunk egy változónak egy tetszőleges értéket, jelen esetben 1-t. A bit változó számolni fogja, hogy hányszor fut le a ciklus. A programot hátultesztelő ciklussal do { } while() ciklussal futtatjuk, mivel az előltesztelő while() ciklus nem számolná bele az első lépést.

A ciklus addig fut újra és újra amíg az x értéke nem 0. Tehát az x értéke kezdetben 1, a bináris értéke pedig 0001, a << (bitshift) operátor csak annyit csinál, hogy egy 0-val eltolja az 1-et, tehát a 0001-ből egy lépés után 0010 lesz ami 2, továbbá a második lépés után 0100 ami 4. A ciklus tehát addig fut amíg csupa 0 érték lesz a gépünk szóhossza, azaz az 1-est kitolja a szóhosszból, így az értékünk 0 lesz, a while ciklus befejeződik és a printf kihatja a bit értékét vagyis hogy hányat lépett az 1-es (hányszor futott le a ciklus), ez aszám megadja hogy hány bites a szóhossz (jelen esetben az int típus 32 bites lesz),

2.6. Helló, Google!

A PageRank egy keresőmotor a Googleban. A programot két fiatal írta meg 1998-ban, nevét az egyik kitalálója Larry Page után kapta.

A következőben, egy 4 weblapból álló PageRank-et 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 k;  
    for (k = 0; k < db; ++k)  
        printf ("%f\n", tomb[k]);  
}  
  
double tavolsag (double PageR[], double PageRmatrix[], int n)  
{  
    double osszeg = 0.0;  
    int i;  
  
    for (i = 0; i < n; ++i)  
    {  
        osszeg += (PageRmatrix[i] - PageR[i]) * ( ←  
            PageRmatrix[i] - PageR[i]);  
    }  
    return sqrt(osszeg);  
}  
  
int main (void)  
{  
    double Honlap[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 PageR[4] = { 0.0, 0.0, 0.0, 0.0 };

double PageRmatrix[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)
    {
        PageR[i] = 0.0;
        for (j = 0; j < 4; ++j)
        {
            PageR[i] += (Honlap[i][j] * PageRmatrix[j]);
        }
    }

    for ( i = 0; i < 4; ++i )
    {
        PageRmatrix[i] = PageR[i];
    }
}

kiir (PageR, 4);

return 0;
}
```

A `math.h` header tartalmazza a matematikai számításokhoz szükséges függvényeket. A `main()` függvényben létrehozunk egy mátrixot, ami a lapok összeköttetését adja meg. Ha az érték 0 akkor a lap nincs összekötve az adott lappal és önmagával sincs. Ahol $1/2$ vagy $1/3$ az érték az oldal másik oldallal való összeköttetését jelenti.

Például az $1/2$: Az oldal 2 oldallal van összekötve.

A `PageR` tömb fogja a PageRank értéket tárolni. A `PageRmatrix` tömb pedig a mátrixal való számításokhoz kell. A következő lépés egy végtelen ciklus, ez majd a számítások végén a "break" parancsal lép ki, ha a megadott feltétel teljesül.

A for ciklusban van maga a PageRank számítása ami a `tavolsag()` függvényt is meghívja, és egy részszámlást tartalmaz. A végtelen cikluson belül lévő ciklusok azért 4-ig mennek mert 4 weblapot nézünk. A ciklusból a "break" parancsal lépünk ki ha a `tavolsag()` függvényben kapott eredmény kisebb mint 0.00000001. A végén a `kiir()` függvény megkapja a `PageR` értékeket és az weblapok számát és kiírja.

2.7. 100 éves a Brun tétel

A tétel kimondja hogy az ikerprímek reciprokösszege a Brun konstanthoz konvergál, ami egy véges érték. A tételt Viggo Brun-ről nevezték el a tételt aki bebizonyította 1919-ben.

Megoldás forrása: <https://github.com/RubiMaistro/Prog1/blob/master/burn.R>

Az R nyelvű matlab könyvtár telepítési parancsai:

```
sudo apt-get install r-base
```

```
sudo apt-get install libopenblas-base r-base
```

```
sudo apt-get install gdebi
```

```
cd ~/Downloads
```

A végtelen cikluson belül lévő ciklusok azért 4-ig mennek mert 4 weblapot nézünk. A ciklusból a "break" paranccsal lépünk ki ha a tavolsag() függvényben kapott eredmény kisebb mint 0.00000001. A végén a kiir() függvény megkapja a PageR értékeket és az weblapok számát és kiírja. rstudio-xenial-1.1.379-amd64.deb

A számoláshoz kell egy matlab könyvtár. A program fő része az stp függvény, a függvény megkapja-et. X egy szam lesz ami megmondja meddig kell a prímeket számolni. Ehhez a primes függvényt használjuk. A 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-es prímszámtól indulva kivonjuk a prímből az előtte lévő prímét. 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 vektorbapedigamiezeknélaprímeknélkettővelnagyobb(azaz ikerprímek). rt1plust2 vektorban végezzük a reciproképzést és a pár reciprokát összeadjuk. A returnban pedig a sum függvénnyel vissza adjuk ezek összegét. Végül a plot() függvénnyel lerajzoljuk grafikusán.

2.8. A Monty Hall probléma

Bevezetés a megértéshez: A kérdés vagy probléma egy vetélkedő játékból indul, amelyben van 3 ajtó és az egyik mögött egy értékes autó van, a másik kettő mögött 1-1 kecske, és amelyiket választja a játékos azt nyereményként megkapja. A versenyzőnek a 3 ajtó közül választania kell vagy sem.

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

Magyarázat: Kezdetben 3 ajtó közül 1 ajtót kell választanunk, azaz 1/3 az esélye, hogy eltaláljuk a jó megoldást és 2/3 hogynem. Ezek után a műsorvezető kinyit egy ajtót ami mögött nincs a nyeremény. Ez a valószínűsége nem változtat, ugyanúgy 1/3 eséllyel választottuk azt az ajtót ami mögött a nyeremény van. Viszont azok az ajtók közül ami mögött nincs semmi, már csak az egyik van csukva. Biztosra tudjuk, hogy a nyeremény a maradék két ajtó közül valamelyik mögött van. Tehát 2/3 az esélye annak, hogy a másik ajtó mögött van a nyeremény, mivel ha elsőre azt az ajtót választottuk amelyik mögött egy kecske van, és amikor megkapjuk a változtatásra a lehetőséget és élük a lehetőséggel és a másik ajtót választjuk, akkor biztosan az autót megnyerjük.

A problémával kapcsolatban egy R nyelvben írt szimuláció a következő:

```
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 )

}
```

A kísérletet 10000x fogjuk eljátszani, a kísérlet vektorban 1 és 3 "ajtó" közül választunk 10000x. A `replace=T`-vel tesszük lehetővé, hogy egy eredmény többször is kijöhessen. A játékos választásait a játékos vektornál ugyan így meghatározzuk. A `sample()` függvénnyel végezzük a kiválasztást. A műsorvezető 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`-el megvizsgáljuk, hogy a játékos által választott ajtó megegyezik-e a kísérletben szereplő ajtóval. Ha a feltétel igaz egy vektorba bele tesszük azokat az ajtókat amiket a játékos nem választott, az `else` ágon pedig ha a feltétel nem igaz, akkor azt az ajtót eltávolítjuk amit nem ő választott és a nyereményt rejtő ajtót.

A műsorvezető vektorban pedig azt az ajtót amit ki fog nyitni. A `nemvaltoztat` és `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 `valtoztat` vektorban pedig azt mikor megváltoztatja a döntését és így nyer ezt egy `for` ciklussal vizsgáljuk. A legvégén kiíratjuk az eredményeket, hogy melyik esetben hányszor nyert.

2.9. Minecraft-MALMÖ bevezető

Egy kis bevezető tájékoztatást szeretnék adni a MALMÖ projekttel kapcsolatban amellyel a továbbiakban nagyon érdekes és különleges feladatokat oldhatunk meg.

A **MALMÖ** egy kreativitást igénylő projekt melynek az alapja a nagy többség által ismert **Minecraft** nevű játék. A Minecraft Malmö projekt a Mojang fejlesztése, melynek első változatát **2016** júliusában publikáltak, a projekt a mesterséges intelligenciára alapoz.

A projekt vezetői: **Katja Hofmanna** fő kutató, **Andre Kramer** kutató mérnök és még sok más kutató. Céljuk a projekttel, hogy a Mesterséges Intelligencia ágát egy új környezetbe építve fejlesszék, kutassák és hogy sokan beszálljanak a projektbe, ezért is választották a Minecraft nevű játékot, ami nagy ismeretkörrel rendelkezik és ideális teret és körülményeket ad ennek a projektnek.

A projektben számtalan kreatív lehetőséggel rendelkezünk ahogyan ezt a projekt vezetői is említik. A lehetőségünk az ágens irányításával kezdve a különböző blockok azonosításával és a környezet felismerésével a blockok mozgásán át, a különböző szituációkban, az npc és mob közelség reakcióáig és még tovább egy kreatív programozó és fan számára kimeríthetetlen lehet.

Ebben könyvben a MALMÖ projekt feladataiban csak a projekt egy minimális részét fogjuk érinteni.

A **Red Flower Hell** repóban különböző érdekes programkódok elérhetők.

Ebben az évben a DE-IK PTI karon a Red Flower Hell (RFH) MALMÖ projekttel foglalkozunk. Ennek célja, hogy a hallgatókat a mesterséges intelligencia kutatása és programozása felé ösztönözze, melynek nem tudhatjuk mennyire hasznos értéke lesz a mesterséges intelligencia tudomány fejlődésére nézve.

A **MALMÖ projekt** hivatalos oldala itt elérhető ahol több és részletesebb információt kaphatunk a projekttel kapcsolatban.

Valamint amikor már elérünk a Schwarzenegger fejezethez, kaphatunk egy rövid összefoglalót is az általunk kidolgozott projekt részekből.

2.10. Vörös Pipacs Pokol/csigá folytonos mozgási parancsokkal

Megoldás videó: <https://youtu.be/uA6RHZxH840>

Megoldás forrása elérhető a következő linken:

<https://github.com/nbatfai/RedFlowerHell>

Link saját repóból: [Csigá mozgása folytonosan](#)

A feladat alapja, hogy Steve az ágens, csigá vonalban haladva jussok minél feljebb a tölcészerű aréna aljáról a tetejére ameddig a láva engedi ami ugyanis folyik fentről lefelé és Steve amint a lávát eléri meghal.

A megoldásához folytonos mozgást használunk. A **self.agent_host.sendCommand(utasítás)** parancs szükséges ahol az utasítás helyen kell megadni Stevenek mit csináljon. Ezek a mozgást leíró utasítások, melyek paraméterét alapvetően, úgy kell használni mint a kapcsolókat, ha 0 az értéke nem, ellenben ha 1 akkor elvégzi a mozgást.

Minden szint falának elérésekor ugrik egyet a **jumpmove** parannccsal, és jobbra fordulva megy tovább. Minden újabb körben feljebb-feljebb jut Steve és így a körök is egyre hosszabbak lesznek. A mozgás időtartamát

a **time.sleep()** paranccsal adhatjuk meg, azaz az argumentumában megadott másodpercig fut. Minden kör egyértelműen 4 hosszú előre haladásból áll, ez a mozgás alapja, ezt kell kiegészíteni a fordulásokkal és ugrásokkal.

DRAFT

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

A gép a nevét Alan Truring után kapta 1936-ban. A gép decimális számrendszerből unáris számrendszerbe írja át a számot. Az unáris számrendszer másnéven egyes számrendszer. A gép úgy működik, hogy csak 1-eseket ír. Tehát például a 6-ost átírva 6 darab 1-est ír le unáris számrendszerbe átváltva, az 1-es helyett csak egy 1-est ír.

A Turing gép kódja c++ nyelven 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;
        ++szaz;
        if (tiz==10) {
            cout<<" "; tiz=0;
        }
        if (szaz==100){
            cout<<"\n"; szaz=0;
        }
    }
    return 0;
}
```

A kódban egyszerűen csak egy fő függvényt használtunk a main()-t, ebben bekértünk egy tetszőleges számot és ezt alakítjuk unárisba. Egy for ciklust meghatároztuk, hogy 0-tól induljon (i=0) és egészen addig fusson amíg eléri azt a számot amit megadtunk

```
(i<a)
```

, persze minden lefutása után eggyel ($i++$) növekedjen. A for ciklusban mindig kiíratunk egy 1-est, a két segédváltozót csak az áttekinthetőség miatt használjuk, tehát amikor már kiíratunk egymás után 10 darab 1-est iratunk egy szóközt, ha 100 darab 1-est akkor egy sortörést.

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

A generatív nyelvek kidolgozása Noam Chomsky nevéhez fűződik. A nyelveket osztályokba rendezzük, vannak erősebb és gyengébb osztályok és az erősebb osztály képes létrehozni gyengébb osztályt. Négy darab alapon fekszik a generatív nyelvtan:

- Terminális szimbólumok, a konstansok.
- Nem terminális jelek, a változók.
- Kezdőszimbólum, egy kijelölt szimbólum.
- Helyettesítési szabályok, ezzel a szavakat értelmezzük majd.

Legyenek a nyelv változói:

```
X Y Z
```

És legyenek a nyelv konstansai:

```
a b c
```

A helyettesítési szabályok:

```
X→abc, X→aYbc, Yb→bY, Yc→Zbcc, bZ→Zb, aZ→aaY, aZ→aa
```

```
X (X→aYbc)
```

```
aYbc (Yb→bY)
```

```
abYc (Yc→Zbcc)
```

```
abZbcc (bZ→Zb)
```

```
aZbbcc (aZ→aa)
```

```
aabbcc
```

```
X (X→aYbc)
```

```
aYbc (Yb→bY)
```

```
abYc (Yc→Zbcc)
```

```
abZbcc (bZ→Zb)
```

```
aZbbcc (aZ→aaY)
```

```
aaYbbcc (Yb→bY)
```

```
aabYbcc (Yb→bY)
```

```
aabbYcc (Yc→Zbcc)
```

```
aabbZbcc (bZ→Zb)
```

```
aabZbbcc (bZ→Zb)
```

```
aaZbbbcc (aZ→aa)
```

```
aaabbbcc
```


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

3.3. Hivatkozási nyelv

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");
}
```

A program egyszerű, kiírja a for ciklus a "Lefut" szöveget. A kódot viszon több nyelvtanban is fordíthatjuk. Ha a C89-es nyelvtannal fordítjuk a kódot akkor "gcc -std=gnu89 fajnev.c -o fajlnev"-et használom. Ekkor a program hibát fog kiírni a for ciklusnál, mert a C89-es nyelvtanban a for cikluson belül deklaráljuk az i változót, mert ebben a régebbi nyelvtanban még erre nincs lehetőségünk, csak cikluson kívül.

De viszont ha C99-es nyelvtannal fordítjuk "gcc -std=gnu99 fajnev.c -o fajlnev"-et használom, akkor a kód hiba nélkül lefut, mivel ebben már szerepel az a lehetőség, hogy cikluson belül deklarálhatunk változókat a C89-es nyelvtannal ellentétben.

Tehát a tanulság, hogy egy programozási nyelven belül sem mindegy milyen típusú nyelvtannal fordítjuk a kódunkat.

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 számokat változóval számoljuk, hogy 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. Leetspeak

Tutor: Kikina Dominik

Lexelj össze egy l33t ciphert!

```
% {  
  
    #include <string.h>  
    int szamok=0;  
} %  
  
%%  
"0" { printf("O"); }  
"1" { printf("I"); }  
"2" { printf("Z"); }  
"3" { printf("E"); }  
"4" { printf("A"); }  
  
"5" { printf("S"); }  
"6" { printf("b"); }  
"7" { printf("T"); }  
"8" { printf("B"); }  
"9" { printf("P"); }  
  
"O" { printf("0"); }  
"I" { printf("1"); }  
"Z" { printf("2"); }  
"E" { printf("3"); }  
"A" { printf("4"); }  
  
"S" { printf("5"); }
```

```
"b" { printf("6"); }
"T" { printf("7"); }
"B" { printf("8"); }
"P" { printf("9"); }

%%

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

Ez a nyelv lefordítja a l33t nyelven írt szöveget vagy a l33t nyelvre írja át. A program működése hasonló az előzőhöz, csak itt a megadott számokat keresi és helyettük a l33t nyelvben a nekik megadott megfelelő betűt írja helyette. Az ellenkező esetben pedig ha l33t nyelvre akarjuk átírni a szöveget, akkor a megadott betűket keresi és alakítja át számokká.

3.6. A források olvasása

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

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

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



Bugok

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

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

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

```
int i=0;

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

```
(1) (2) (3) (4) (5)
Process returned 0 (0x0)   execution time : 0.069 s
Press any key to continue.
```

Ez egy for ciklus, a benne lévő *i* változó kezdőértéke 0, a ciklus addig fut le újra és újra amíg az *i* értéke kisebb mint 5. Ebben az esetben az *i* értékét még a lefutás előtt mindig növeljük 1-el, mivel "++i"-ként használjuk, ebben az esetben az *i*=0-ra ha teljesül a feltétel akkor az *i*=1-el (tehát mindig a rákövetkezőjére) fut le a ciklus.

```
int i=0;

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

```
(0) (1) (2) (3) (4)
Process returned 0 (0x0)   execution time : 0.100 s
Press any key to continue.
```

Ez a for ciklus hasonló az előzőhöz, a különbség viszont az, hogy az *i* értékét majd csak a ciklus lefutása után növeljük 1-el, ebben az esetben pedig *i*=0-ra ha teljesül a feltétel akkor szintén *i*=0-val fut le a ciklus.

```
int i=0;
int tomb[6];

for(i=0; i<5; tomb[i] = i++){
    printf("%d ", tomb[i]);
}
```

```
4200816 0 1 2 3
Process returned 0 (0x0)   execution time : 0.062 s
Press any key to continue.
```

Ez a for ciklus egy tömböt feltölt az *i* értékével. Nagyon érdekes módon történik ez mivel a tömb első eleme memóriaszemét lesz, ez azért történik meg mert amikor *i*=0-tól indul a for ciklus a *tomb* is az első eleménél az az a 0.-nál tart. Bug: A for cikluson belül semmi nem történik ezért az első értékadás a *tomb*-be úgy mond nem sikerül, majd végrehajtódik a for ciklus harmadik utasítása a léptetés, ahol értéket adunk a *tomb*-be és az *i* 1-el növekszik, ezért a for ciklus minden futásnál megkapja az aktuális *i* értékének az előző futás kezdőértékét, ezért 0-tól kezdve felvesz minden *i*-t 0-tól indulva, de a *tomb* utolsó eleme 3 lesz, mert 5-ig alaphoz nem megy el.

```
int i=0, n=10, a=10;
int *d, *s;
*d = a;
```

```
s = d;
for(i=0; i<n && (*d++ = *s++); ++i)
{
    printf("%d  %d  %d\n",i,s,d);
}
```

```
0 2945028 2945028
1 2945032 2945032
2 2945036 2945036
3 2945040 2945040
4 2945044 2945044

Process returned 0 (0x0)   execution time : 0.069 s
```

Ebben az esetben a for ciklusban két feltétel van, tehát akkor fut le a for ciklus ha mind a két feltétel teljesül. Az első feltétel, hogy i kisebb mint az n . A második feltétel, hogy a d és az s mutató egyenlő, és minden ciklusnál növeljük a értékeket. A két feltételt és operátorral fűzzük össze. Bug: A második feltételt célszerűbb lenne a for cikluson belül if-ekkel vizsgálni, mert nem logikai feltétel.

```
int f(int a, int b){
    return a+b;
}
int main(){

    int a = 0;
    printf("%d %d %d", f(a,++a), f(++a,a), a);
    return 0;
}
```

```
4 2 2
Process returned 0 (0x0)   execution time : 0.137 s
Press any key to continue.
```

A `printf()` függvény kiírja az összeget a képernyőre. Itt két egész típusú (%d) változót, az `f()` függvénnyel határozzuk meg a számot, a hiba csak az, hogy nem megfelelő a sorrend.

```
int f(int a){
    a+=a;
    return a;
}
int main(){

    int a = 1;
    printf("%d %d", f(a), a);
    return 0;
}
```

```
26 13
Process returned 0 (0x0)   execution time : 0.154 s
Press any key to continue.
```

A printf() függvény kiír két egész számot, az első számot az "a"-t a f() függvény határozza meg itt 2 lesz és a második pedig maga az "a" változó értéke az 1.

```
int f(int a){
    return a;
}
int main(){

    int a = 1;
    printf("%d %d", f(&a), a);
    return 0;
}
```

```
6422300 214
Process returned 0 (0x0)   execution time : 0.078 s
Press any key to continue.
```

A printf() függvény kiír két egész számot, amiben eltér az előző kódcsipettől, hogy itt az f() függvény az a változó memóriacímét kapja meg.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

$(\forall \text{forall } x \ \exists \text{exists } y \ ((x < y) \wedge (y \ \text{text}\{ \text{prím}})) \wedge (Ssy \ \text{text}\{ \leftarrow \text{prím}})))$

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

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

A természetes nyelvet mi emberek megértjük és ennek függvényében gondolkodunk és cselekszünk, viszont létre kellett hoznunk egy nyelvelt jelen esetben az Ar nyelvet amellyel a számítógépünkkel is tudunk kommunikálni. Egyszerűen csak meg kell tanulnunk melyik parancs mit jelent, az Ar nyelv egy komplex nyelv. Vannak benne logikai összekötőjelek (például: és = \wedge, nem = \neg, vagy = \vee, implikáció(ha

$A, akkor\ B) = \supseteq$). Vannak kvantorok (például: "létezik" = `\exists` és a "minden" = `\forall`). Az "S" értéknövelés, a kiírást pedig a `\text`-el végezzük.

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

```
#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 egész típusú mutatót deklarál ami tárolja az a memóriacímét (azaz "b" mutat "a"-ra).

```
int &r = a;
```

Az r egész típusú változó már létezik és itt referenciát deklarálunk, a referencia más szóval paraméter érték átadás a már meglévő változó értékét jelen esetben megvázolatjuk és értékül kapja az "a" értékét.

```
int c[5];
```

Egy egész típusú 5 elemű tömböt deklarál.

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

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

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

Egy egész típusú 5 elemű tömböt deklarál melynek az összes eleme egy-egy mutató.

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

Az egész típusú változó visszatérési értékét tartalmazó függvény;

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

Egy egész típusra mutató visszaadó függvényre mutató mutató.

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

Egy egész típusú változót visszaadó és két egész típusú változót kapó függvényre mutató mutatót visszaadó egész típusú változót kapó függvény.

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

Egy függvényt mutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó egészet kapó függvény.

3.9. Vörös Pipacs Pokol/csiga diszkrét mozgási parancsokkal

Megoldás videó: <https://youtu.be/Fc33ByQ6mh8>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Ez a feladat az előző Malmós feladathoz hasonló, mivel itt is az a cél hogy Steve csiga vonalban haladjon egyre feljebb az arénában, amíg eléri a látát. Amiben változik itt a feladat az csak annyi, hogy diszkrét mozgást végezzen, tehát előre meghatározott lépést haladjon és meghatározott időben és mennyiségben ugorjon és forduljon.

Amit meg kell adnunk az, hogy egy bizonyos gyűrűn való csiga járást négy részre bontjuk és csak egy résszel foglalkozunk mert az aránya szimmetrikus, és ezeket az előre haladásokat fordulásokkal és esetleges ugrásokkal kötünk össze.

A diszkrét parancsok a következők:

- egy kockányi ugrás: **move 1**
- ugrás közbeni mozgás: **turn 1**
- egy fordulat jobbra: **jumpmove 1**

A **time.sleep()** parancs a mozgás időtartamát befolyásolja, viszont a kiadott parancs mennyisége nem fog változni.

A program kód elérhető a következő link: [csiga_diszkreten](#)

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

A következő programban egy alsó háromszögmátrixot hozunk létre. Forrás: <https://gitlab.com/nbatfai/bhax/blob/r>

Saját megoldás videó: https://www.youtube.com/watch?v=vKI3Ri_BKtg

```
#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");
}

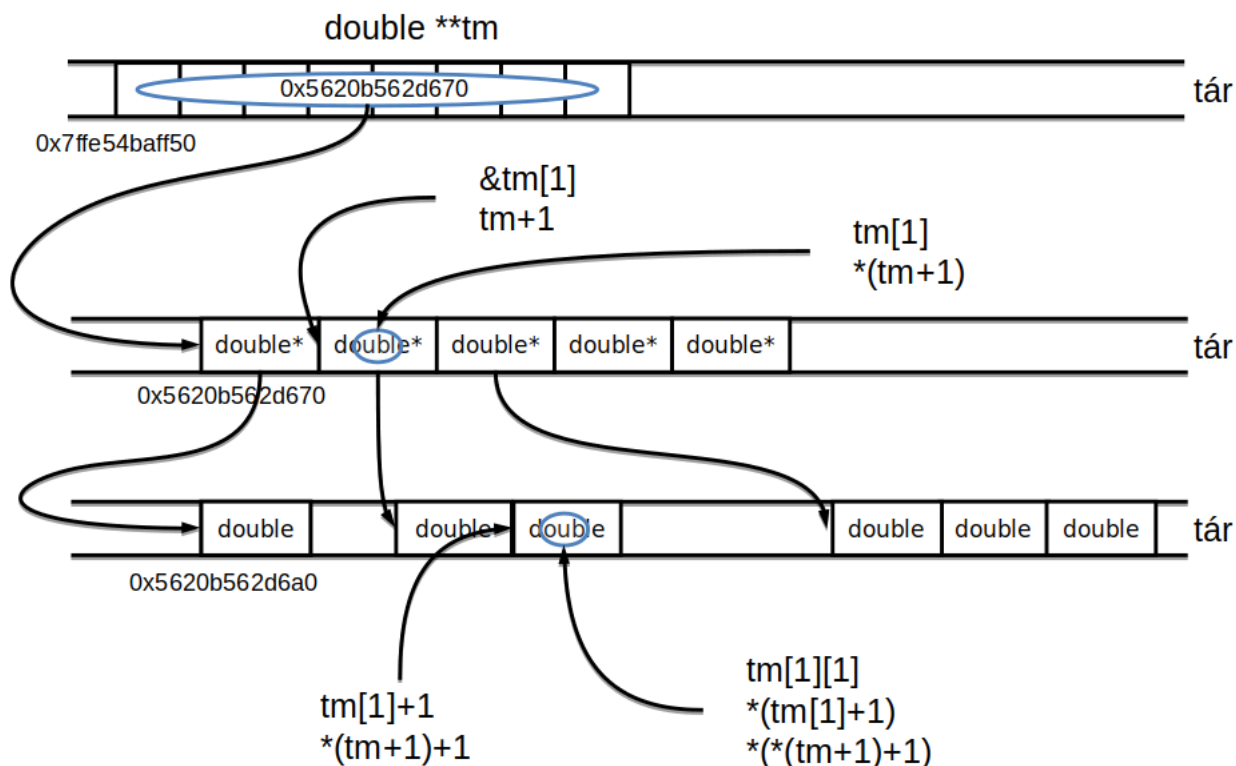
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

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

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

free (tm);

return 0;
}
```



4.1. ábra. A `double **` háromszögmátrix a memóriában

Magyarázat: Includeoljuk a szükséges headereket és a `main()` főfüggvényben dolgozunk tovább. Az első változó az `nr`, amely értéke meghatározza hogy hány soros legyen a kimenet. A `double **tm` sorral foglalunk le tárhelyet a memóriában. Az első `if`-ben 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-t, a harmadiknak 3-t, egészen az `nr` ig. A 3. for ciklussal megadjuk a mátrix elemeit. Az `i` a mátrixnak a sorai, a `j` pedig a benne lévő mutatók. A `tm[i][j]=i*(i+1)/2+j`-vel érjük el azt, hogy az elemek mindig egyel nőjenek. A 4. for ciklus pedig a kírítás.

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 memória szivárgást.

4.2. C EXOR titkosító

A feladat lényege, hogy egy szöveget titkosítsunk Exor-ral(XOR).

Az XOR (kizáró vagy) művelet biteket vizsgál, tehát ha a bitek azonosak (0,0; 1,1) akkor 0-t ad vissza értéknek, ha pedig különbözőek (1,0;0,1) akkor 1-et.

Forrás: https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063_01_paruhuzamos_prog_linux/ch05s02.ht
IwAR2X9zgwtSH6GW2_K67UrxjYDAVgljqV0i5KmBHuAaZ2DjWlvyFW4LrCtA

```
#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);
    }
}
```

Magyarázat:

Includeoljuk a szükséges headereket, aztán két állandó (globális) változót definiálunk a #define paranccsal, ezek értéke nem változik. Az első állandó a MAX_KULCS, az értéke 100, a második pedig a BUFFER_MERET, az értéke pedig 256.

A fő függvényben egy-egy char típusú tömböt hozunk létre amelyek méretei a globális változók(100 és 256). Aztán 2 egész típusú változót hozunk létre, a kulcs_index, ami a kulcsunk aktuális elemét tárolja, és az olvasott_bajtok ami a beolvasott bájtokat tárolja. Egy további egész típusú változó a kulcs_merete, a változóban a kulcs méretét adjuk meg a strlen() függvény segítségével, amit mi adunk meg egyik argumentumként. Az strncpy függvényt pedig a kulcs kezeléséhez használjuk.

Ezután a while ciklusban beolvassuk a buffer tömbbe a bemenetet, a while ciklus addig fut, ameddig van mit beolvasni. A read() függvényel lépünk ki a ciklusból. A while cikluson belül lévő for ciklusban végig megyünk az összes bajton és így titkosít a program.

A fordítás: gcc fajlnev.c -o fajlnev miután lefut, utánna ./fajlnev 23134012 (ez a kulcs) titkositando.txt > titkos.szoveg (titkosított fajl). A titkos szöveget, a more titkos.szoveg paranccsal nézhetjük meg

4.3. Java EXOR titkosító

Tutorált: Talinger Mark Imre

Az előző feladathoz hasonló, a különbség csak a használt programozási nyelv. Itt a Java programozási nyelvet használjuk ami objektum orientált, vagyis objektumokból, osztályokból áll.

Hasonló a C++-hoz, de egyszerűbb mivel csak osztályokat (Classokat) használunk, az osztályokban különböző függvények vannak. Az osztályokat három részre oszthatjuk. A public rész, az ebben lévő függvényeket bárholnan meghívhatjuk. A private rész, az ebben lévő függvényeket csak az adott osztályon belül hívhatjuk meg (az osztály titkos függvényei). A protected rész hasonló a privatehez de van egy kis különbség, a függvényeket csak az osztályban hívhatjuk meg, kivéve ha barát függvényként definiáljuk őket, ez utóbbi esetben meghívhatóak bárholnan.

```
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, 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. A `throw()` függvény hiba üzenetet ad vissza ha rosszul adtunk meg valamit.

A titkosítási eljárás ugyan az mint az előző feladatban, XOR-al történik. Ebben a nyelvben van `byte` típus ami 8 bit. A kulcs és a buffer `byte` típusú lesz, amik tárolják a kulcsot és a beolvasott szöveget.

Mivel java nyelv ezért a `main()` függvény az osztály része és egyben az egyik függvénye. A `main()` függvényt jellemezzük, a `public` (azaz nyilvános), `static` (azaz része az osztálynak) és `void` (amely kiíratást végez). A `main()`-be terminálból is adhatunk értéket. A függvényben pedig van egy `try()` és egy `catch()` függvény, a `try()` hiab üzenetet küld és a `catch()` ezt elkapja aztán kiírja.

A fordításhoz java fordítót kell használnunk, jelen esetben `javac`-t ha ez nekünk nincs telepítve akkor jelzi a számítógép hogyan telepíthetjük.

Fordítás: `javac ExorTitkosito.java`

Futtatás: `java ExorTitkosito titkosítandó.szöveg > titkosított.szöveg`

4.4. C EXOR törő

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

Ebben a feladatban írunk egy programot ami feltöri az előző 4.2-es feladatban titkosított szöveget. A program működése azon az elven alapszik, mint a 4.2 mivel ugyan így XOR-t használunk és ezzel alakítjuk vissza a szöveget. A kulcsot amivel titkosítottunk azt ismernünk kell, mert ezzel a kulccsal tudjuk feltörni. Úgy működik, hogy a titkosított bájtokat össze exortáljuk a kulccsal, és így újra az eredeti bájtokat kapjuk.

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

```
#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(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;

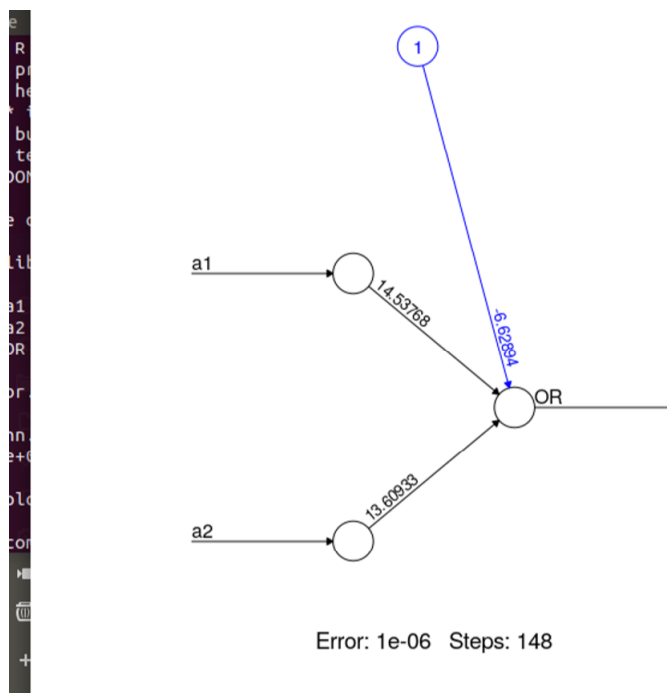
    while((olvasott_bajtok = read(0,
                                (void *) p,
                                (p-titkos+OLVASAS_BUFFER < ←
                                MAX_TITKOS) ?
                                OLVASAS_BUFFER      : titkos+ ←
                                MAX_TITKOS-p)
      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

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

    //A kulcs
    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;
```



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



A program elején meghívjuk a neuralnet könyvtárat ami tartalmazza a szükséges függvényeket. A bemenet az a1 és az a2 lesz, a gép pedig az OR-t (logikai VAGY) fogja megtanulni. Ha a1 és a2 bemenet 0-t ad, akkor az OR értéke is 0 lesz, minden más esetben az OR értéke 1. Ezeket az or.data-ban tárolja el a program. Az nn.or értékét pedig a neuralnet() függvényel határozzuk meg. A függvény:

Az első argumentumában a megtanulandó érték van, azaz hogy az OR értéke 0 legyen vagy 1.

A második argumentumban adjuk meg az or.data-t, hogy mi alapján tanulja meg a program.

A harmadik argumentumban rejtett neuronok száma van.

A stepmax a lépésszámot adja.

A plot függvénnyel kirajzolunk (a képen) a tanulás folyamatának egyik esetét.

Logikai ÉS (AND):

```
library(neuralnet)

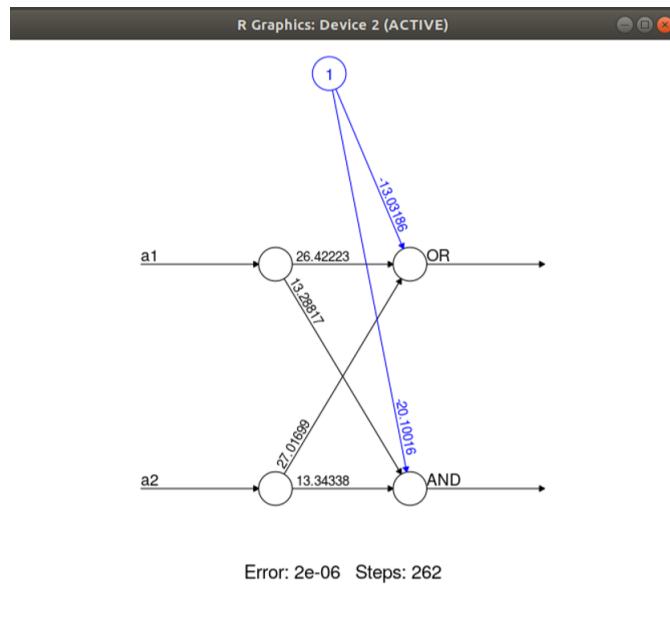
a1 <- c(0,1,0,1)
a2 <- c(0,0,1,1)
OR <- c(0,1,1,1)
AND <- c(0,0,0,1)

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 annyival bővül, hogy a program az OR-on kívül az AND-et (logikai ÉS) is meg fogja megtanulni. Az AND csak akkor kap 1 értéket, ha a1 és a2 értéke is 1, különben az AND értéke 0. A tanulás folyamata ugyan olyan mint az előző programban. A tanulás módját az orAnd.data-ba mentjük.

EXOR:

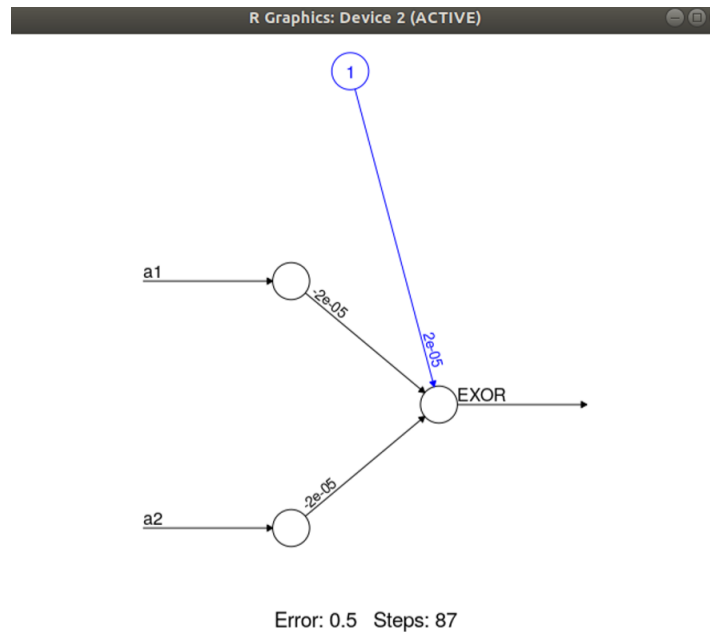
```
library(neuralnet)

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 értéke akkor 1 (igaz), ha az a1 és a2 értéke 1,0 (a1= igaz, a2= hamis) vagy 0,1 (a1= hamis, a2= igaz). Ha mindkét érték 0,0 (hamis, hamis) vagy 1,1 (igaz, igaz) akkor az EXOR értéke 0 lesz. Ezt a tanulási mintát az exor.data-ban mentjük el. És a tanulás megegyezik a fentiekkel. 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.

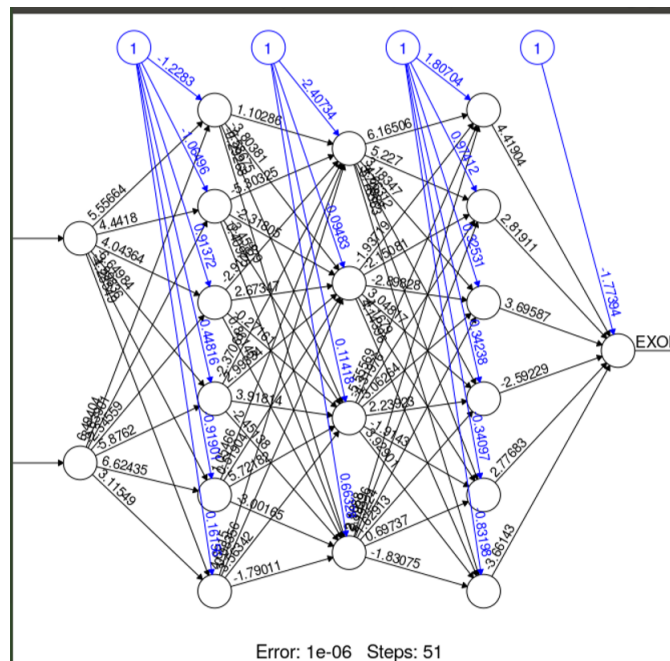
A 3 réteg:

```
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 annyiban változtattunk a kódon, hogy a rejtett (hidden) neuronoknak 3 réteget hoztunk létre (az értékek: 6,4,6). Ahogy a képen is látszik, így helyes az eredmény.

4.6. Hiba-visszaterjesztéses perceptron

A program fel tudja dolgozni és meg tudja tanulni a bemenetet, ami 0-ból és 1-ből áll.

A program c++ nyelven kódolt.

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

```
#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 programban két új headert includeolunk, az "mlp.hpp"-t és a "png++/png.hpp"-t, ezek a megjelenítéshez kellenek és ebben van a perceptron elve is. A fő függvényünk (main) elején lefoglaljuk a tárhelyet a képnek és megadjuk a méreteit.

Aztán a perceptron létrehozása, a méret (size) a kép méret magasságnak és szélességének lesz a szorzata.

A majd létrehozunk egy double típusú size méretű képet, utána feltöltjük a megadott képpel, amelyeket a két for ciklus végel el.

A delete parancsokkal megakadályozzuk a memória szivárgást, azaz töröljük a perceptront és a képet.

4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve

Megoldás videó: <https://youtu.be/-GX8dzGqTdM>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Ebben a feladatban Stevenek érzékelnie kell az önmaga körül lévő kockákat. Az alapvető mozgás megmarad viszont kibővítjük egy observations használatával. Ezt egy 3x3x3-mas tömb alakban fogjuk használni amiben Steve környezetét vizsgáljuk. A programban a LineOfSight fogja meghatározni, hogy mi van Steve előtt, majd kiíratjuk az információkat a képernyőre.

A kódot két nyelven is futtathatjuk, az első a C++ nyelv, amelyben a program a már leírt módon működik, a második a Python nyelv, amelyben Steve már ha a pipacsokat érzékeli ki is üti azokat.

A program kód Python nyelven elérhető a repómban a következő linken: [mit_lat_Steve](#)

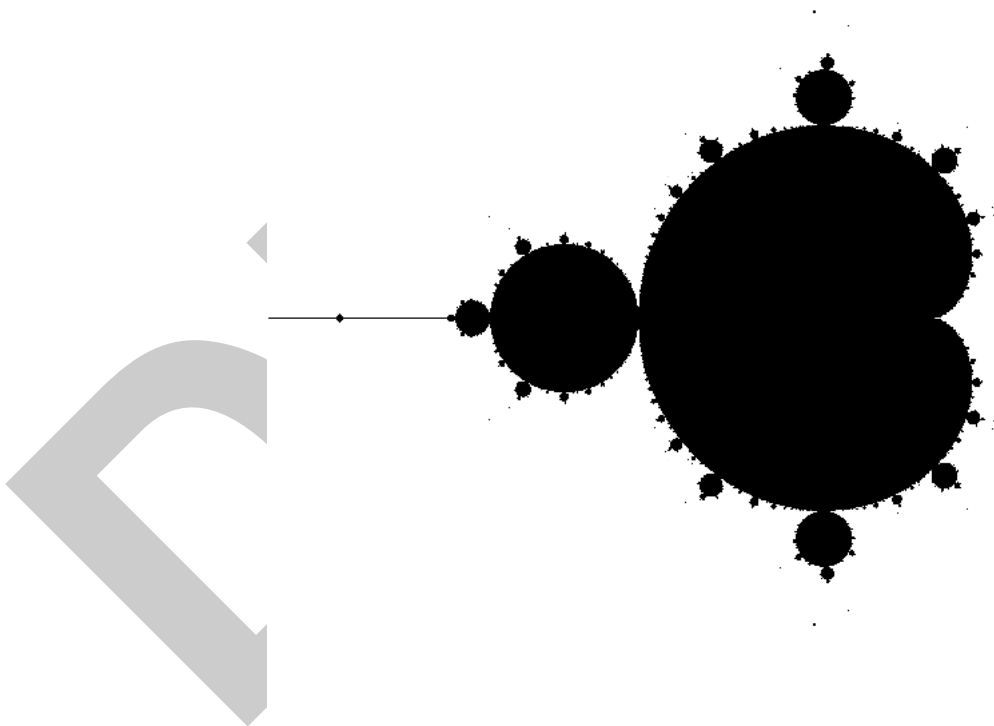
5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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



5.1. ábra. A Mandelbrot halmaz a komplex síkon

A Mandelbrot halmazt 1980-ban találta meg Benoit Mandelbrot a komplex számsíkon. Komplex számok azok a számok, amelyek körében válaszolni lehet az olyan egyébként értelmezhetetlen kérdésekre, hogy melyik az a két szám, amelyet összeszorozva -9 -et kapunk, mert ez a szám például a $3i$ komplex szám.

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1} = z_n^2 + c$, ($0 \leq n$) képlet alapján úgy, hogy a c az éppen vizsgált rácspon. A z_0 az origó. Alkalmazva a képletet a

- $z_0 = 0$
- $z_1 = 0^2 + c = c$
- $z_2 = c^2 + c$
- $z_3 = (c^2 + c)^2 + c$
- $z_4 = ((c^2 + c)^2 + c)^2 + c$
- ... s így tovább.

Azaz kiindulunk az origóból (z_0) és elugrunk a rács első pontjába a $z_1 = c$ -be, aztán a c -től függően a további z -kbe. Ha ez az utazás kivezet a 2 sugarú körből, akkor azt mondjuk, hogy az a vizsgált rácspon nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végtelen sok z -t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácsponthoz. Ha eközben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácspon a halmaz része. (Színes meg úgy lesz a kép, hogy változatosan színezzük, például minél későbbi z -nél lép ki a körből, annál sötétebbre).

Program 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;

    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;
}
```

Elsősorban a png++ headerre van szükségünk ahhoz hogy png-t tudjunk kezelni. Le kell töltenünk az internetről egy fájlt ami tartalmazza a headert. Mert a gépünk nem biztosítja ezt a headert. Ezután még telepíteni kell a libpng könyvtárat az alábbi módon: "sudo apt-get install libpng++-dev".

Továbbá definiálunk globális (állandó) változókat is, ezek a kép magassága és szélessége (x és y koordináta tengelyen).

Aztán létrehozunk egy `GeneratePNG()` nevű függvényt amely egy képet generál a következő módon. Létrehoz egy üres pngt ami 500x500 pixel (M és N) és itt használtuk először a png headert, majd a for cikluson belül 0-tól kezdve i és j változók (x és y tengely) segítségével "pixelről pixelre" meghatározzuk az rgb színkóddal (a png header használata újra) a színes pixeleket, végül az `image.write()` függvényel küldjük a kimenetre a képet.

Létrehoztunk egy struktúrát is amelyben két double típusú változót deklaráltunk. A komplex számok létrehozásához szükséges.

A `main()` fő függvényünkben létrehozunk egy egész típusú 500x500 (NxM) elemű tömböt, azánt két int típusú változót deklaráltunk a lépkedéshez a for ciklusba, továbbá két double típusút a komplex számokat (a pixelek meghatározásához).

Lefoglalunk a memóriában helyet a C, Z és Zuj változóknak, majd deklarálunk egy int típusú iteracio nevű változót amely az RGB színkód meghatározásához lesz szükséges, és a for ciklusban elvégezzük a számításokat amiket a tömbbe tesszük és meghívjuk a `GeneratePNG()` függvényt amely legenerálja a képet a számítások alapján.

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

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

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

Megoldás forrása:

A **Mandelbrot halmaz** pontban vázolt ismert algoritmust valósítja meg a repó [bhaxor/attention_raising/Mandelbrot/3.1.2.cpp](https://github.com/bhaxor/attention_raising_Mandelbrot/3.1.2.cpp) nevű állománya.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ↵
-0.01947381057309366392260585598705802112818 ↵
-0.0194738105725413418456426484226540196687 ↵
0.7985057569338268601555341774655971676111 ↵
0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ↵
0.4127655418209589255340574709407519549131 ↵
0.4127655418245818053080142817634623497725 ↵
0.2135387051768746491386963270997512154281 ↵
0.2135387051804975289126531379224616102874
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -l --line-numbers=1 --left-footer=" ↵
BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
```

```
//  
//  
// Copyright (C) 2019  
// Norbert Bátfai, batfai.norbert@inf.unideb.hu  
//  
// This program is free software: you can redistribute it and/or modify  
// it under the terms of the GNU General Public License as published by  
// the Free Software Foundation, either version 3 of the License, or  
// (at your option) any later version.  
//  
// This program is distributed in the hope that it will be useful,  
// but WITHOUT ANY WARRANTY; without even the implied warranty of  
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
// GNU General Public License for more details.  
//  
// You should have received a copy of the GNU General Public License  
// along with this program. If not, see <https://www.gnu.org/licenses/>.  
  
#include <iostream>  
#include "png++/png.hpp"  
#include <complex>  
  
int  
main ( int argc, char *argv[] )  
{  
  
    int szelesseg = 1920;  
    int magassag = 1080;  
    int iteraciosHatar = 255;  
    double a = -1.9;  
    double b = 0.7;  
    double c = -1.3;  
    double d = 1.3;  
  
    if ( argc == 9 )  
    {  
        szelesseg = atoi ( argv[2] );  
        magassag = atoi ( argv[3] );  
        iteraciosHatar = atoi ( argv[4] );  
        a = atof ( argv[5] );  
        b = atof ( argv[6] );  
        c = atof ( argv[7] );  
        d = atof ( argv[8] );  
    }  
    else  
    {  
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↵"  
            << std::endl;  
        return -1;  
    }  
}
```

```
    }

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

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

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

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

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

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

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

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

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

                ++iteracio;
            }

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

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

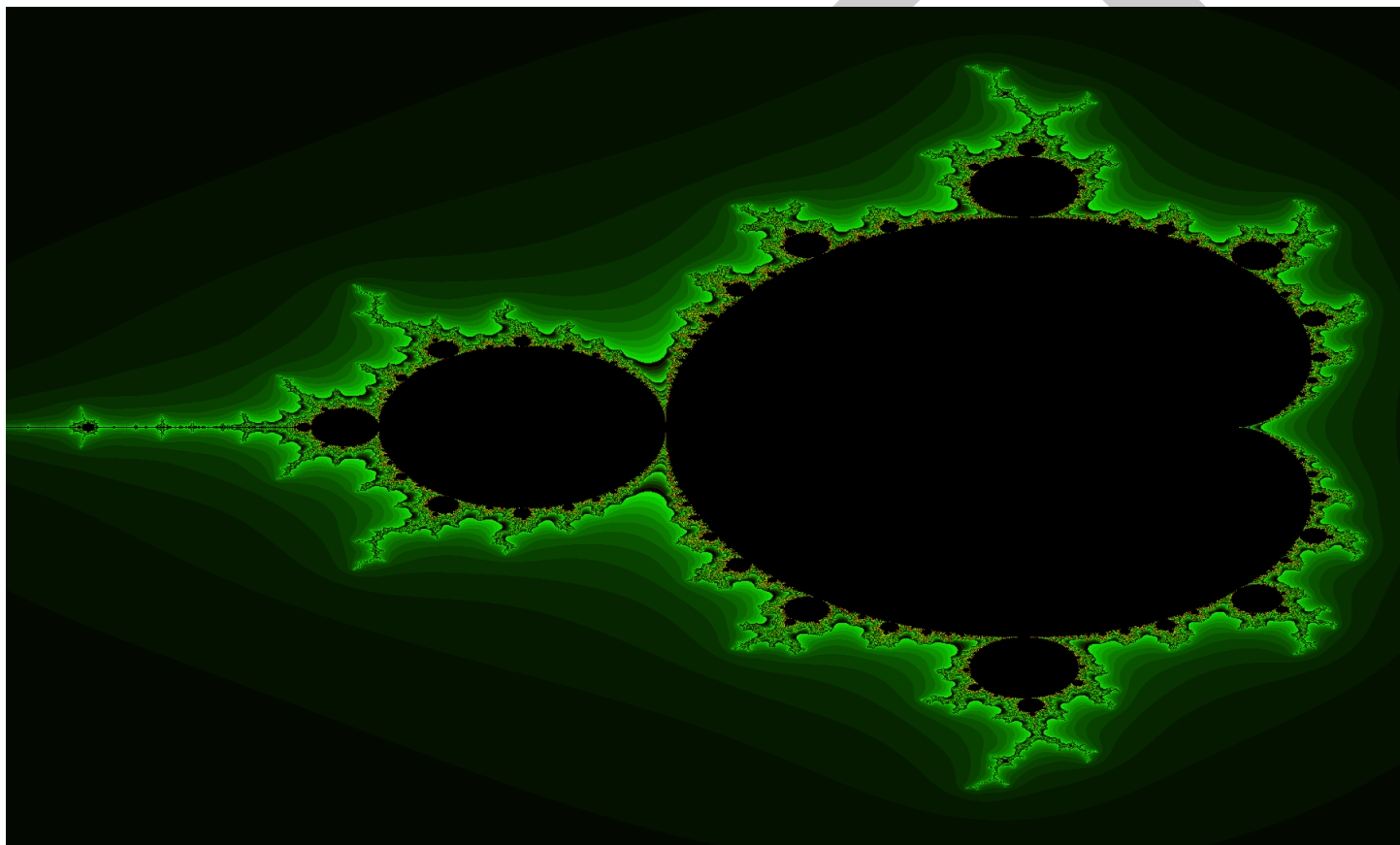
    kep.write ( argv[1] );
    std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

Magyarázat: A különbség az előző feladathoz képest az, hogy itt nem kell strukturával létrehozni komplex számokat, mert az új headerben a complex header már alapból tartalmaz komplex számokat.

A fő függvényben deklarálunk 2 változót, ha argumentumként jól adjuk meg ezeket, akkor ezeket átadja a változóknak, hanem jól adjuk meg, akkor kiírjuk, hogy kell helyesen használni. Aztán megadjuk a szélességet és a magasságot, ami ebbe az esetben FullHD(szélesség = 1920, magasság = 1080) és az iterációs határt.

Továbbá deklarálunk változókat amik a kép elkészítéséhez szükségesek. Ezek után lefoglaljuk a helyet a képnek. A dx, dy-hez hozzá rendeljük a megfelelő változókat. A forciklusban végig megyünk minden elemen (pixelen) és megadjuk a c változó értékét ekkor használjuk a complex headert. A while ciklusban végezzük a számításokat, utánna rgb kóddal a pixeleket kiszínezzük.

A futtatáshoz szükségünk lesz a -lpng kapcsolóra.



5.3. Biomorfok

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

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

A különbség a **Mandelbrot halmaz** és a Julia halmazok (biomorf) között az, hogy a komplex iterációban az előbbiben a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
```

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

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

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

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

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

++iteracio;
}
```

Ezzel szemben a Julia halmazos csipetben a cc nem változik, hanem minden vizsgált z rácspontra ugyanaz.

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
// k megy az oszlopokon
for ( int k = 0; k < szelesseg; ++k )
{
double reZ = a + k * dx;
double imZ = d - j * 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;
}
}
}
```

A biomorf programhoz 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 hozzuk létre. Megint végig megy a program minden ponton és ahol kell használjuk az RGB kódos színezést. A legvégén pedig kiküldjük a

képet a kimenetre.

a program kód a következő:

```
#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;

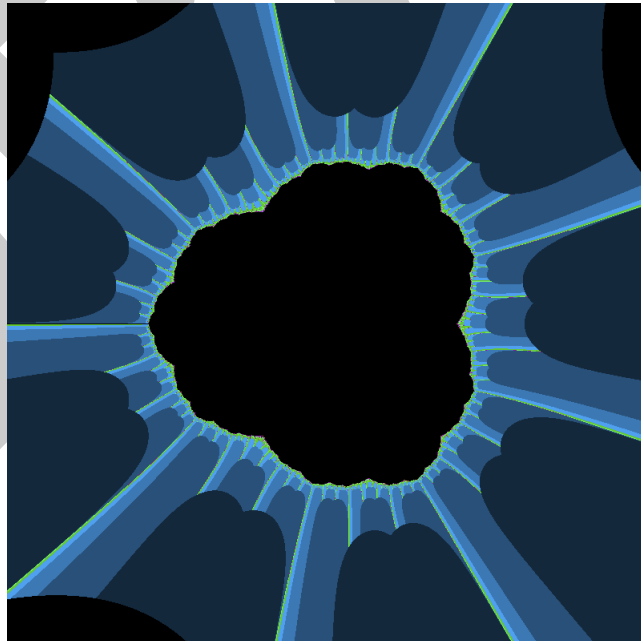
for (int i=0; i < iteraciosHatar; ++i)
{
    z_n = std::pow(z_n, 3) + cc;
    if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
    {
        iteracio = i; break;
    }
}

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

A CUDA az Nvidia videokártyáknak egy párhuzamos számításokat segítő technológia. Ezen technika segítségével fogjuk felgyorsítani a kép létrehozását. Szükségünk lesz egy Nvidida videokártyára ami rendelkezik CUDA-val. Továbbá telepítenünk kell. A kód kiterjesztése ".cu"

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

A kód és a magyarázat a következő:

```
#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, ujureZ, 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;
}

```

Includeoljuk a két állandót, 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.

```

/*
__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);
}

```

A következő függvény előtt "`__global__`" jelzés van, ezzel azt jelezzük, hogy a Cuda fogja végezni a számítást. A "`threadIdx`" jelzi az aktuális szálat és a "`blockIdx`" pedig, hogy melyik blokkban folyik a számítás. A kép értékeit a `j` és a `k` változókban tároljuk el. Ezt a két értéket fogja kapni az előző függvény.

```

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);
}

```

A következő függvény a `cudamandel()`, ez egy 600x600-as tömböt kap. Deklarálunk egy mutatót és a `Mal-loc` segítségével lefoglaljuk a megfelelő tárhelyet és a mutató ide fog mutatni. Itt hozzuk létre a megfelelő blokkokat és a végén a tárhelyet felszabadítjuk.

```
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" << "\n";
std::endl;
}
```

A fő függvényünkben 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 fordítás: "nvcc mandelpngc_60x60_100.cu -lpng16 -O3 -o mandelpngc".

Fordítás után futtatjuk: "./mandelpngc c.png"-vel.

Ha egymás mellé tesszük a Cudas és a nem Cudas képalkotást, láthatjuk, hogy a kép elkészítési ideje a Cudasnál sokkal gyorsabb.

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: https://progpater.blog.hu/2011/03/26/kepes_egypercesek

A program programozási nyelve a c++, további feladatban más programozási nyelven is magyarázzuk ezt a programot.

A jelenlegi programhoz több forrásra van szükségünk - például "frakablak.h" header - amelyeket egy mappába kell letölteni, majd telepítenünk kell a "libqt4-dev"-et a következő parancsal:

```
"sudo apt-get install libqt4-dev"
```

A qmake -project parancsal létrehozunk egy .pro fájlt, ebbe meg kell adnunk a QT+=Widgets parancsot a megfelelő helyre. Ez létrehoz egy fájlok.o fájlt és egy makefilet, ezek után make parancsal létrehozuk a nagyítót.

A frakszal.cpp-ben készül el az ábránk amit majd nagyítani fogunk. Az rgb pixel színezést azonban már a frakablak végzi.

A program kód a következő:

```
#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();
}
```

5.6. Mandelbrot nagyító és utazó Java nyelven

Tutorált: Talinger Mark Imre

Ez a feladat ugyan az mint az előző az 5.5-ös, azaz belenagyítunk a mandelbrot halmazba, a különbség viszont az, hogy itt a programozási nyelv a Java.

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

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, majd a kijelölt területen újra számoljuk a halmazt. Aztán feldolgozza a létrejött kép szélességét és magasságát.

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

A pillanatfelvétel() függvénnyel egy pillanatfelvételt készítünk. A függvényen belül elnevezzük a tartomány szerint és egy png formátumú képet készítünk a pillanat felvételből. A nagyítás során láthatunk egy segítő négyzetet, ezt a paint() függvényel hozzuk létre.

```
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);  
    }  
}
```

5.7. Vörös Pipacs Pokol/fel a láváig és vissza

Megoldás videó: <https://youtu.be/I6n8acZoyoo>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Ebben a feladatban már Steve érzékeli a veszélyt és el is menekül előle. Ami itt fontos az a megfelelő időben a veszély (láva) érzékelése. Tehát Steve felszalad egészen addig amíg a 3x3x3-mas önmaga körüli területen érzékeli a lávát. Ahol ugyanis érzékeli gyorsan megfordul és el kezd oldalazva menni lefelé kockánként. Egy **turn 1** utasítással tesszük ezt lehetővé.

A programkód megtalálható a repóban a következő linken: [lava_erzekelese](#)

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Az Objektum Orientált Programozás (OOP) alapja az osztályozás (class), vagyis minden egyes osztály egy-egy objektum.

Akkor nevezünk egy nyelvet objektum orientáltnak, ha egymással kommunikáló és műveleteket végző objektumokból áll egy program. Az OOP egyszerűsít a programon, növeli a hatékonyságot és átláthatóbb. A C++ és a Java egyaránt OOP nyelvek.

AZ első program az OOP bevezetéshez a polargen. Fontos lépés a feladatban, hogy egy számítási lépés két normális eloszlású számot állít elő, és minden második meghíváskor fel fogja használni az előzőleg tároltban elhelyezett számot, tehát nem fog mindig számolni.

Először a C++ nyelven fogunk végignézni a programot. A program 3 részből fog állni.

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

Bátfai Norbert forrását felhasználva.

Az első program C++ nyelven lesz látható, amely képes objektumokkal is dolgozni ezért OOP nyelv.

```
#ifndef POLARGEN__H
#define POLARGEN__H

#include <cstdlib>
#include <cmath>
#include <ctime>

class PolarGen
{
```

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

private:
bool nincsTarolt;
double tarolt;

};

#endif
```

Az első kódrészlet a header fájl. Itt definiáljuk a polargen headert és includeoljuk a szükséges headereket. A cmath header a matematikai számításokat tartalmazza, a ctime header a nekünk szükséges srand() időmérő függvényt tartalmazza.

Létrehozuk a PolarGen osztályt (class), amelynek a publikus részében egy konstruktort majd egy destruktort hozunk létre. A private részben deklarálunk két változót, egy logikai (bool) nincsTarolt ami a tárolásról ad majd információt, és egy double tarolt változót, ezek azért kerültek a privát részbe, hogy az értékükön ne legyen módunk változtatni. Viszátérve a publikus részhez, a konstruktorban kezdőértéket adunk a logikai változónak, ami jelen esetben igaz, vagyis hogy üres a változó. Az srand() függvény csak az időt méri, felhasználva a számítógép idő bitjét. A destruktort a programunk végén fog lefutni, amely a memóriaszivárgást akadályozza meg.

```
#include "polargen.h"

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 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);

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

```
        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

        return r * v1;
    }
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}
```

A második kódrészlet a számításokat végzi, itt valósítjuk meg a polártranszformációt. Includeoljuk a már megbeszélt `polargen.h` headert.

A lényeg, hogy az `if` fogja vizsgálni a `nincsTarolt` logikai értékét, azaz hogy van-e számunk a változóban, ha van akkor elvégzi a matematikai számításokat (ezek most nem fontosak). Ha nincs benne szám akkor új számot fogunk készíteni.

```
#include <iostream>
#include "polargen.h"

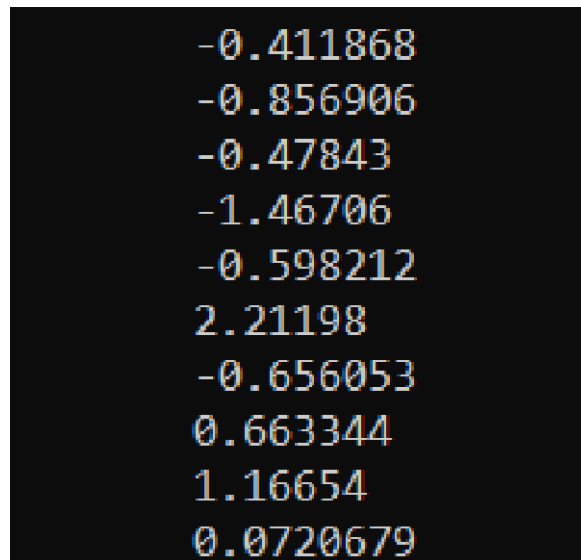
using namespace std;

int
main (int argc, char **argv)
{
    PolarGen pg;

    cout<<endl;
    for (int i = 0; i < 10; ++i)
        cout <<"\t"<< pg.kovetkezo () << endl;

    return 0;
}
```

Végül a harmadik kódrészlet maga a fő függvényünk amelyben kidomborodik a program, itt futtatja le a program a `for` ciklusban a számításokat végző függvényünket. Jelen esetben 10x fog lefutni.



```
-0.411868
-0.856906
-0.47843
-1.46706
-0.598212
2.21198
-0.656053
0.663344
1.16654
0.0720679
```

6.1. ábra. Polargenteszt.cpp futtatása

A következő program ugyan az mint az előző csak Java nyelven, ez a programozási nyelv amit már az előző fejezetekben tárgyaltunk, egy OOP nyelv amely csak objektumokkal dolgozik, a C++-al szemben.

```
public class PolarGen{

    boolean nincsTarolt = true;
    double tarolt;

    public PolarGen() {

        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;watch?v=9_ylSciSjBw&feature=youtu.be
    }
    else {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
}

public static void main(String[] args) {

    PolarGen g = new PolarGen();

    for(int i=0; i<10; ++i)
        System.out.println(g.kovetkezo());

}

}
```

Különösen nincs mit magyarázni, mert a program ugyan úgy működik mint az előző. A különbség annyi, hogy ebben a nyelvben csak objektumokat/függvényeket tudunk létrehozni, így már a program csipetben látható is, hogy a program kód egyben van és nincs részekre bontva.

6.2. LZW

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

A program egy bináris fát fog felépíteni bemeneti adatokból. Az LZW binfa minden csomópontjának (elágazás) két gyermeke (további ága) lehet. A csomópontok gyermekei vagy 0-ás vagy 1-es lehet. Az 1-es a jobb oldali, a 0-ás a bal oldali.

A binfa lényege hogy a gyökérből (kitüntetett elem) elérhetünk minden elemet.

Megoldás forrása Bátfai Norbert Tanárúrtól: https://progfather.blog.hu/2011/03/05/labormeres_otthon_avagy_hog_p5MQomtcQIdfTeZvPinhgRxu-CCsxGOx453MSrGk

```
BINF_PTR gyoker = uj_elem ();
gyoker->ertek = '/';
gyoker->bal_nulla = gyoker->jobb_egy = NULL;
BINFA_PTR fa = gyoker;
long max=0;
while (read (0, (void *) &b, sizeof(unsigned char)))
{
    for(i=0;i<8; ++i)
```

```
{
    egy_e= b& 0x80;
    if ((egy_e >>7)==0)
        c='1';
    else
        c='0';
}
if (c == '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;
    }
}
}
```

A fa építésében két esetünk van. A 0-ás és az 1-es beépítése a megfelelő helyre.

Ha 0-ás a bemenet akkor a gyökértől kezdve megnézzük, hogy van egy 0-ás (bal oldali) gyermeke, ha van akkor rálépünk, ha viszont nincs akkor létre kell hozni, majd miután létrehoztuk, vissza lépünk a gyökérre és olvassuk tovább a bemenetet.

Az 1-es bemenet esetén hasonlóan járunk el mint a 0-ásnál. A gyökértől kezdve megnézzük, hogy van egy 1-es (jobb oldali) gyermeke, ha van akkor rálépünk, ha viszont nincs akkor létre kell hozni, majd miután létrehoztuk, vissza lépünk a gyökérre és olvassuk tovább a bemenetet.

Végül az LZWBinfa ezen algoritmus alapján fog felépülni, és beépülni minden egyes csomópont (1-es vagy 0-ás) a fába.

Ebben a programrészletben a fát inorder bejárással dolgozzuk fel, ami azt jelenti hogy elsőre a bal oldallal, aztán a gyökérrel és végül a jobb oldallal foglalkozunk.

6.3. Fabejárás

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

Az előző feladatban létrehozott fát több féleképpen be lehet járni: inorder, preorder és postorder.

Lent e három fabejárás kódját lehet látni.

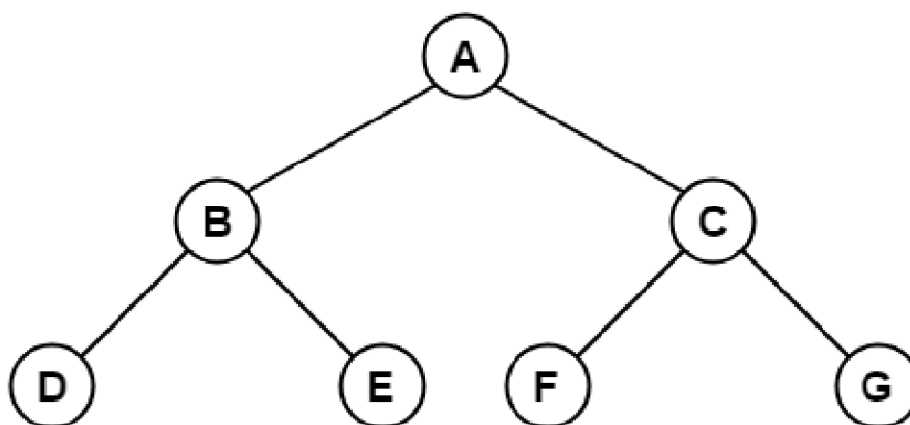
Az első az inorder:

```
void kiir (Csomopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyenesGyermeke (), os); //egyenes gyermeke ↵
        feldolgozása

        for (int i = 0; i < melyseg; ++i)
            os << "----";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << ↵
        std::endl; //csomópont feldolgozása

        kiir (elem->nullasGyermeke (), os); //nullas gyermeke ↵
        feldolgozása
        --melyseg;
    }
}
```

Ez már az előző feladatban tárgyalva volt. Tehát az inorder bejárással a fát úgy dolgozzuk fel, hogy egy adott csomópontnak mindig a bal gyereket dolgozzuk fel először, azután az aktuális csomópontot, azután pedig a jobb gyereket. A kitüntetett elem feldolgozása a bejárás közepén kerül sorra.



Inorder Traversal : D , B , E , A , F , C , G

6.2. ábra. Inorder fa bejárás

Kép forrása: <https://medium.com/@andrewmf/iterative-in-order-tree-traversal-using-dynamic-programming-508f189eb494>

Az fenti programrészletben a void típusú kiir() függvény inorder módon fogja az elemeket kiíratni.

A második a preorder:

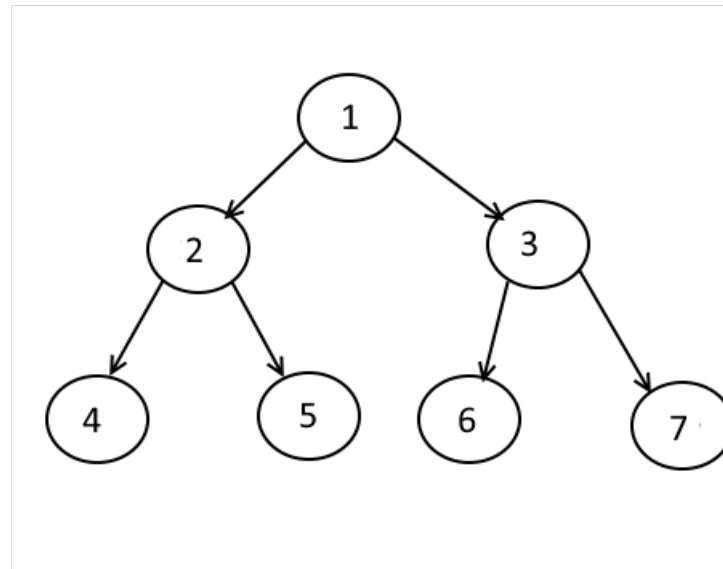
```
void kiir (Csomopont * elem, std::ostream & os)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a ←
    // rekurzió leállítása
    if (elem != NULL)
    {
        ++melyseg;

        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << ←
        std::endl;

        kiir (elem->egyresGyermekek (), os); //egyes gyermekek ←
        //feldolgozása

        kiir (elem->nullasGyermekek (), os); //nullás gyermekek ←
        //feldolgozása
        --melyseg;
    }
}
```


A preorder bejárással először a csomópontot, majd a bal, aztán a jobb gyermeket dolgozzuk fel. Ez azt jelenti, hogy a fát a gyökérből indulva, haladva a fa bal oldalán végig feldolgozzuk az összes csomópont bal oldali gyermekét, majd a jobb oldalit, amíg vissza nem érünk a gyökérhez, és akkor a feldolgozás ugyanúgy folytatódik, tovább a fa jobb oldalát is bejárjuk. A kitüntetett elem lesz feldolgozva először.



Preorder Traversal: 1 2 4 5 3 6 7

6.3. ábra. Preorder fa bejárás

Kép forrása: <https://algorithms.tutorialhorizon.com/binary-tree-preorder-traversal-non-recursive-approach/>

Az fenti programrészletben a void típusú kiir() függvény preorder módon fogja az elemeket kiíratni.

A harmadik a postorder:

```
void kiir (Csomopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;

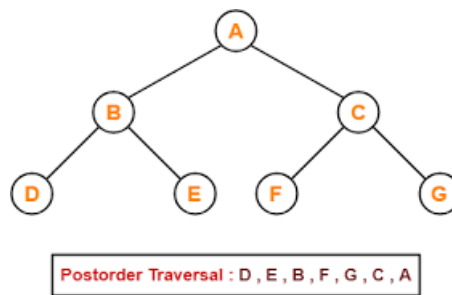
        kiir (elem->egysegGyermeke (), os);

        kiir (elem->nullasGyermeke (), os);

        for (int i = 0; i < melyseg; ++i)
            os << "----";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << " " <<
        std::endl;
    }
}
```

```
        --melyseg;  
    }  
}
```

A postorder bejárással először a bal oldali gyermeket, majd a jobb oldali gyermeket és végül az adott csomópontot dolgozzuk fel, elérve egészen a gyökérig. Itt a kitüntetett elem kerül utoljára feldolgozásra.



6.4. ábra. Postorder fa bejárás

Kép forrása: <https://www.wikitechy.com/technology/java-algorithm-iterative-postorder-traversal-set-2-using-one-stack/>

Az fenti programrészletben a void típusú kiir() függvény postorder módon fogja az elemeket kiíratni.

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!

A Csomópont osztályban létrehozzuk a '/' betűt tartalmazó objektumot, ami része a fának, vagyis tagként veszi a gyökeret is.

Megoldás forrása: <https://github.com/RubiMaistro/Bevprog/blob/master/Binfa/z3a7.cpp>

A forrás Báfai Norbert Tanárúrtól származik, a repó pedig saját.

```
#include <iostream>  
#include <cmath>  
#include <fstream>  
  
class LZWBinFa  
{  
public:  
  
    LZWBinFa () : fa (&gyoker)  
    {  
    }  
}
```

```
~LZWBinFa ()
{
    szabadit (gyoker.egyesGyermekek ());
    szabadit (gyoker.nullasGyermekek ());
}

void operator<< (char b)
{
    if (b == '0')
    {
        if (!fa->nullasGyermekek ())
        {
            Csomopont *uj = new Csomopont ('0');

            fa->ujNullasGyermekek (uj);

            fa = &gyoker;
        }
        else
        {
            fa = fa->nullasGyermekek ();
        }
    }
    else
    {
        if (!fa->egyesGyermekek ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyesGyermekek (uj);
            fa = &gyoker;
        }
        else
        {
            fa = fa->egyesGyermekek ();
        }
    }
}

void kiir (void)
{
    melyseg = 0;

    kiir (&gyoker, std::cout);
}

int getMelyseg (void);
```

```
double getAtlag (void);
double getSzas (void);

friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)
{
    bf.kiir (os);
    return os;
}
void kiir (std::ostream & os)
{
    melyseg = 0;
    kiir (&gyoker, os);
}
```

A program magyarázata:

Az LZWBinfa osztályban van privát és publikus rész.

Az osztály konstruktora és destruktora a publikus részében szerepel. A public részben szerepel az algoritmus, azaz a 0-ás és 1-es csomópontok beágyazása. Itt van a még kiir() függvény amely a kimenetre írat.

```
private:
class Csomopont
{
public:

    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy ←
        (0)
    {
    };
    ~Csomopont ()
    {
    };

    Csomopont *nullasGyermekek () const
    {
        return balNulla;
    }

    Csomopont *egysegGyermekek () const
    {
        return jobbEgy;
    }

    void ujNullasGyermekek (Csomopont * gy)
    {
        balNulla = gy;
    }
}
```

```

void ujEgyesGyermeK (Csomopont * gy)
{
    jobbEgy = gy;
}

char getBetu () const
{
    return betu;
}

```

A private részben van az LZWBinfa osztálynak a Csomopont osztálya. Ennek van konstruktora, destruktora (memóriaszivárgás elkerülés végett), nullás és egyes gyermeket lekérdező függvények (a vizsgáló függvények) és új nullás és új egyes gyermek létrehozásáért felelős függvények, valamint a lekérdező függvény, hogy mi található a csomópontban.

```

private:

    char betu;

    Csomopont *balNulla;
    Csomopont *jobbEgy;

    Csomopont (const Csomopont &);
    Csomopont & operator= (const Csomopont &);
};

```

A Csomopont osztály private részben van deklarálva a betű változó, ami megmondja, hogy milyen betű van a csomópontban, majd a jobb és bal gyermeket is deklaráljuk. Ez alatt található a Csomopont osztály másoló konstruktora.

```

Csomopont *fa;

int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;

LZWBinFa (const LZWBinFa &);
LZWBinFa & operator= (const LZWBinFa &);

void kiir (Csomopont * elem, std::ostream & os) //PREORDER
{
    if (elem != NULL)
    {
        ++melyseg;

        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << " ←"
        std::endl;

        kiir (elem->egyesGyermeK (), os);
    }
}

```

```
        kiir (elem->nullasGyermekek (), os);

        --melyseg;
    }
}
void szabadit (Csomopont * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->egyenesGyermekek ());
        szabadit (elem->nullasGyermekek ());

        delete elem;
    }
}
```

Tovább haladva, az LZWBinfa másoló konstruktora, a kiir függvényben kiírjuk a függvényt az os csatornára, itt tudjuk megadni, hogy milyen bejárással írja ki a fát, a fa bejárások feladatban volt róla szó. Ezt követi egy szabadit függvény, mely felszabadítja a szabad tárból az egyes gyermeket, a nullásat rekurzívan és végül az elemet is.

```
protected:

    Csomopont gyoker;
    int maxMelyseg;
    double atlag, szoras;

    void rmelyseg (Csomopont * elem);
    void ratlag (Csomopont * elem);
    void rszoras (Csomopont * elem);

};

int
LZWBinFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (&gyoker);
    return maxMelyseg - 1;
}

double
LZWBinFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double) atlagosszeg) / atlagdb;
```

```
        return atlag;
    }

    double
    LZWBinFa::getSzoras (void)
    {
        atlag = getAtlag ();
        szorasosszeg = 0.0;
        melyseg = atlagdb = 0;

        rszoras (&gyoker);

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

        return szoras;
    }

    void
    LZWBinFa::rmelyseg (Csomopont * elem)
    {
        if (elem != NULL)
        {
            ++melyseg;
            if (melyseg > maxMelyseg)
                maxMelyseg = melyseg;
            rmelyseg (elem->egyesGyermekek ());

            rmelyseg (elem->nullasGyermekek ());
            --melyseg;
        }
    }

    void
    LZWBinFa::ratlag (Csomopont * elem)
    {
        if (elem != NULL)
        {
            ++melyseg;
            ratlag (elem->egyesGyermekek ());
            ratlag (elem->nullasGyermekek ());
            --melyseg;
            if (elem->egyesGyermekek () == NULL && elem->nullasGyermekek () ←
                == NULL)
            {
                ++atlagdb;
                atlagosszeg += melyseg;
            }
        }
    }
}
```

```
    }
}

void
LZWBinFa::rszoras (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyenesGyermekek ());
        rszoras (elem->nullasGyermekek ());
        --melyseg;
        if (elem->egyenesGyermekek () == NULL && elem->nullasGyermekek () == NULL)
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag))
                ;
        }
    }
}
```

A protected rész arra szolgál, hogy a jövőbeni változtatások látszódnak majd a gyermek osztályban is. Itt jelenik meg a Csomopont gyoker is, valamint az rmelyseg, ratlag, rszoras függvények is. A protected rész tulajdonképpen öröklődésre, friend függvényekre használatos.

```
void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    if (argc != 4)
    {
        usage ();

        return -1;
    }

    char *inFile = *++argv;

    if ((*++argv) != 'o')
    {
        usage ();
        return -2;
    }
}
```



```
}

std::fstream beFile (inFile, std::ios_base::in);

if (!beFile)
{
    std::cout << inFile << " nem letezik..." << std::endl;
    usage ();
    return -3;
}

std::fstream kiFile (*++argv, std::ios_base::out);

unsigned char b;
LZWBinFa binFa;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
    if (b == 0x0a)
        break;

bool kommentben = false;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
{
    if (b == 0x3e)
    {
        kommentben = true;
        continue;
    }

    if (b == 0x0a)
    {
        kommentben = false;
        continue;
    }

    if (kommentben)
        continue;

    if (b == 0x4e)
        continue;

    for (int i = 0; i < 8; ++i)
    {
        if (b & 0x80)
            binFa << '1';
        else
            binFa << '0';
        b <<= 1;
    }
}
```

```
    }

    }

    kiFile << binFa;

    kiFile << "depth = " << binFa.getMelyseg () << std::endl;
    kiFile << "mean = " << binFa.getAtlag () << std::endl;
    kiFile << "var = " << binFa.getSzoras () << std::endl;

    kiFile.close ();
    beFile.close ();

    return 0;
}
```

A main függvényben láthatjuk az egyszerű hibakezeléseket: az argumentumszám, a kapcsoló. Majd olvasuk a bemeneti fájlból a karaktereket az fstream segítségével, ezeket átalakítjuk 0-vá vagy 1-sé, ezt logikai és használatával valósítjuk meg. A kiFile-ba irányítjuk a binfát, és kiírunk még róla néhány információt ezt követően (mélység, átlag, szórás). Majd bezárjuk az fstream fájlokat.

6.5. Mutató a gyökér

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

Az LZWBInfa programot vesszük alapul és a gyökér elemet átalakítjuk mutatóvá. A gyökér elem az osztály protected részében van.

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

A Csomopont osztályban a gyoker elemet át is írtuk mutatóvá. De ennyivel még nem vagyunk készen, mert ahogyan most egyszerűen átírtuk a kódban az osztály elemet mutatóra, minden helyen ahol használtuk az elemet, mutatóként kell használnunk.

```
//előtte:
fa=&gyoker;

//utánna:
fa=gyoker;

//előtte:
szabadit (gyoker.egyesGyermekek ());
szabadit (gyoker.nullasGyermekek ());

//utánna:
szabadit (gyoker->egyesGyermekek ());
```

```
szabadit (gyoker->>nullasGyermekek ());
```

A fenti kódcsipetben hoztam példákat. Először is nem a mutató memóriacímét akarjuk már átadni hanem az értékét, ezért töröltük a & jeleket. Továbbá nem elemként (.) hivatkozunk a gyokerre hanem mutatóként (->).

A gyökér mutatónak foglalni kell memóriát is, a következőképpen.

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

A konstruktorban megy végbe a memóriefoglalás, így a gyökér is egy csomópont lesz. A destruktorban a felszabadítást (a szabadit() függvényünkkel) és a törlést (a delete() függvénnyel) láthatjuk.

6.6. Mozgató szemantika

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

A feladat annyiból áll, hogy mozgató konstruktort adunk hozzá a programunkhoz, hogy fában az elemeket mozgatni/másolni tudjuk.

Ez a feladat a védés második feladata. A program már előre megírt, Bátfai Norbert Tanárúr felhasználásra bocsátása jóvoltából használom fel ehhez a feladathoz.

Saját megoldás videó: (videó link) (készül)

Megoldás forrása: https://gitlab.com/nbatfai/bhax/-/blob/master/distance_learning/ziv_lempel_welch/z3a18qa5_f

```
[
    #include <iostream>
    #include <random>
    #include <functional>
    #include <chrono>

    class Unirand {

    private:
        std::function <int ()> random;

    public:
        Unirand(long seed, int min, int max): random(
```

```
        std::bind(
            std::uniform_int_distribution<>(min, max),
            std::default_random_engine(seed)
        )
    ) {}

    int operator() () {return random();}

};

template <typename ValueType>
class BinRandTree {

protected:
    class Node {

    private:
        ValueType value;
        Node *left;
        Node *right;
        int count{0};

        // TODO rule of five
        Node(const Node &);
        Node & operator=(const Node &);
        Node(Node &&);
        Node & operator=(Node &&);

    public:
        Node(ValueType value, int count=0): value(value), count ←
            (count), left(nullptr), right(nullptr) {}
        ValueType getValue() const {return value;}
        Node * leftChild() const {return left;}
        Node * rightChild() const {return right;}
        void leftChild(Node * node){left = node;}
        void rightChild(Node * node){right = node;}
        int getCount() const {return count;}
        void incCount() {++count;}

    };

    Node *root;
    Node *treep;
    int depth{0};

private:
    // TODO rule of five

public:
    BinRandTree(Node *root = nullptr, Node *treep = nullptr): ←
        root(root), treep(treep) {
```

```
        std::cout << "BT ctor" << std::endl;
    }

    BinRandTree(const BinRandTree & old) {
        std::cout << "BT copy ctor" << std::endl;

        root = cp(old.root, old.treep);
    }

    Node * cp(Node *node, Node *treep)
    {
        Node * newNode = nullptr;

        if(node)
        {
            newNode = new Node(node->getValue(), 42 /*node-> ←
                getCount() */);

            newNode->leftChild(cp(node->leftChild(), treep));
            newNode->rightChild(cp(node->rightChild(), treep));

            if(node == treep)
                this->treep = newNode;
        }

        return newNode;
    }

    BinRandTree & operator=(const BinRandTree & old) {
        std::cout << "BT copy assign" << std::endl;

        BinRandTree tmp{old};
        std::swap(*this, tmp);
        return *this;
    }

    BinRandTree(BinRandTree && old) {
        std::cout << "BT move ctor" << std::endl;

        root = nullptr;
        *this = std::move(old);
    }

    BinRandTree & operator=(BinRandTree && old) {
        std::cout << "BT move assign" << std::endl;

        std::swap(old.root, root);
        std::swap(old.treep, treep);
    }
}
```

```

        return *this;
    }

    ~BinRandTree() {
        std::cout << "BT dtor" << std::endl;
        deltree(root);
    }
    BinRandTree & operator<<(ValueType value);
    void print() {print(root, std::cout);}
    void printr() {print(*root, std::cout);}
    void print(Node *node, std::ostream & os);
    void print(const Node &cnode, std::ostream & os);
    void deltree(Node *node);

    Unirand ur{std::chrono::system_clock::now(). ←
        time_since_epoch().count(), 0, 2};

    int whereToPut() {

        return ur();
    }

};

template <typename ValueType>
class BinSearchTree : public BinRandTree<ValueType> {

public:
    BinSearchTree() {}
    BinSearchTree & operator<<(ValueType value);

};

template <typename ValueType, ValueType vr, ValueType v0>
class ZLWTree : public BinRandTree<ValueType> {

public:
    ZLWTree(): BinRandTree<ValueType>(new typename BinRandTree< ←
        ValueType>::Node(vr)) {
        this->trep = this->root;
    }
    ZLWTree & operator<<(ValueType value);

};

```

```
template <typename ValueType>
BinRandTree<ValueType> & BinRandTree<ValueType>::operator<< ( ←
    ValueType value)
{
    int rnd = whereToPut();

    if(!treep) {
        root = treep = new Node(value);
    } else if (treep->getValue() == value) {
        treep->incCount();
    } else if (!rnd) {
        treep = root;
        *this << value;
    } else if (rnd == 1) {
        if(!treep->leftChild()) {
            treep->leftChild(new Node(value));
        } else {
            treep = treep->leftChild();
            *this << value;
        }
    } else if (rnd == 2) {
        if(!treep->rightChild()) {
            treep->rightChild(new Node(value));
        } else {
            treep = treep->rightChild();
            *this << value;
        }
    }

    return *this;
}
```

```
template <typename ValueType>
BinSearchTree<ValueType> & BinSearchTree<ValueType>::operator <<
    <<(ValueType value)
{
    if(!this->treep) {

        this->root = this->treep = new typename BinRandTree<
            ValueType>::Node(value);

    } else if (this->treep->getValue() == value) {

        this->treep->incCount();

    } else if (this->treep->getValue() > value) {

        if(!this->treep->leftChild()) {

            this->treep->leftChild(new typename BinRandTree<
                ValueType>::Node(value));

        } else {

            this->treep = this->treep->leftChild();
            *this << value;
        }

    } else if (this->treep->getValue() < value) {

        if(!this->treep->rightChild()) {

            this->treep->rightChild(new typename BinRandTree<
                ValueType>::Node(value));

        } else {

            this->treep = this->treep->rightChild();
            *this << value;
        }

    }

    this->treep = this->root;

    return *this;
}

template <typename ValueType, ValueType vr, ValueType v0>
ZLWTree<ValueType, vr, v0> & ZLWTree<ValueType, vr, v0>::
operator<<(ValueType value)
```



```
{

    if(value == v0) {

        if(!this->treep->leftChild()) {

            typename BinRandTree<ValueType>::Node * node = new ←↵
                typename BinRandTree<ValueType>::Node(value);
            this->treep->leftChild(node);
            this->treep = this->root;

        } else {

            this->treep = this->treep->leftChild();
        }

    } else {

        if(!this->treep->rightChild()) {

            typename BinRandTree<ValueType>::Node * node = new ←↵
                typename BinRandTree<ValueType>::Node(value);
            this->treep->rightChild(node);
            this->treep = this->root;

        } else {

            this->treep = this->treep->rightChild();
        }

    }

    return *this;
}

template <typename ValueType>
void BinRandTree<ValueType>::print(Node *node, std::ostream & ←↵
    os)
{
    if(node)
    {
        ++depth;
        print(node->leftChild(), os);

        for(int i{0}; i<depth; ++i)
            os << "---";
        os << node->getValue() << " " << depth << " " << node-> ←↵
            getCount() << std::endl;

        print(node->rightChild(), os);
    }
}
```

```
        --depth;
    }

}

template <typename ValueType>
void BinRandTree<ValueType>::print(const Node &node, std::ostream & os)
{
    ++depth;

    if(node.leftChild())
        print(*node.leftChild(), os);

    for(int i{0}; i<depth; ++i)
        os << "---";
    os << node.getValue() << " " << depth << " " << node.getCount() << std::endl;

    if(node.rightChild())
        print(*node.rightChild(), os);

    --depth;
}

template <typename ValueType>
void BinRandTree<ValueType>::deltree(Node *node)
{
    if(node)
    {
        deltree(node->leftChild());
        deltree(node->rightChild());

        delete node;
    }
}

BinRandTree<int> bar()
{
    BinRandTree<int> bt;
    BinRandTree<int> bt2;

    Unirand r(0, 0, 1);

    bt << 0 << 0 << 0;
    bt2 << 1 << 1 << 1;
    bt.print();
}
```

```
        std::cout << " --- " << std::endl;
        bt2.print();

        return r()?bt:bt2;
    }

    BinRandTree<int> foo()
    {
        return BinRandTree<int>();
    }

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

        std::cout << " *** " << std::endl;
        BinRandTree<int> bt2{bar()};
        std::cout << " *** " << std::endl;
        bt2.print();

    }
```

Az alábbi program fogja használni a mozgató szemantikát.

```
#include "vedes_Binfafa.h"

int main()
{
    ZLWTree<char, '/', '0'> tree1;
    ZLWTree<char, '/', '0'> tree2;

    tree1 = tree2;

    return 0;
}
```

Ha lefuttattuk a programot akkor láthatjuk a kimeneten, hogy milyen részek futottak le a programban, mert az egyszerűség kedvéért a feladatban tárgyalt kódrészletek lefutására bizonyítékként kiírtattuk, hogy az adott rész lefutott.

Az első két sor a kimeneten a BT ctor, ez annyit jelent hogy a Binfafa konstruktora lefutott. Tehát a tree1 és tree2 fákat létrehozta a program.

```
ZLWTree<char, '/', '0'> tree1;
ZLWTree<char, '/', '0'> tree2;
```

Aztán a BT copy assign sor azt jelenti, hogy amikor a tree1-et egyenlővé tesszük a tree2-vel,

```
tree1 = tree2;
```

akkor lefut a másoló értékadás és létrehoz egy átmeneti fát amibe lementi a tree2-t majd később abból rakja a tree1-be. A feladat arra ösztönösít hogy a mozgatót az értékadásra alapozzuk, vagyis meghívjuk a függvényben a másoló konstruktort.

```
BinRandTree & operator=(const BinRandTree & old) {  
    std::cout << "BT copy assign" << std::endl;  
  
    BinRandTree tmp{old};  
    std::swap(*this, tmp);  
    return *this;  
}
```

Majd a BT copy ctor sor, a másoló konstruktor lefutását mutatja.

```
BinRandTree(const BinRandTree & old) {  
    std::cout << "BT copy ctor" << std::endl;  
  
    root = cp(old.root, old.treep);  
}
```

Miután lemásolta, haladunk tovább és a mozgató konstruktor is meghívódik, és a mozgató konstruktor a mozgató értékadásra van alapozva, látszik lent hogy két objektum van egyenlővé téve.

```
BinRandTree(BinRandTree && old) {  
    std::cout << "BT move ctor" << std::endl;  
  
    root = nullptr;  
    *this = std::move(old);  
}
```

A BT move assign, azaz a mozgató értékadás meghívódik, mivel van két értékadás és még egy az átmeneti, így fut le háromszor.

```
BinRandTree & operator=(BinRandTree && old) {  
    std::cout << "BT move assign" << std::endl;  
  
    std::swap(old.root, root);  
    std::swap(old.treep, treep);  
  
    return *this;  
}
```

És végül a BT dtor sorok, azt mutatják hogy a destruktort is lefutott, vagyis töröltünk a memóriából mindent, így megakadályozva a memóriafolyást vagy memóriaszivárgást.

```
~BinRandTree() {  
    std::cout << "BT dtor" << std::endl;  
    deltree(root);  
}
```

6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Ebben a feladatban kiszélesítjük egy kicsit Steve látóterét, azaz már nem 3x3x3-mas hanem 5x5x5-ös tömbbe helyezzük Stevet amelyben mindent érzékelni fog. A programkódban megírt 3x3x3-mas helyére 5x5x5-öst írunk, majd az xml fájlban a szélső értékeket állítsuk át a következőképpen:

- minimumok: $x = -2$, $y = -2$, $z = -2$,
- maximumok: $x = 2$, $y = 2$, $z = 2$.

A programkód megtalálható a repóban a következő linken: [erzekelni_5x5x5_ben](#)

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

Tutorált: Talinger Mark Imre

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

Egy QT programban fogjuk szimulálni a hangyák mozgását.

A hangyák egymás közötti kommunikáció révén tájékozódnak, ezt feromonokkal érik el amit maguk után hagynak. A hangyák a feromon csíkok erősségének megfelelően választanak útvonalat maguknak. A lényege, hogy ez által megjelölik az útvonalukat és ezt jelezve a többi hangya felé. Ahol erősebb a feromon az azt jelenti, hogy a hangyák nagyon kedvelik azt a helyet ezért egyre többen járnak arra.

Ebben a szimulációban a hangyák közötti kommunikációt fogjuk reprodukálni.

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

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

A kód Bátfai Norbert tanárúrtól származik.

```
// BHAX Myrmecologist
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or ↵
// or modify
// it under the terms of the GNU General Public License as ↵
// published by
// the Free Software Foundation, either version 3 of the ↵
// License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be ↵
// useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty ↵
// of
```

```
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See ↵
// the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public ↵
// License
// along with this program. If not, see <https://www.gnu.org/ ↵
// licenses/>.
//
// https://bhaxor.blog.hu/2018/09/26/hangyaszimulaciok
// https://bhaxor.blog.hu/2018/10/10/myrmecologist
//
```

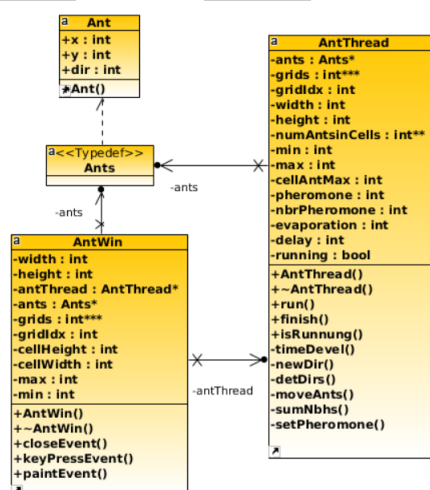
Először is szükségünk van a következő kódrészekre is:

```
ant.h
antwin.h
antthread.h

antwin.cpp
antthread.cpp
main.cpp
```

Ezek a következő linken elérhetők:

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



7.1. ábra. Hangya szimuláció

A kód magyarázata pedig a következő:

1. antthread.h:

```
#ifndef ANTTHREAD_H
```

```
#define ANTTHREAD_H

#include <QThread>
#include "ant.h"

class AntThread : public QThread
{
    Q_OBJECT

public:
    AntThread(Ants * ants, int ***grids, int width, int height,
              int delay, int numAnts, int pheromone, int ←
              nbrPheromone,
              int evaporation, int min, int max, int cellAntMax);

    ~AntThread();

    void run();
    void finish()
    {
        running = false;
    }

    void pause()
    {
        paused = !paused;
    }

    bool isRunnung()
    {
        return running;
    }

private:
    bool running {true};
    bool paused {false};
    Ants* ants;
    int** numAntsinCells;
    int min, max;
    int cellAntMax;
    int pheromone;
    int evaporation;
    int nbrPheromone;
    int ***grids;
    int width;
    int height;
    int gridIdx;
    int delay;

    void timeDevel();
}
```



```

    int newDir(int sor, int oszlop, int vsor, int voszlop);
    void detDirs(int irany, int& ifrom, int& ito, int& jfrom, ←
                int& jto );
    int moveAnts(int **grid, int row, int col, int& retrow, int ←
                & retcol, int);
    double sumNbhs(int **grid, int row, int col, int);
    void setPheromone(int **grid, int row, int col);

signals:
    void step ( const int &);

};

#endif

```

Ebben a headerben létrehozunk egy osztályt amiben a public részben a QT-s program ablakot fogjuk beállítani, méretezni és szabályozni. A private részben pedig az ablak adatai fogjuk tárolni (feromon, minimum, maximum, szélesség, magasság, fut vagy sem stb.). Aztán alatta a hangya vezérlő, útvonal meghatározó, feromon szint beállító függvények is itt lesznek, valamint a step konstans is.

2. ant.h header:

```

#ifndef ANT_H
#define ANT_H

class Ant
{

public:
    int x;
    int y;
    int dir;

    Ant(int x, int y) : x(x), y(y) {

        dir = grand() % 8;

    }

};

typedef std::vector<Ant> Ants;

#endif

```

Az ant.h headerben létrehozunk egy osztályt amely a hangyák jellem osztálya, itt definiáljuk a hangyák tulajdonságait, a koordinátáit az ablakon belüli elhelyezkedésüket (x,y), valamint az útvonalukat a konstuktorkkal, ami randomizált a grand() függvénnyel.

3. antwin.h:

```
#ifndef ANTWIN_H
#define ANTWIN_H

#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCloseEvent>
#include "antthread.h"
#include "ant.h"

class AntWin : public QMainWindow
{
    Q_OBJECT

public:
    AntWin(int width = 100, int height = 75,
          int delay = 120, int numAnts = 100,
          int pheromone = 10, int nbhPheromon = 3,
          int evaporation = 2, int cellDef = 1,
          int min = 2, int max = 50,
          int cellAntMax = 4, QWidget *parent = 0);

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {

        antThread->finish();
        antThread->wait();
        event->accept();
    }

    void keyPressEvent ( QKeyEvent *event )
    {

        if ( event->key() == Qt::Key_P ) {
            antThread->pause();
        } else if ( event->key() == Qt::Key_Q
                    || event->key() == Qt::Key_Escape ) {
            close();
        }

    }

    virtual ~AntWin();
    void paintEvent(QPaintEvent*);

private:

    int ***grids;
```

```
int **grid;
int gridIdx;
int cellWidth;
int cellHeight;
int width;
int height;
int max;
int min;
Ants* ants;

public slots :
    void step ( const int &);

};

#endif
```

Ez az utolsó header amelyben használni fogjuk a másik két headert. Ebben a headerben a QT-s ablakot bővítjük ki a kezelhetőséggel, vagyis itt találhatóak azok a függvények amelyek leállíthatják az ablakot (closeEvent()), szüneteltethetik (keyPressEvent). Valamint megtaláljuk a private részben ismét az ablak tulajdonságait (szélesség, magasság, min, max stb.). De az Ants vektor példányosítása is itt van.

4. antthread.cpp

```
#include "antthread.h"
#include <QDebug>
#include <cmath>
#include <QDateTime>

AntThread::AntThread ( Ants* ants, int*** grids,
                      int width, int height,
                      int delay, int numAnts,
                      int pheromone, int nbrPheromone,
                      int evaporation,
                      int min, int max, int cellAntMax)
{
    this->ants = ants;
    this->grids = grids;
    this->width = width;
    this->height = height;
    this->delay = delay;
    this->pheromone = pheromone;
    this->evaporation = evaporation;
    this->min = min;
    this->max = max;
    this->cellAntMax = cellAntMax;
    this->nbrPheromone = nbrPheromone;

    numAntsinCells = new int*[height];
    for ( int i=0; i<height; ++i ) {
```

```
        numAntsinCells[i] = new int [width];
    }

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j ) {
            numAntsinCells[i][j] = 0;
        }

    qsrand ( QDateTime::currentMSecsSinceEpoch() );

    Ant h {0, 0};
    for ( int i {0}; i<numAnts; ++i ) {

        h.y = height/2 + qrand() % 40-20;
        h.x = width/2 + qrand() % 40-20;

        ++numAntsinCells[h.y][h.x];

        ants->push_back ( h );
    }

    gridIdx = 0;
}

double AntThread::sumNbhs ( int **grid, int row, int col, int  ←
dir )
{
    double sum = 0.0;

    int ifrom, ito;
    int jfrom, jto;

    detDirs ( dir, ifrom, ito, jfrom, jto );

    for ( int i=ifrom; i<ito; ++i )
        for ( int j=jfrom; j<jto; ++j )

            if ( ! ( ( i==0 ) && ( j==0 ) ) ) {
                int o = col + j;
                if ( o < 0 ) {
                    o = width-1;
                } else if ( o >= width ) {
                    o = 0;
                }

                int s = row + i;
                if ( s < 0 ) {
                    s = height-1;
                } else if ( s >= height ) {
```

```
        s = 0;
    }

    sum += (grid[s][o]+1)*(grid[s][o]+1)*(grid[s][o] ←
        ]+1);

    }

    return sum;
}

int AntThread::newDir ( int sor, int oszlop, int vsor, int ←
    voszlop )
{

    if ( vsor == 0 && sor == height -1 ) {
        if ( voszlop < oszlop ) {
            return 5;
        } else if ( voszlop > oszlop ) {
            return 3;
        } else {
            return 4;
        }
    } else if ( vsor == height - 1 && sor == 0 ) {
        if ( voszlop < oszlop ) {
            return 7;
        } else if ( voszlop > oszlop ) {
            return 1;
        } else {
            return 0;
        }
    } else if ( voszlop == 0 && oszlop == width - 1 ) {
        if ( vsor < sor ) {
            return 1;
        } else if ( vsor > sor ) {
            return 3;
        } else {
            return 2;
        }
    } else if ( voszlop == width && oszlop == 0 ) {
        if ( vsor < sor ) {
            return 7;
        } else if ( vsor > sor ) {
            return 5;
        } else {
            return 6;
        }
    } else if ( vsor < sor && voszlop < oszlop ) {
        return 7;
    } else if ( vsor < sor && voszlop == oszlop ) {
```

```
        return 0;
    } else if ( vsor < sor && voszlop > oszlop ) {
        return 1;
    }

    else if ( vsor > sor && voszlop < oszlop ) {
        return 5;
    } else if ( vsor > sor && voszlop == oszlop ) {
        return 4;
    } else if ( vsor > sor && voszlop > oszlop ) {
        return 3;
    }

    else if ( vsor == sor && voszlop < oszlop ) {
        return 6;
    } else if ( vsor == sor && voszlop > oszlop ) {
        return 2;
    }

    else { //(vsor == sor && voszlop == oszlop)
        qDebug() << "ZAVAR AZ EROBEN az iranynal";

        return -1;
    }
}

void AntThread::detDirs ( int dir, int& ifrom, int& ito, int& ←
    jfrom, int& jto )
{

    switch ( dir ) {
    case 0:
        ifrom = -1;
        ito = 0;
        jfrom = -1;
        jto = 2;
        break;
    case 1:
        ifrom = -1;
        ito = 1;
        jfrom = 0;
        jto = 2;
        break;
    case 2:
        ifrom = -1;
        ito = 2;
        jfrom = 1;
        jto = 2;
        break;
    }
```

```
        case 3:
            ifrom = 0;
            ito = 2;
            jfrom = 0;
            jto = 2;
            break;
        case 4:
            ifrom = 1;
            ito = 2;
            jfrom = -1;
            jto = 2;
            break;
        case 5:
            ifrom = 0;
            ito = 2;
            jfrom = -1;
            jto = 1;
            break;
        case 6:
            ifrom = -1;
            ito = 2;
            jfrom = -1;
            jto = 0;
            break;
        case 7:
            ifrom = -1;
            ito = 1;
            jfrom = -1;
            jto = 1;
            break;
    }

}

int AntThread::moveAnts ( int **racs,
                          int sor, int oszlop,
                          int& vsor, int& voszlop, int dir )
{
    int y = sor;
    int x = oszlop;

    int ifrom, ito;
    int jfrom, jto;

    detDirs ( dir, ifrom, ito, jfrom, jto );

    double osszes = sumNbhs ( racs, sor, oszlop, dir );
    double random = ( double ) ( grand() %1000000 ) / ( double ←
```

```
        ) 1000000.0;
    double gvalseg = 0.0;

    for ( int i=ifrom; i<ito; ++i )
        for ( int j=jfrom; j<jto; ++j )
            if ( ! ( ( i==0 ) && ( j==0 ) ) )
            {
                int o = oszlop + j;
                if ( o < 0 ) {
                    o = width-1;
                } else if ( o >= width ) {
                    o = 0;
                }

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

                //double kedvezo = std::sqrt((double)(racs[s][o] ↔
                //+2)); //(racs[s][o]+2)*(racs[s][o]+2);
                //double kedvezo = (racs[s][o]+b)*(racs[s][o]+b ↔
                //);
                //double kedvezo = ( racs[s][o]+1 );
                double kedvezo = (racs[s][o]+1)*(racs[s][o]+1) ↔
                *(racs[s][o]+1);

                double valseg = kedvezo/osszes;
                gvalseg += valseg;

                if ( gvalseg >= random ) {

                    vsor = s;
                    voszlop = o;

                    return newDir ( sor, oszlop, vsor, voszlop ↔
                    );

                }

            }

    qDebug() << "ZAVAR AZ EROBEN a lepesnel";
    vsor = y;
    voszlop = x;

    return dir;
```



```
}

void AntThread::timeDevel()
{

    int **racsElotte = grids[gridIdx];
    int **racsUtana = grids[ ( gridIdx+1 ) %2];

    for ( int i=0; i<height; ++i )
        for ( int j=0; j<width; ++j )
        {
            racsUtana[i][j] = racsElotte[i][j];

            if ( racsUtana[i][j] - evaporation >= 0 ) {
                racsUtana[i][j] -= evaporation;
            } else {
                racsUtana[i][j] = 0;
            }

        }

    for ( Ant &h: *ants )
    {

        int sor {-1}, oszlop {-1};
        int ujirany = moveAnts( racsElotte, h.y, h.x, sor, ←
            oszlop, h.dir );

        setPheromone ( racsUtana, h.y, h.x );

        if ( numAntsinCells[sor][oszlop] <cellAntMax ) {

            --numAntsinCells[h.y][h.x];
            ++numAntsinCells[sor][oszlop];

            h.x = oszlop;
            h.y = sor;
            h.dir = ujirany;

        }

    }

    gridIdx = ( gridIdx+1 ) %2;
}

void AntThread::setPheromone ( int **racs,
                               int sor, int oszlop )
{
```

```
        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 = width-1;
                        } else if ( o >= width ) {
                            o = 0;
                        }
                    }
                    int s = sor + i;
                    {
                        if ( s < 0 ) {
                            s = height-1;
                        } else if ( s >= height ) {
                            s = 0;
                        }
                    }

                    if ( racs[s][o] + nbrPheromone <= max ) {
                        racs[s][o] += nbrPheromone;
                    } else {
                        racs[s][o] = max;
                    }

                }

        if ( racs[sor][oszlop] + pheromone <= max ) {
            racs[sor][oszlop] += pheromone;
        } else {
            racs[sor][oszlop] = max;
        }
    }

    void AntThread::run()
    {
        running = true;
        while ( running ) {

            QThread::msleep ( delay );

            if ( !paused ) {
                timeDevel();
            }
        }
    }
}
```

```
        emit step ( gridIdx );
    }

}

AntThread::~AntThread()
{
    for ( int i=0; i<height; ++i ) {
        delete [] numAntsinCells[i];
    }

    delete [] numAntsinCells;
}
```

Az antthread.cpp nevű fájlban az antthread.h header fileban létrehozott függvényeket, osztályt stb. használjuk, ezeket dolgozzuk ki.

A függvények amelyek nagyon fontosak:

sumNbhs() függvény: Statisztikát végző rész, és összegzés.

newDir() függvény: Létrehoz egy új pontot az ablakban. Azaz újabb hangya felbukkanása.

detDirs() függvény: Amelyben egy switch segítségével döntjük el a hangya tájékozódási pontjának helyzetét.

moveAnts() függvény: A hangyák hogyan fognak mozogni.

timeDevel() függvény: Az idő múlásának megfelelően milyen változások mennek végbe az ablakban.

setPheromone() függvény: A feromon szint nyilvántartása, változtatása.

run() függvény: A program futtatásért felelős.

destruktor: Ami törli a cellákból a hangyákat, amiket abba tárolunk, hogy meg tudjuk jeleníteni őket.

5. antwin.cpp:

```
#include "antwin.h"
#include <QDebug>

AntWin::AntWin ( int width, int height, int delay, int numAnts,
                int pheromone, int nbhPheromon, int evaporation ←
                , int cellDef,
                int min, int max, int cellAntMax, QWidget * ←
                parent ) : QMainWindow ( parent )
{
    setWindowTitle ( "Ant Simulation" );

    this->width = width;
    this->height = height;
    this->max = max;
    this->min = min;
```

```
cellWidth = 6;
cellHeight = 6;

setFixedSize ( QSize ( width*cellWidth, height*cellHeight ) ↔
               );

grids = new int**[2];
grids[0] = new int*[height];
for ( int i=0; i<height; ++i ) {
    grids[0][i] = new int [width];
}
grids[1] = new int*[height];
for ( int i=0; i<height; ++i ) {
    grids[1][i] = new int [width];
}

gridIdx = 0;
grid = grids[gridIdx];

for ( int i=0; i<height; ++i )
    for ( int j=0; j<width; ++j ) {
        grid[i][j] = cellDef;
    }

ants = new Ants();

antThread = new AntThread ( ants, grids, width, height, ↔
                             delay, numAnts, pheromone,
                             nbhPheromon, evaporation, min, ↔
                             max, cellAntMax);

connect ( antThread, SIGNAL ( step ( int) ),
          this, SLOT ( step ( int) ) );

antThread->start();

}

void AntWin::paintEvent ( QPaintEvent* )
{
    QPainter qpainter ( this );

    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i ) {
        for ( int j=0; j<width; ++j ) {

            double rel = 255.0/max;
```

```
        QPainter::fillRect ( j*cellWidth, i*cellHeight,
                             cellWidth, cellHeight,
                             QColor ( 255 - grid[i][j]*rel,
                                       255,
                                       255 - grid[i][j]*rel) ) ↔
        ;

    if ( grid[i][j] != min )
    {
        QPainter::setPen (
            QPen (
                QColor ( 255 - grid[i][j]*rel,
                        255 - grid[i][j]*rel, 255),
                1 )
        );

        QPainter::drawRect ( j*cellWidth, i*cellHeight,
                              cellWidth, cellHeight );
    }

    QPainter::setPen (
        QPen (
            QColor (0,0,0 ),
            1 )
    );

    QPainter::drawRect ( j*cellWidth, i*cellHeight,
                          cellWidth, cellHeight );

    }
}

for ( auto h: *ants) {
    QPainter::setPen ( QPen ( Qt::black, 1 ) );

    QPainter::drawRect ( h.x*cellWidth+1, h.y*cellHeight+1,
                          cellWidth-2, cellHeight-2 );

    }

    QPainter::end();
}

AntWin::~AntWin()
{
    delete antThread;

    for ( int i=0; i<height; ++i ) {
```

```
        delete[] grids[0][i];
        delete[] grids[1][i];
    }

    delete[] grids[0];
    delete[] grids[1];
    delete[] grids;

    delete ants;
}

void AntWin::step ( const int &gridIdx )
{

    this->gridIdx = gridIdx;
    update();
}
```

Ez a file a megjelenítésért lesz felelős. Itt láthatjuk azt a függvényt amely a hangyákat jeleníti meg (antWin), és a paintEvent() függvény amely a feromon csíkot jeleníti meg a cellákban feromon erősség szerint. A destruktork pedig elvégzi a piszkos munkát, vagyis törli a cellákat és a step() függvény frissíti a lépéseket.

6. main.cpp:

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

#include "antwin.h"

/*
 *
 * ./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a ↵
 * 255 -i 3 -s 3 -c 22
 *
 */

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 erteke 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() );
```

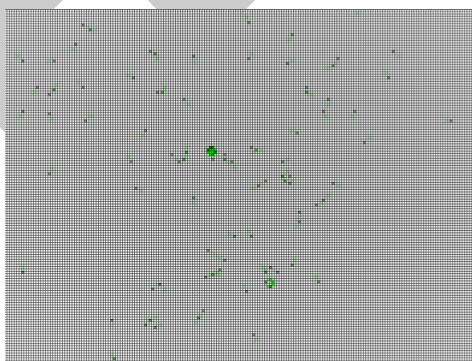
```
AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n. ←  
    toInt(), feromon.toInt(), szomszed.toInt(), parolgas. ←  
    toInt(),  
        alapertek.toInt(), mincella.toInt(), maxcella. ←  
        toInt(),  
        cellameret.toInt() );  
  
w.show();  
  
return a.exec();  
}
```

A main (fő) függvényben QT-s parancsokat használjuk és definiáljuk, valamint a korábban beszélt antwin.h headert is megadjuk. Itt tulajdonképpen a futtatáskor megjelenő beállítható paraméterezést láthatjuk.

```
QT += widgets  
  
TEMPLATE = app  
TARGET = myrmecologist  
INCLUDEPATH += .  
  
HEADERS += ant.h antwin.h antthread.h  
SOURCES += main.cpp antwin.cpp antthread.cpp
```

Ez a rész gyűjti össze a header és cpp fájlokat ami a szimuláció működéséhez szükségesek.

A programunkat fordítani a **qmake myrmecologist.pro** és a **make** parancsokkal tudjuk, futtatni pedig a **./myrmecologist** parannccsal. Ez egy default értékekkel történő futtatás, de meg lehet adni paramétereket is: **./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s 3 -c 22** parannccsal, ami a sima futtatásnál látványosabb.



7.2. ábra. Hangya szimuláció

7.2. Java életjáték

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

Az életjátékot azaz sejtautomatákat először Naumen János vetette fel, a gép önreprodukciójának matematikai modellalkotást tartalmazta. A legismertebb modell a John Horton Conway-féle életjáték.

A "játék" egy négyzetrácsos mezőn zajlik amin mozognak a sejtek. A sejtek "élete" szabályokhoz van kötve. Megvan adva hogy mi a feltétele egy sejt kialakulásának, életbenmaradásának vagy elpusztulásának. Conway erre 3 feltételt szabott meg:

- 1.szabály (túlélés): Egy sejt csak úgy éli túl, ha kettő vagy három szomszédja van.
- 2.szabály (elpusztulás): Egy sejt elpusztul, ha kettőnél kevesebb szomszédja van, ezt az elszigetelődés, vagy ha háromnál több szomszédja van, ez a túlnépesedés.
- 3.szabály (születés): Egy sejt születik, ha egy cellának a körzetében 3 sejt található.

Ezen a 3 szabály meghatározásával kapunk egy önműködő sejtautomatát. Beleszólásunk csak kezdetben van, utánna 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=IwAR0GcQ7v>

```
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 pillanatfelvételSzámoló = 0;

    public Sejtautomata(int szélesség, int magasság)
    {
        this.szélesség = szélesség;
        this.magasság = magasság;
        rácsek[0] = new boolean[magasság][szélesség];
        rácsek[1] = new boolean[magasság][szélesség];
        rácIndex = 0;
        rác = rácsek[rácIndex];

        for(int i=0; i<rác.length; ++i)
            for(int j=0; j<rác[0].length; ++j)
                rác[i][j] = HALOTT;

        siklóKilövő(rác, 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();
});

addMouseMotionListener(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 szomszédokSzáma(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 idő Fejlődés() {
        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(várakozás);
            } 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;
```

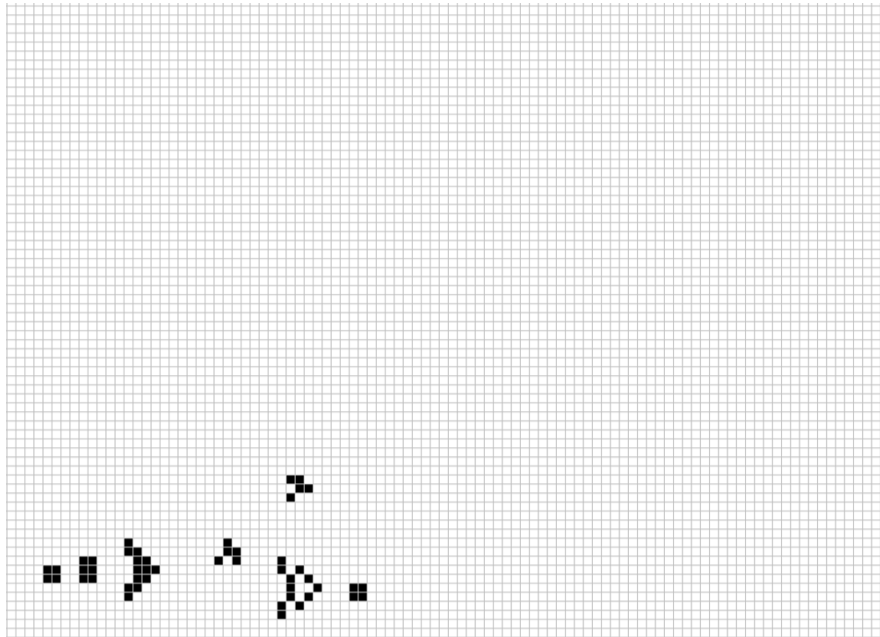
```
rács[y+ 11][x+ 25] = ÉLŐ;
}

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

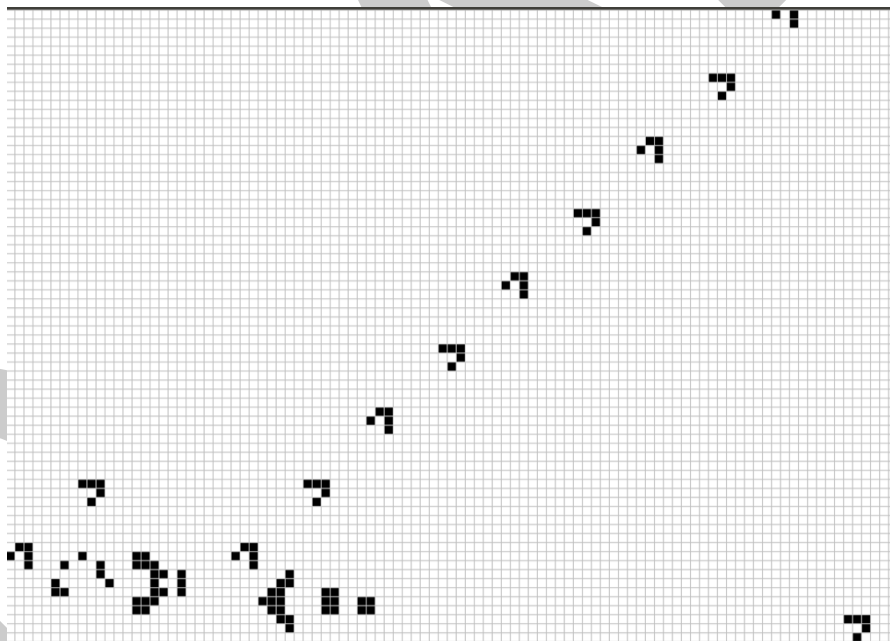
    try
    {
        javax.imageio.ImageIO.write(felvetel, "png", new java. ←
            io.File(sb.toString()));
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
}

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

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



7.3. ábra. Életjáték



7.4. ábra. Életjáték

A program elején megadjuk, hogy egy sejt lehet élő vagy halott. A feladatban 2 rácsfélét 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 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 beérkező parancsaink is, különböző feladatokkal ellátva pl a "g" betűvel, a ké tállapot közötti időt csökkentjük. Ugyan így vannak az egérrel történő információk feldolgozására szolgáló függvények, külön kattintásra és mozgásra. Külön tudunk készíteni pillanatfelvételt az aktuális állapotról az "s" gomb segítségével.

A programban a sejtter rajzolását a paint() függvénnyel végezzük. A szomszédokSzama() függvényben vizsgáljuk a szabályokat és a szerint történik a sejtek viselkedése.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Ez a feladat ugyan az mint az előző, a különbség itt a program nyelvben van, ez a kód QT C++-ban van írva.

A programhoz szükségesek az alábbi forráskódok.

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

```
#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 forrás fájljaink a Sejtablak.cpp, sejtszal.h, sejtszal.cpp és a sejtablak.h.

A sejtablak.h és sejtablak.cpp tartalmazza a függvényeket amivel majd a kirajzolás fog történni és ebben van a sikló lövés is, mint a java kódban megírtánál, külön minden egyes cellát megadunk amiben sejt van.

A sejszal.h és sejszal.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

A benchmark egy elemzés, tesztfeladat. Egy bizonyos tesztet végez el és azt az elért pontszám alapján összehasonlítja a tesztet elvégzők között és megtudhatjuk, hogy ki teljesített a legjobban és egymáshoz is tudjuk viszonyítani őket.

A BrainB egy kutatás céljával elkészült program, amely felméri az esport játékosok koncentrációs képességét. Nem csak esport játékosokra van kifejlesztve a program, hanem akik szeretik a videójátékokat, és egy rövid felmérést szeretnének kapni a saját koncentrációs képességükről.

Ebben a programban egy Samu Entropy nevű köröcskén kell lenyomva tartani a cursort 10 percig. A koncentráció méréséhez azt várja el a program, hogy a lenyomott cursorral kövessük az egyébként mozgolódó Samu Entropyt. Minél több ideig sikerül a köröcskében maradni annál gyorsabban nő a pontszámunk, és kezdenek megjelenni más Entropy köröcskék is, továbbá Samu is fürgébben fog mozogni. Amikor elvesztjük a Samut a sok Entropy között, akkor lelassul és a pontunk is csökkenni kezd. Az eredményünkről a 10 perc lejárla után kapunk információt.

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

A programhoz minden forrás megtalálható a fenti linken amely Bátfa Norbert tanárúrtól származik.

A következő programcsipet a BraintBTheard.h headerből való:

```
class Hero
{
    public:
        int x;
        int y;
        int color;
        int agility;
        int conds {0};
        std::string name;

        Hero ( int x=0, int y=0, int color=0, int agility=1, std::string name = "Samu Entropy" ) :
            x ( x ), y ( y ), color ( color ), agility ( agility ), name ( name )
        {}
        ~Hero() {}

        void move ( int maxx, int maxy, int env ) {

            int newx = x + ( ( ( double ) agility*1.0 ) * ( double ) ( std::rand() / ( RAND_MAX+1.0 ) ) - agility/2 ) );
            if ( newx-env > 0 && newx+env < maxx ) {
                x = newx;
            }
            int newy = y + ( ( ( double ) agility*1.0 ) * ( double ) ( std::rand() / ( RAND_MAX+1.0 ) ) - agility/2 );
            if ( newy-env > 0 && newy+env < maxy ) {
                y = newy;
            }

        }

};
```

Itt láthatunk egy Hero osztályt amiben tulajdonképpen létrehozzuk az Entropynkat. Továbbá létrehozzuk a tulajdonságait is: az elhelyezkedését, a nevét, a színét, a nagyságát stb.. Majd látunk még egy move() függvényt amely a mozgásért felelős.

A következő a BraintBTheard.cpp:

```
#include "BrainBThread.h"

BrainBThread::BrainBThread ( int w, int h )
{
    dispShift = heroRectSize+heroRectSize/2;
    this->w = w - 3 * heroRectSize;
    this->h = h - 3 * heroRectSize;

    std::srand ( std::time ( 0 ) );

    Hero me ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100,
             this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 9 );

    Hero other1 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100,
                 this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 5, "Norbi Entropy" );

    Hero other2 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100,
                 this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 3, "Greta Entropy" );

    Hero other4 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100,
                 this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 5, "Nandi Entropy" );

    Hero other5 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100,
                 this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 7, "Matyi Entropy" );

    heroes.push_back ( me );
    heroes.push_back ( other1 );
    heroes.push_back ( other2 );
    heroes.push_back ( other4 );
    heroes.push_back ( other5 );
}
```

```
    }

    BrainBThread::~BrainBThread() {} //Destruktor

    void BrainBThread::run()
    {
        while ( time < endTime ) {
            QThread::msleep ( delay );
            if ( !paused ) {
                ++time;
                devel();
            }
            draw();
        }
        emit endAndStats ( endTime );
    }

    void BrainBThread::pause()
    {
        paused = !paused;
        if ( paused )
        {
            ++nofPaused;
        }
    }

    void BrainBThread::set_paused ( bool p )
    {
        if ( !paused && p )
        {
            ++nofPaused;
        }
        paused = p;
    }
}
```

Ebben a header fájlban lévő függvények kidolgozását láthatjuk. Itt hozunk létre a Samu Entropy mellé még másik négy Entropyt, majd egy destruktort. Alatta egy run() függvényt ami a program futtatásáért felelős, továbbá egy pause() illetve egy set_pause() függvény amely a szüneteltetésért illetve a leállásért felelős.

A következő a BrainBWin.h:

```
#include <QKeyEvent>
#include <QMainWindow>
#include <QPixmap>
#include <QPainter>
#include <QFont>
#include <QFile>
#include <QString>
#include <QCloseEvent>
#include <QDate>
```

```
#include <QDir>
#include <QDateTime>
#include "BrainBThread.h"

enum playerstate {
    lost,
    found
};

class BrainBWin : public QMainWindow
{
    Q_OBJECT

    BrainBThread *brainBThread;
    QPixmap pixmap;
    Heroes *heroes;

    int mouse_x;
    int mouse_y;
    int yshift {50};
    int nofLost {0};
    int nofFound {0};

    int xs, ys;

    bool firstLost {false};
    bool start {false};
    playerstate state = lost;
    std::vector<int> lost2found;
    std::vector<int> found2lost;

    QString statDir;

public:
    static const QString appName;
    static const QString appVersion;
    BrainBWin ( int w = 256, int h = 256, QWidget *parent = 0 ) ↔
        ;

    void closeEvent ( QCloseEvent *e ) {

        if ( save ( brainBThread->getT() ) ) {
            brainBThread->finish();
            e->accept();
        } else {
            e->ignore();
        }

    }
}
```

```
virtual ~BrainBWin();  
void paintEvent ( QPaintEvent * );  
void keyPressEvent ( QKeyEvent *event );  
void mouseMoveEvent ( QMouseEvent *event );  
void mousePressEvent ( QMouseEvent *event );  
void mouseReleaseEvent ( QMouseEvent *event );
```

Mivel a programhoz ismét használunk QT-t itt fent látható az ablakkezelés. Amely jelen esetben is az ablak méretezéssel, paraméterezéssel foglalkozik, valamint az eventekkel.

A főfüggvény a main a következő:

```
#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," << endl;
```

```

        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();
}

```

Itt tulajdonképpen csak leírást kapunk a program céljáról és a fontosabb információkról. Majd a kódcsipet végén meghívjuk a futáshoz szükséges függvényeket.

Végül láthatjuk azt a programrészt is amely összegyűjti a header és cpp fájlokat amelyek a működéshez szükségesek.

```

QT += widgets core
CONFIG += c++11 c++14 c++17
QMAKE_CXXFLAGS += -fopenmp
LIBS += -fopenmp
LIBS += `pkg-config --libs opencv`

TEMPLATE = app
TARGET = BrainB
INCLUDEPATH += .

HEADERS += BrainBThread.h BrainBWin.h
SOURCES += BrainBThread.cpp BrainBWin.cpp main.cpp

```

A fordítása a programnak a **qmake** paranccsal, valamint a **make** paranccsal, a fordítása pedig a **./BrainB** paranccsal történik.

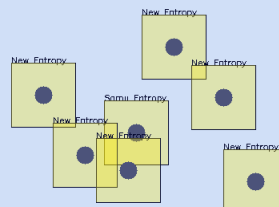
```

NEMESPOR BrainB Test 6.0.3
time      : 241

```

```
bps      : 11160
noc      : 7
nop      : 0
lost     :
17430 14380 3640 5250 0 0 0 12750
mean     : 6681
var      : 7139.94
found    : 0 2940 11710 10180 11340 16150 36520
mean     : 12691
var      : 11867.5
lost2found: 0
mean     : 0
var      : 0
found2lost: 12750
mean     : 12750
var      : 0
mean(lost2found) < mean(found2lost)
time     : 0:22
U R about 0.778198 Kilobytes
```

Egy tesztként lefuttatott próba eredménye látható fent és vizuálisan a program ablakban az Enrtopyk lent.



7.5. ábra. BrainB Benchmark

7.5. Vörös Pipacs Pokol/19 RF

Megoldás videó: <https://youtu.be/VP0kfvRYD1Y>

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Első passz.

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó: <https://youtu.be/j7f9SkJR3oc> linken elérhető, Bátfai Norbert megoldása.

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

A Python nyelv egy magas szintű programozási nyelv melyet 1989-ben Guido van Rossum holland származású programozó kezdett kifejleszteni, majd 1991-ben kiadta művét. A python nyelv dinamikus típusokat használ, a típusoknak két fajtája létezik, úgymond a mutálható és a mutálhatatlan, ebből érthetjük hogy tömören csak a megváltoztathatóságról van szó. Ez egy kicsit hasonlíthat már az előbbi feladatokban a C++ nyelvben -ami a python után alakult ki- a globális és nem globális OPP részekhez. Valmint a már számunka ismert OPP (objektumorientált) programozást is támogatja.

Az eredeti forráskód:

```
# Copyright 2015 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License" ↵
#   ");
# you may not use this file except in compliance with the ↵
#   License.
# You may obtain a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, ↵
#   software
# distributed under the License is distributed on an "AS IS" ↵
#   BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express ↵
#   or implied.
# See the License for the specific language governing ↵
#   permissions and
# limitations under the License.
```

```
# ↵
=====

"""A very simple MNIST classifier.

See extensive documentation at
http://tensorflow.org/tutorials/mnist/beginners/index.md
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

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

import tensorflow as tf

flags = tf.app.flags
FLAGS = flags.FLAGS
flags.DEFINE_string('data_dir', '/tmp/data/', 'Directory for ↵
                    storing data')

mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

sess = tf.InteractiveSession()

# 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.nn.softmax(tf.matmul(x, W) + b)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), ↵
                    reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize( ↵
                    cross_entropy)

# Train
tf.initialize_all_variables().run()
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    train_step.run({x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1) ↵
    )
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf. ↵
```

```
float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test. ←
labels}))
```

A fenti forráskód szabadon terjeszthető és felhasználható a TensorFlow Authors fejlesztőinek feltüntetésével. Így ebben a kódban kedvünkre dolgozhatunk, az alábbiakban egy átdolgozott megvalósítását láthatjuk a TensorFlow programnak.

Az átdolgozott progi és magyarázata:

A Minst kézzel írott számok adatbázisa, amely 6000 képet tartalmaz. Ez az alapja azoknak a programoknak, ami képről ismeri fel a tárgyakat. A Minst program a kézel írt számokról el fogja dönteni, hogy milyen szám, de a kézírás mindenkinek más, viszont a programnak mindig tudnia kell, hogy melyik számot kell felismernie, ez az igazán izgalmas dolog ebben a programban.

A programhoz a TensorFlow-t használjuk. A TensorFlow egy a Google által alkotott gépi tanulási rendszer, melyet sok helyen használnak, az egyik az a Google Mapsben található utcakép. A TensorFlow nyílt forráskódú, ezért bárki letöltheti, felhasználhatja, bővítheti és új ötleteit valósíthatja meg egy már megírt kód segítségével.

A forráskódot részekre bontva magyarázom:

```
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
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

import matplotlib.pyplot

FLAGS = None
```

Azokat a könyvtárakat, amelyek szükségesek a **from** kulcsszóval adjuk a programunkhoz, hogy dolgozni tudjunk velük. Majd importáljuk a "TensorFlow" könyvtárt az **import** kulcsszóval és elnevezzük tf-nek, hogy később könnyebben hivatkozhatunk a rövid névre. Majd importáljuk a "matplotlib.pyplot" könyvtárat, ami majd a kép kirajzolásához lesz szükséges.

```
def readimg():
    file = tf.read_file("sajat.png")
    img = tf.image.decode_png(file, 1)
    return img
```

Ezután következik `reading()` függvény, ami a képeinket olvassa be és dekódolja, majd lent a `main()` függvény, amiben a kiíratás felépítése van, hogy hogyan küldjük ki az eredményeket, a függvények definiálásához **def** kulcsszót használjuk.

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

    # The raw formulation of cross-entropy,
    #
    #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y))
    #
    #   ,
    #                                     reduction_indices=[1]))
    #
    # can be numerically unstable.
    #
    # So here we use tf.nn.softmax_cross_entropy_with_logits on the
    # raw
    # outputs of 'y', and then average across the batch.
    cross_entropy = tf.reduce_mean(tf.nn.
        softmax_cross_entropy_with_logits(labels = y_, logits = y))
    train_step = tf.train.GradientDescentOptimizer(0.5).minimize(
        cross_entropy)

    sess = tf.InteractiveSession()
    # Train
    tf.initialize_all_variables().run(session=sess)
    print("-- A halozat tanitasa")
    for i in range(1000):
        batch = mnist.train.next_batch(50)
        batch_xs, batch_ys = mnist.train.next_batch(100)
        correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_
            , 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.
            float32))
        train_accuracy = accuracy.eval(feed_dict={
            x: batch[0], y_: batch[1]})
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
        mylist.append(train_accuracy)
        ilist.append(i)
```

```

if i % 100 == 0:
    print(i/10, "%")

```

A sess azaz egy session segítségével fogjuk a tanítást végezni, hasonlóan mint a neurális hálózathoz. A session objektum lehetővé teszi, hogy egy kérés folytatódjon egy bizonyos paraméteren keresztül. Ez a Train program rész azaz a vonat rész, a session tulajdonságából kiindulva, hogy hosszú időn keresztül folytatódik a folyamat. A pontosság miatt a ciklust ezerszer fogjuk futtatni egy for ciklusban, hogy az eredmény tökéleteset megközelítse. Aztán kiíratjuk mennyire lett pontos az eredmény százalékban, a print() függvény segítségével.

```

# Test trained model
print("-- A halozat tesztelese")

print("-- A MNIST 42. tesztkepenek felismerese, mutatom a ←
      szamot, a tovabblepeshez 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 -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(" ←
      -----")
matplotlib.pyplot.plot(ilist,mylist, color='red', linestyle=' ←
    solid', linewidth = 1,marker='o',

```

```
markerfacecolor='blue', markersize=3)

matplotlib.pyplot.show()
```

Elérkeztünk a képzett modell tesztelés (Test trained modell) kódrészhez. A programcsipetben először megjelenik egy a mnist 42. tesztkép, a rajzoláshoz a **matplotlib.pyplot**-t használjuk. Majd megjelenik a saját képünk és a session tovább futtat. Végül megjelenik a saját képünk értékelése, hogy mennyire volt felismerhető a kézírásunk.

```
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()
```

Ha egy képet bezárunk akkor jön a következő kép és így tovább. Legvégül program a kapott eredményeket egy tömbben tárolja el.

A program viszonylag nagy pontossággal fogja felismerni a képeken lévő számokat, ezt a többszöri futtatásnak, jelen esetben már említett 1000 futtatásnak köszönhetünk.

8.2. Mély MNIST

Python

Passz.

8.3. Minecraft-MALMÖ

Információkat a Malmö projekttel kapcsolatban bevezetésképpen a Turing fejezetben még az első ilyen feladat előtt írtam tájékoztatást, hogy érthető legyen ezen projekt célja. Itt már egy picit komolyabb kifejtésre kerül sor, de a számos feladatnak a külön-külön magyarázatának az olvasásával mélyebb betekintést nyerhet az olvasó.

Megoldás videó: initial hack: <https://youtu.be/bAPSu3Rndi8>. Red Flower Hell: <https://github.com/nbatfai/-RedFlowerHell>.

A projektben számtalan kreatív lehetőséggel rendelkezünk ahogyan ezt a projekt vezetői is említik. A lehetőségünk az ágens irányításával kezdve a különböző blockok azonosításával és a környezet felismerésével a blockok mozgatásán át, a különböző szituációkban, az npc és mob közelség reakcióáig és még tovább egy kreatív programozó és fan számára kimeríthetetlen lehet.

Ezen RFH vagy Vörös Pipacs Pokol projektnek lényege, hogy egy aréna jellegű map-al rendelkezünk, mely egy tölcsérhez hasonló. A láva folyik az aréna peremétől egészen az aréna közepéig, pontosan 300

másodperc alatt ér le a közepére, és feladatunk, hogy a lehető legtöbb Pipacsot szedjünk össze ez idő alatt, melyek minden szinten el vannak helyezve, random koordinátákkal.

Van lehetőség előre és hátra menni a **move**, fordulni a **turn**, a nézés irányát változtatni a **look**, ütni a **attack** parancsok segítségével és még több másra is képesek vagyunk. A 2020-as tavaszi félév során a kezdetben még csak a csigavonalban haladó Steve intelligenciája a program megírása alapján már eljutott addig, hogy a piros virágokat érzékelje és küsse majd felszedje és tovább menjen, természetesen ezt mind időre a láva leérkezése előtt.

Az ágensprogramozás nem olyan egyszerű mint ahogyan azt gondolnánk, de a végeredmény mosolyt tud csalni az arcunkra. Ellentétben azzal, hogy számtalanszor lehet nemvárt hibákba futni, amelyeket nem a legkönnyebb kijavítani és tökéletessé csiszolni. Viszont a cél az ágens megismerése és minden paramétert megfelelően beállítani annak érdekében, hogy a megfelelő eredményt érjük el.

A [Red Flower Hell](#) repóban különböző érdekes programkódok elérhetők.

A mesterséges intelligencia fejlesztői is azon dolgoznak, hogy ezt ne csak egy ilyen MALMÖ projektben ahol különböző blockok felismerésével érünk el eredményt hanem már pixelek alapján is értelmezhetővé váljon a vizuális térben való elhelyzkedés és különböző feladatok elvégzése.

8.4. Vörös Pipacs Pokol/javíts a 19 RF-en

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Második passz.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Tutorált: Talinger Mark Imre

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

A Lisp egy olyan programozási nyelv amely felkapott lett a MI (mesterséges intelligencia) kutató, fejlesztők körében, habár a nyelv nem erre a célra jött létre. A nyelv szintaktikája egyedi, listákat lácol egybe és ezeket dolgozza fel. A listákat és némely esetben segítségül hívva ezt a program nyelvet matematikai órákon és feladatok számításában használják.

Ez a nyelv egy kifejezés orientált nyelv, mely segítségével ebben a feladatban a faktoriálisliságot szemléltetjük.

Iteratíván:

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

A **defun** szóval adjuk meg a függvény és a benne lévő változó nevét. Majd a **do** konstrukcióval megadjuk azokat az utasításokat amelyeket el kell végezni. A következő sorban megnöveli az *i* értékét, érdekes lehet hogy a műveleti jelet listában a két érték elé tesszük. A következő sorban szorozzuk az *i*-t a proddal, majd a következőben addig növelgetjük az *i*-t amíg egyenlő nem lesz a proddal.

Rekurzívan:

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

Itt is a **defun** szóval megadjuk a függvény és benne lévő változó nevét. Majd egy **if** feltétellel megnézzük hogy az *n* egyenlő-e eggyel, ha igen akkor szorozzuk össze az (*n*-1)-szeresével.

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

Harmadik passz.

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_hackese_a_scheme_programozasi_nyelv

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

Negyedik passz.

9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-edén

Megoldás forrása: <https://github.com/nbatfai/RedFlowerHell>

Ötödik passz.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

A programozási nyelveknek három szintje van: gépi nyelv, assembly szintű nyelv, magas szintű nyelv. A magas szintű nyelven írt programot forrásszövegnek nevezzük. Egy magas szintű programozási nyelv: szemantikai és szintaktikai szabályok együttese. A szintaktikai szabályok: forrásszöveg összeállítására vonatkozó formai és nyelvtani szabályok összessége. A szemantikai szabályok: tartalmi, értelmezési és jelentésbeli szabályok.

A gépi nyelvet a processor ismeri, tehát a magas szintű forrásszöveget fordítóprogram segítségével vagy interpreteléssel kell a processorhoz juttatni.

Fordítóprogram: gépi kódú tárgyprogramot állít elő. Lépései:

lexikális elemzés

szintaktikai elemzés

szemantikai elemzés

kódgenerálás (csak szintaktikailag helyes forrásprogramból lehet előállítani tárgyprogramot.)

Az interpreteres technika esetén is megvan az első három lépés, de az interpreter nem készít tárgyprogramot, hanem utasításonként értelmezi a forrásprogramot és végrehajtja azt. Az egyes programnyelvek együttesen alkalmazzák mindkét technikát.

Hivatkozási nyelv: a programnyelv szabálya.

Implementációk: fordítóprogramok vagy interpreterek. A hordozhatóság problémája az implementációk inkompatibilitását jelenti, melyek adott platformon realizált fordítóprogramok vagy interpreterek. A problémára már 50 éve nincs teljesen korrekt megoldás.

Programozási nyelvek osztályai: Imperatív: algoritmus nyelvek és utasítások sorozata, az algoritmus módosítja a processzort, a változó jelenléte és közvetlen elérése (Alcsoportjai: Eljárásorientált, Objektumorientált nyelvek).

Deklaratív nyelvek: nem algoritmikus nyelvek, nincs lehetőség memóriaműveletekre, csak a problémát adja meg a programozó (Alcsoportjai: Funkcionális, Logikai nyelvek).

Más nyelvek: nincs egységes jellemzőjük, tagadják valamelyik imperatív jellemzőt.

A forrásszöveg legkisebb alkotórészei a karakterek. Alapvető a karakterkészlet, ezekből állíthatók össze a bonyolult nyelvi elemek. Eljárásorientált nyelvek esetén ezek: lexikális egységek, szintaktikai egységek és utasítások, programegységek, program.

A karakterek kategorizálása: betűk, számjegyek, egyéb karakterek. Minden programnyelvben betű az angol ABC 26 nagybetűje, és ezek közül a kis és nagy betűket is egyes nyelvek elfogadják (C), más nyelvek viszont nem (Pascal).

Még előforduló karakterek: `_`, `$`, `#`, `@`, `+`, `-`, `*`, `/`, `[`, `]`, `.`, `:`, `{`, `}`, `'`, `"`, `;`, `?`, `!`, `~`.

A lexikális egységek a program szövegének azon elemei, melyeket a fordító a lexikális elemzés során felismer és tokenizál.

Fajtái: Többkarakteres szimbólumok: `++`, `--`, `/*`, `*/` stb.

Szimbolikus nevek:

Azonosító (karakteresorozat ami betűvel kezdődik és betűvel vagy számjeggyel folytatódik) pl: `x`, `ab`, `hallgato_azonosito`, `SzemelyNev`

Nem azonosító pl: `x+y`, `123abc`.

Kulcsszavakb(alapszó) pl: `if`, `for`, `case`, `break`.

Cimke: speciális karakteresorozat, amely lehet előjel nélküli egész szám vagy azonosító. Általános, hogy utasítás előtt áll és `-`-al van elválasztva.

Cimke felépítése példa: azonosító (C), 4 számjegyből álló egész szám (Pascal)

Megjegyzés (Komment vagy magyarázat): A program szövegét olvasó embernek szól, nem a fordítónak. `//` vagy `/*` itt magyarázhatjuk a programrész működését `*/`

Literálok (Konstansok): Fix, explicit értékek a program szövegében. pl: egész literálok, valós literálok, karakter literálok, sztring literálok.

Forrásszöveg összeállításának általános szabályai:

A kötött formátumú nyelvek esetén egy sorban egy utasítás volt elhelyezhető, a szabad formátumú nyelvek esetén akárhányszor utasítás egy sorban, két pontosvessző között áll egy utasítás. A lexikális egységeket alapszóval vagy szóközzel kell elválasztani.

Adattípusok:

Az adattípus egy absztrakt programozási eszköz, amely mindig más. Konkrét programozási eszköz egy komponens. Az adattípusnak neve van, egy azonosító.

Egy adattípust meghatároz:

a tartománya: ahol felvehető értéként,

a műveletei: a tartomány elemein,

és reprezentációja: egyes típusok tartományába tartozó értékek tárban való megjelenése.

Egyszerű típusok: egész (fixpontos), valós (lebegőpontos), karakteres (karakteres ábrázolás), logikai (igaz vagy hamis), felsorolás, sorszámozott.

Összetett típusok: tömb (dimenzió száma, indexkészlet típusa és tartománya, elemeinek típusa) A C nem ismeri a többdimenziós tömböt (egydimenziós tömb egydimenziós tömb-elemekkel képzeletben). A mutató: egyszerű típus, tartományának elemei tárcímek, azaz egy adott tárbeli területre mutat. Speciális tartománybeli eleme a NULL.

A nevesített konstans három komponensből áll: név, típus és érték. Mindig deklarálni kell, akkor használjuk amikor egy érték sokszor előfordul és ezt elnevezzük egy nevesített konstansra, amire egyszerűen hivatkozhatunk. A C ben #define név a literál.

A változó négy komponense: név, attribútumok, cím, érték. Ezek minden változónál szerepelnek. A név egy azonosító.

Az attribútumok a futásközbeni viselkedést határozzák meg (ez lehet akár a típusa). Változó attribútumok esetén deklarációt alkalmazunk: explicit deklaráció: programozó végzi, teljes nevéhez kell az attribútumokat megadni; implicit deklaráció: betűhöz rendel attribútumokat, azonos kezdőbetűjű változók u.olyan attribútumúak lesznek; automatikus deklaráció: a fordítóprogram rendel attribútumokat a változókhoz.

A változó címe: ahol a tárnak azt a részét határozza meg ahol a változó értéke elhelyezkedik. A cím rendelhető: statikusan, dinamikusan, programozó által. De mindháromra kell olyan eszköz ami megszünteti a címkomponenst.

A változó értéke: értékadó utasítás által (változó = kifejezés), input (egy perifériáról), kezdőértékadás (explicit,implicit).

Az alapelemek a C nyelvben: aritmetikai típusok (egyszerű); integrális típusok; valós típusok; származtatott (összetett) pl tömb (egydimenziós), függvény, mutató, struktúra; void típus.

Kifejezések: Szintaktikai eszközök.

Két komponensük az érték és a típus.

Formális összetevői az operandusok (érték), az operátorok (műveleti jelek) és a kerek zárójelek(sorrend szabályozásra).

Vannak egyoperandusú (unáris), kétoperandusú (bináris) és háromoperandusú (ternáris) operátorok.

Három alakja van a kifejezéseknek:

prefix (operátor operandusok előtt): * 3 5,

infix (között): 3 * 5,

postfix (után): 3 5 *.

Műveletek végrehajtási sorrendje:

- balról jobbra (standard)
- jobbról balra (fordított)
- balról jobbra (precedencia táblával, vagyis prioritás(zárójelekkel) segítségével)

Infix alakban balról-jobbra történő művelet végrehajtási szabály van. Infix alak esetén kell használni zárójeleket (ezek lesznek az elsődlegesek). Vannak logikai operátorok is(és,vagy...).

A kiérékelések típusai:

- teljes (pl.FORTRAN)
- rövidzár (pl.PL/I)
- rövidzár operátorok: and then, or else
- nem rövidzár op.-ok: and, or

A kifejezés típusa lehet:

tipusegyenértékű (kétooperandusú operátornak csak azonos típusú operandusai lehetnek) vagy tipuskényszerítő (különböző típusú operandusok is lehetnek).

A konstans kifejezés kiértékelését a fordító végzi. Operandusai lehetnek literálok és nevesített konstansok.

A C egy alapvetően kifejezésorientált nyelv. A mutató típusú összeadás és kivonás végezhető. A tömb típusú eszköz neve mutató típusú, tehát $a[i] = *(a+i)$.

Példák C beli operátorokra (precedencia táblázat alapján):

1. balról jobbra:

$()$ (függvényoperátor,precedencia felülírás),

$[]$ (tömboperátor),

$\&\&$ (és operátor, kétooperandusú)

$?:$ (háromoperandusú).

2. jobbról balra:

$=$ (értékadás)

$*=$ (szorzás és értékadás a bal oldalra),

$+=$, $\hat{=}$, stb.

Utasítások:

Az utasítások az eljárásorientált nyelvek egyes lépéseit adják meg, és ezáltal generálja a fordítóprogram a tárgyprogramot, melynek két csoportja van, deklarációs és végrehajtható utasítások. A deklarációs utasítások mögött nem áll tárgykód, ezek a fordítóprogramnak szólnak, tehát befolyásolják a tárgykódot, de ők nem kerülnek fordításra. A végrehajtható utasításokból generálódik a tárgykód a fordítóprogram által. Besorolhatjuk őket több alosztályba:

Értékadó utasítás: beállít vagy módosít egy változó értékén a program futása közben egy bizonyos időpontban.

Üres utasítás: gyértelmű programszerkezet alakítható ki velük.

Ugró utasítás: Korai nyelvekben használatos, mely egy feltétel teljesülésének következtében egy meghatározott részére ugrik a programnak, és egy adott címkével (felétel) ellátott utasítást fog végrehajtani.

Elágaztató utasítás: Két választási lehetőség van a program adott részén feltételek alapján (pl. if else), vagy több lehetőség közül (pl. switch case 1, case 2, ...stb.)

Ciklusszervező utasítás: A program adott pontján egy tevékenységet akárhányszor elvégezhetünk.

A ciklusszervező utasítások, a C nyelvben:

Kezdőfeltételes: $\text{While}(\text{feltétel})\{\text{utasítás}\}$,

Végfeltételes: $\text{Do}\{\text{utasítás}\} \text{ while}(\text{feltétel})$,

For-ciklus: $\text{For}(\text{kif1}; \text{kif2}; \text{kif3})\{\text{utasítás}\}$

Vezérlő utasítások a C-ben:

Continue; Újrakezdi a feltételvizsgálatot, ami pedig utána van az nem hajtódik végre,

Break; Megtöri vagy leállítja a ciklust, és kilép az utasításból,

Return (kifejezés); (Befejezteti a függvényt és visszaadja a vezérlést a hívónak).

Programok szerkezete:

Az eljárásorientált nyelvekben a program szövege programegységekre tagolható

Az alábbi programegységek léteznek:

Alprogramok: Akkor használjuk, ha több helyen is felhasználnunk egy programrészt, és ezt külön egy helyre leírjuk, amit majd később ismertetek, hogy hogyan használhatjuk fel többször is. Négy komponensből áll: név, paraméter lista, törzs, környezet.

A neve egy azonosító, a paraméter lista lehet üres is vagy azonosítók szerepelnek benne, amelyeknek szerepe lesz az alprogramban. A törzsben deklarációs, végrehajtandó utasítások vannak, itt van leírva, hogy mit csináljon az alprogramunk. Az alprogram környezete alatt a globális változók együttesét értjük.

Az alprogramok két kategóriába tartoznak: eljárás és függvény.

Az eljárás egy olyan alprogram amely több utasítást hajt végre, a hívás helyén az eredményét használjuk fel. A függvény egy olyan alprogram, mely egyetlen értéket határoz meg, és ezzel tér vissza. A függvény visszatérési érték a hívás helyére tér vissza.

Függvény hívás: függvénynév (paraméter lista)

Eljárás hívása: [alapszó] eljárásnév(paraméter lista).

A hívási lánc, vagy rekurzió az bizonyos programegységek egymásba ágyazott hívásán alapszik. Egy programegység bármikor meghívhat egy másik programegységet, és a vezérlés oda ugrik.

Amikor egy aktiv alprogramot hívunk meg, azt nevezzük rekurciónak.

Rekurzió lehet:

- közvetlen: egy alprogram önmagát hívja, azaz magára hivatkozik.
- közvetett: a hívási láncban korábban szereplő alprogramot hívunk meg.

Másodlagos belépési pontok: Vannak nyelvek, melyek megengedik, hogy egy alprogramot ne csak fejen keresztül lehessen meghívni, hanem a törzsben ki lehessen alakítani ún. másodlagos belépési pontokat, tehát ezzel is lehet hivatkozni az alprogramra.

A paraméterkiértékelés az a folyamat, amikor egymáshoz rendelődnek a formális és aktuális paraméterek egy alprogram hívásánál. Mindig a paraméter lista az elsődleges, az aktuális paraméterlistából akárhány lehet, attól függ hogy hányszor hívjuk az alprogramot. A paraméterszám lehet fix, de tetszőleges is.

A paraméter átadás egy kommunikációs forma az alprogramok és más programegységek között. Mindig van egy hívó, és egy hívott.

A nyelvek által ismert paraméterátadási módok:

- érték szerinti,
- cím szerinti,
- eredmény szerinti,
- érték-eredmény szerinti,
- név szerinti,
- szöveg szerinti.

A blokk egy programegység. Más programegység belsejében helyezkedik el kizárólag. Van kezdete, törzse és vége.

A hatáskör nevekhez kapcsolódik. Hatáskör alatt értjük a program szövegének egy olyan részét, ahol jelentése felhasználási módja és jellemzői azonosak. A hatáskör lehet lokális, vagyis egy programegységen belül van deklarálva, és lehet globális, amely mindenhol elérhető a program területén.

10.2. C programozás bevezetés

Rövid olvasónapló a [KERNIGHANRITCHIE] könyvről.

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

Alapismeretek

A könyv első fejezete a C nyelv alapismereteinek elsajátításáról szól, főképpen azok irányába ajánlott aki már tanultak programozni. A képernyőre való kiírás megvalósítását ismerjük meg a printf() függvény használatával.

Megismerjük a Fahrenheit-hőmérséklet Celsius-értékké alakításának megoldását is, valamint a változó típusokat. Az ismétlődő ciklusokat is bemutatja, azaz a while és a for.

Bevezetést kapunk a szimbolikus állandók fogalmába is, néhány alapvető függvényt (getchar(), putchar()) ismertet. Napi rendszerességgel használt eljárás: karakterek számlálása, szavak számlálása, sorok számlálása, valamint tömbök, függvények, argumentumok és érték szerinti hívások.

Típusok, operátorok, kifejezések

Ebben a fejezetben változónevek szabályairól, használati utasításait olvashatjuk, valamint tovább boncoljuk az adattípusokat és a hozzá kapcsolódó méreteket. A matematikai állandók is szóbajönnek, mint pl. az e vagy pi, de állandó létrehozást is kifejtik a fejezetben.

A deklarációra is kitér a típusok, operátorok, kifejezések, változók tekintetében. Megismerkedhetünk az aritmetikai operátorokkal, relációs és logikai operátorokkal. A típuskonverziók témaköre rendkívül hasznos és megtudhatjuk, hogy csak az értelmes konverziók történnek meg.

Az inkrementáló, dekrementáló operátorok is szóbajönnek, amik szintén alap szintű felfogást igényelnek a megértéshez, valamint a bitenkénti logikai operátorok. Értékadó operátorok, feltételes kifejezések, precedencia (kiértékelés sorrendje).

Vezérlési szerkezetek

Itt ismerjük meg az utasításokat és blokkokat, valamint az if-else utasítást, else-if utasítást, switch utasítást, while, for utasítást, do-while utasítást, break utasítást, continue utasítást, goto utasítást, címkéket.

Fontos ezeket megjegyezni, mivel sok példával van szemlélteve, mert tulajdonképpen alapkövek a C nyelvben, és más nyelvekben is előkerülnek, lehet hogy más formában, de előkerülnek. Egyszerűek, meg lehet őket jegyezni, sok gyakorlással és alkalmazással meg a kisujjunkba kerülhetnek.

Függvények és programstruktúra

Itt a függvényeket és a program felépítési szabályait vesszük ki. A függvények előnyeiről, hasznosságáról találhatunk példákat, valamint konkrét programrészleteket, helyes és működő programstruktúrákról. Külső változók, regiszter változók, érvényességi tartomány szabályai sem marad ki, ezeket is nagyon érthetően elemzi a könyv, és a kiváló példákkal elülteti az ember agyában. Statikus változók és a blokkstruktúra is

megjelenik, valamint további nyelvekkel kerül összehasonlításba a C nyelv. Az inicializálás, rekurzió, C előfeldolgozó, állomány beiktatás és makrohelyettesítés témakörei is taglalva vannak.

Utasítások

Az utasítások egymást követően sorban hajtódnak végre. Több fajtáját is megkülönböztetjük az utasításoknak:

Kifejezés utasítás: értékadások, függvényhívások.

Összetett utasítás/blokk: ahol elvileg egy utasítás helyezhető el, ott a blokk használatával többet is használhatunk.

Feltételes utasítás: Akkor használjuk, ha két lehetőségből kell választani. if-else a példa rá, ha az if igaz akkor azt végzi el, különben az else ágat.

While utasítás: Végrehajtódik az utasítás mindaddig amíg amíg a kifejezés értéke nemnulla marad.

Do-while utasítás: Mindaddig ismétlődik az utasítás amíg a kifejezés értéke nullává nem válik.

For utasítás: Képesek vagyunk megadni, hogy hányszor hajtódjon végre az utasítás, és hogy mi legyen a feltétel.

Switch utasítás: ez a többágú feltételes utasítás: egy kifejezést több esetre bonthatunk, case:-kre valamint van egy default: eset is.

Break utasítás: Befejeződik az őt körülvevő while, do-while, for vagy switch utasítás.

Continue: átugorja a többi utasítást és az őt körülvevő while,do-while,for utasítás ciklusfolytató részére ugrik a vezérlés.

Return utasítás: A függvény a hívójához a return utasítással tér vissza.

Goto utasítás: a vezérlés feltétel nélkül adható át az adott helyre.

Cimkézett utasítás: Azutasítások címkével láthatók el, amely a goto célpontjaként szolgál.

Nulla utasítás: hordozhat címkét, vagy képezhet üres ciklustörzset.

10.3. C++ programozás

Rövid olvasónapló a BMECPP könyvről.

Az első fejezetben megismerhetjük a C++ nyelv rövid történetét, objektum-orientált tulajdonságait és a generikus programozás fogalmait. A C++ C nyelvre való épülése fontos dolog, amit mindenkinek tudni kell, és ezen nyelvek szoros, testvéri kapcsolatáról olvashatunk.

A második fejezetben néhány nem objektum orientált újdonságait ismerhetjük meg a C++ nyelvnek. Összehasonlításra kerül a C nyelvvel szintén, a tetszőleges számú paraméterrel való hívás példája van kiemelve (C ben void f() míg C++ban void f(void)). Megismerjük a main függvény használati módját is, valamint a return 0 nem kötelező használatát ebben a függvényben.

A bool mint logikai típus bevezetése is meg van említve, mely a C nyelvben még nem szerepelt. Több bájtos stringek fogalma is ismertetésre kerül, és ezek használatához szükséges includeolandó könyvtárak is fel vannak sorolva. A változó deklaráció mint utasítás is szóbajön, azaz minden olyan helyen állhat változódeklaráció, ahol utasítás is állhat. Megismerjük a függvények túlterheltségét is: C ben a függvény neve

azonosítja egyértelműen a függvényt, míg C++-ban a függvény neve és az argumentumlistájuk együttesen azonosítja. Ezért történhet meg az, hogy C++-ban azonos néven létezhet két függvény, ha az argumentumlistájuk különböző és egyedi.

Lehetőségünk van arra is a C++ nyelvben, hogy a függvény argumentumainak alapértelmezett értéket adjunk meg.

A C++-nyelven továbbá lehetőségünk van paraméterátadásra referencia típussal. Ez azt jelenti, hogy a változó címét adjuk át, nem pedig az értékét, és ez nagyon hatékony tud lenni egyes esetekben.

A C++ a cím szerinti paraméter átadást referenciákkal valósítja meg, ezt a mechanizmust referencia szerinti paraméterátadásnak nevezzük.

A függvénynek átadott argumentum könnyű megváltoztathatósága a referenciának csak az egyik alkalmazási területe. ezért például nagy méretű argumentumok: pl. struktúrák esetén teljesítménynövekedést érhetünk el.

Objektumok és osztályok

Az objektum orientáltság bevezetése több alapelvet követett: legyen átláthatóbb a program, a program bonyolultsága ne növekedjen drasztikusan stb.

Az osztályoknak lehetnek példányai, önálló egyedei melyeket objektumoknak nevezünk, és ezek az objektumok tudnak egymást közt kommunikálni. Egy adott témakörhöz, pl. a Számlához létrehozhatunk egy osztályt, abban egyedeket, és műveleteket. Ezek egy egységbe záródnak, és értelemszerűen együtt működnek.

Az objektum orientáltság egy szemléletmód, ami a modern felfogást és gondolkodást szemlélteti, valamint az évek múlásával és a programozás fejlődésével alakult ki.

Egységbe záras a C++-ban: Egy C++ programban megvalósíthatók olyan programok is, melyek tartalmazznak tagváltozókat (struktúra adattagjai), és tagfüggvényeket (osztály részeként, lehetséges osztályon belül vagy osztályon kívül).

Adatrejtés: Az egységbe záráshoz kapcsolódik, mely átláthatóbbá teszi a programunkat. Lehetőség van rejtetni az adatainkat,

private: részként megadni a struktúrában,

public: részben pedig amit nem védünk.

Osztályon belül alaphoz elendő csak a public: kihangsúlyozása

Konstruktorok, destruktorkok: Az osztályunkba ha nem írunk konstruktort, akkor alaphoz létezik egy olyan konstruktor ami nem csinál semmit. Ha írunk paraméteres konstruktort, akkor példányosítani tudjuk az osztályunkat mikor objektumot hozunk létre. Tehát a konstruktor szerepe az inicializálás, a destruktork ~ jellel kezdődik és akkor hívódik, ha az objektum megszűnik: felszabadul.

Dinamikus memóriakezelésnek nevezzük azt amikor new utasítással foglalunk helyet a free storeban, a delete szóval pedig töröljük azt.

Dinamikus adattagokat a dinamikus memóriakezelés során hozzuk létre, szóval nem gyártjuk le előre az adattagokat mint a gyár, hanem csak mikor kell, akkor hozunk létre újakat, és foglalunk neki annyi memóriát amennyit kell.

A másoló konstruktor is egy konstruktor, mellyel az a célunk, hogy már meglévő objektum alapján az újonnan létrehozott objektumot inicializáljuk, tehát egy másolatot szeretnénk létrehozni. A másolókonstruktornak átadott argumentumból kell egy másolatot létrehozni, a függvényparamétert inicializáljuk az átadott értékkel.

A friend függvények és osztályok azzal a jellemzővel bírnak, hogy feljogosítanak bizonyos más osztálybeli tagfüggvényeket vagy globális függvényeket a saját védett tagváltozói és tagfüggvényei elérésére. Ezt a feljogosítást a friend kulcsszóval tehetjük meg. Az osztály tervezője mondja meg, hogy milyen függvények és vagy osztályok férhetnek hozzá a saját osztályához.

A tagváltozókat inicializálhatjuk konstruktorainkban, a : karakter után felsoroljuk az inicializálni kívánt tagváltozókat. Fontos, hogy az inicializálás kezdő értéket állít be, azaz konstruktorhívás, míg az értékadás egy meglévő objektumnak ad értéket.

Az osztályon belül létrehozhatók statikus tagok is, melyek tulajdonsága, hogy az adott osztályhoz tartoznak, nem azok objektumaihoz. A statikus tagok lehetnek statikus tagváltozók és tagfüggvények. A statikus tagok objektum nélkül is használhatók. A memóriában egy helyen vannak. A statikus tagfüggvények statikus változókkal dolgoznak. Ezeket is a static kulcsszóval lehet jelezni. Akkor kell statikus tagváltozókat használni, ha az osztály minden objektumára közös változóra van szükségünk. Ez a helyzet a statikus tagfüggvényekkel is. Fontos tudni, hogy a statikus tagváltozók az alkalmazás indításakor inicializálódnak, a main függvénybe való lépés előtt, a globális változókkal egyidejűleg.

10.4. Python nyelvi bevezetés

Rövid olvasónapló a BMEPY könyvről.

Python nyelv bemutatása

A python nyelv átlagos számú elérhető funkciókkal a többi nyelvhez képest, nagyon gyors fejlesztési gyorsasággal rendelkező és sok támogatott eszköz számmal rendelkező magas szintű, általános célú programozási nyelv.

Guido van Rossum 1990-ben alkotta meg, a célja a rengeteg pozitív tulajdonsággal rendelkező, magas szintű, dinamikus, objektumorientált és platform független nyelv megalkotása volt.

Fontos jellemzője a Python nyelvnek, hogy fordítóra nincs szükség. A Python interpreter számos platformon elérhető : Windows, MacOS X, Unix...

A Python nyelv szintaxisában nem találhatók meg a jól ismert begin, end, pontosvessző használatra, mivel behúzásalapú a szintaxisa. A sor végéig tart az utasítás. Az értelmező minden sort tokenekre bont amelyek közt tetszőleges whitespace karakter lehet. Lefoglalt kulcsszavak közül néhány: and, del, for, if, is, elif, while, print, import, class, break, return..

Megismerkedünk a Tipusok és változók fogalmával a Python nyelvben. Nincs szükség a változók típusainak explicit megadására, mivel a rendszer futási időben kitalálja azt. Néhány adattípus a Python nyelvben: számok, stringek, listák, szótárak..

A felsoroltak közül az ennesek lehet ismeretlen számunkra, ezek az objektumok gyűjteményei vesszővel elválasztva. Ezeket általában zárójelek közé írjuk, vesszővel elválasztva, pl: ('a','b','c'), (), (1,"szia",3)..

A lista elemeit szögletes zárójelek közé írjuk pl: ['a','b','c'], list('abc')..

A szótár kulcsokkal azonosított elemek rendezetlen halmaza. Pl: {'a':1, 'b':5, 'e':1982}..

A pythonban a NULL érték neve a None.

A nyelvben a változóknak nincsenek típusai, akár több típusú objektumra is hivatkozhatnak. Pl: a=b=c=1; x,y=y,x (felcseréli a két változót).

A del kulcsszóval törölünk változó hozzárendelést. Itt is léteznek globális illetve lokális változók, alpból lokális.

Néhány listán végezhető műveletre példa: count(e) visszaadja az e előfordulásainak számát,

insert(i,e) beszúrja az e elemet az i-edik helyre,

sort([f]) sorbarendezi(helyben) a lista elemeit az f függvény felhasználásával.

copy() visszatér a szótár egy másolatával,

keys(), iterkeys() a szótár kulcsait tartalmazó listával, illetve iterátorral tér vissza.

clear() kitörli az összes elemet a szótárból.

A könyv további részében megismerkedünk a nyelv eszközeivel. Ide tartozik a print, az elágazás(if elif else), ciklusok(for), while(i kisebbmint 3): print i stb.

Találhatók címkék, ugrások is a nyelvben, valamint függvények, melyeket def kulcsszóval definiálhatunk, pl:

```
def hello(): print "Hello" return
```

A python nyelvben is találhozhatunk az osztályok és az objektumok témakörével, melyek klasszikus objektumorientált fejlesztési eljárások. pl: class Koszonto: def MonddSzia(self, ember): print 'Kedves', ember, ', udvozollek.'

A python nyelvben még megismerkedhetünk a modulokkal (mobilkészülékeken való fejlesztés megkönnyítésére), kivételkezeléssel, mely hasonló alapon működik mint a többi objektum orientált nyelv esetén.

És még a végén találhatunk részletes példákat forráskódokkal az eddig tárgyalt témakörök megalapozásának érdekében.

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

DRAFT

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

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

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