

Programozói kézikönyv

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Programozói kézikönyv		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Lucski, Attila	2019. szeptember 23.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	12
2.6. Helló, Google!	12
2.7. 100 éves a Brun tétel	15
2.8. A Monty Hall probléma	16
3. Helló, Chomsky!	18
3.1. Decimálisból unárisba átváltó Turing gép	18
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	19
3.3. Hivatkozási nyelv	20
3.4. Saját lexikális elemző	21
3.5. l33t.1	22
3.6. A források olvasása	24
3.7. Logikus	26
3.8. Deklaráció	26

4. Helló, Caesar!	28
4.1. int *** háromszögmátrix	28
4.2. C EXOR titkosító	30
4.3. Java EXOR titkosító	31
4.4. C EXOR törő	33
4.5. Neurális OR, AND és EXOR kapu	36
4.6. Hiba-visszaterjesztékes perceptron	39
5. Helló, Mandelbrot!	46
5.1. A Mandelbrot halmaz	46
5.2. A Mandelbrot halmaz a std::complex osztállyal	50
5.3. Biomorfok	52
5.4. A Mandelbrot halmaz CUDA megvalósítása	55
5.5. Mandelbrot nagyító és utazó C++ nyelven	59
5.6. Mandelbrot nagyító és utazó Java nyelven	68
6. Helló, Welch!	81
6.1. Első osztályom	81
6.2. LZW	84
6.3. Fabejárás	88
6.4. Tag a gyökér	93
6.5. Mutató a gyökér	99
6.6. Mozgató szemantika	104
7. Helló, Conway!	111
7.1. Hangyaszimulációk	111
7.2. Java életjáték	129
7.3. Qt C++ életjáték	137
7.4. BrainB Benchmark	146
8. Helló, Schwarzenegger!	170
8.1. Szoftmax Py MNIST	170
8.2. Szoftmax R MNIST	170
8.3. Mély MNIST	170
8.4. Deep dream	170
8.5. Robotpszichológia	171

9. Helló, Chaitin!	172
9.1. Iteratív és rekurzív faktoriális Lisp-ben	172
9.2. Weizenbaum Eliza programja	172
9.3. Gimp Scheme Script-fu: króm effekt	172
9.4. Gimp Scheme Script-fu: név mandala	172
9.5. Lambda	173
9.6. Omega	173
10. Helló, Gutenberg!	174
10.1. Juhász István - Magas szintű programozási nyelvek 1; olvasónapló	174
10.2. Kerningham és Richie; olvasónapló	175
10.3. BME: Szoftverfejlesztés C++ nyelven / Benedek Zoltán, Levendovszky Tihamér ; olvasónapló	176
10.4. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven es Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.	178
10.5. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba	179
11. Helló, Arroway!	182
11.1. OO szemlelet	182
11.2. Homokozó	185
11.3. Gagyí	192
11.4. Yoda	193
11.5. Kódolás from scratch	194
12. Helló, Gutenberg!	196
12.1. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven es Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.	196
12.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba	201
III. Második felvonás	203
13. Helló, Arroway!	205
13.1. A BPP algoritmus Java megvalósítása	205
13.2. Java osztályok a Pi-ben	205

IV. Irodalomjegyzék	206
13.3. Általános	207
13.4. C	207
13.5. C++	207
13.6. Lisp	207

DRAFT

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

```
#include <unistd.h>
int main ()
{
    for (;;)
        sleep (1);
    return 0;
}
//összes mag

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... A `for (; ;)` ciklust parametereknek nélkül használva végtelen ciklusként funkcionál, a `sleep` függvény alattja a szálát a parameterül adott milliszekundumig ami jelen esetben egy milliszekundum, ekkor egy magot terhejük a processzornak. A `#pragma omp parallel` kódot használva a ciklus az összes magot terheli maximumon nem csak egyet mivel ezzel parhuzamosítunk.

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

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

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

    boolean Lefagy2(Program P)
```



```
{
    if (Lefagy (P))
        return true;
    else
        for(;;);
}

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

A feladatnak nincs megoldása mivel: Ha a vizsgált program tartalmaz végtelen ciklust: Lefagy=true és Lefagy2 szintén true lesz. Ha a vizsgált program nem tartalmaz végtelen ciklust: Lefagy=false=>Lefagy2=false ciklus lesz.

Ilyen program nem létezik, de azt ellenőrizhetjük hogy van-e elég memória a gépünkben a program használatához mert ha elfogy a memória akkor is lefagy.

2.3. Változók értékének felcserélése

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

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

```
#include <stdio.h>

int main()
{
    int a = 2;
    int b = 1;
    b = b-a;
    a = a+b;
    b = a-b;
```

```
printf(a);  
printf(b);  
return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat... A printf függvény I/O típusú így include-olni kell a stdio-t. A két változó deklarálása és inicializálása után a b változó értékét módosítjuk arra hogy a b változó értéke mennyivel nagyobb az a változó értékénél. E különbséget és az a változó értékének összegére módosítjuk az a változó értékét így az a változó értéke a b változó inicializált értéke lesz. Majd a b változó értékét módosítjuk úgy hogy a változó értékéből ami az inicializált b változó kivonjuk a b változó értékét ami a különbség így a b változó értéke a a változó inicializált értéke lesz. Majd a printf függvény használva kiíratjuk az a változó értékét és a b változó értékét és ellenőrizzük a cserét.

2.4. Labdapattogás

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

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

```
#include <stdio.h>  
#include <urses.h>  
#include <unistd.h>  
  
int main ( void )  
{  
    WINDOW *ablak;  
    ablak = initscr ();  
  
    int x = 0;  
    int y = 0;  
  
    int xnov = 1;  
    int ynov = 1;  
  
    int mx;  
    int my;  
  
    for ( ;; ) {  
  
        getmaxyx ( ablak, my , mx );  
  
        mvprintw ( y, x, "O" );  
  
        refresh ();  
        usleep ( 100000 );  
    }
```

```
clear();

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

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

Tanulságok, tapasztalatok, magyarázat... Az `int main (void)` függvény `void` parameter azt jelenti hogy a `main` függvény csak parameter nélkül lehet hívni. A `WINDOW *initscr()` függvény `curses` modba kapcsolja a terminált. Mivel `ncurses` típusu így használatához `include`-olni kell a `curses.h`-t és fordítaskor `-lncurses` parancsot kell használni. Az ablak változó arra szükséges hogy az `initscr` függvény tudjunk hivatkozni. A `for` ciklus végtelen ciklus így addig fut amíg `^C` billentyűparanccsal le nem állítjuk. A `getmaxyx` függvény megadja az ablak maximum oszlop és sorszámát ami az `my` változó és `mx` változó. Mivel nem inicializáltuk ezen változókat így a teljes terminalon lehetnek koordináták és egerrel állíthatjuk a terminal méretet. Az `mvprintw` függvény `y` változó és `x` változó meghatározza a kurzor helyzetet és a 3. parameterként megadott szöveget ideiglenes tárolóba menti. E tárolót a `refresh` kiuríti és az adatait megjeleníti a terminalba. A `usleep` késlelteti a ciklusok végrehatási idejét a parameterként megadott microsekundummal és használatához `include`-olni kell az `unistd.h`-t. A `clear` törli a terminal tartalmát. Majd az `x` és `y` növeljük az `xnov` `ynov` melynek értékeit inicializáltuk. Az `x` és `y` megadja a kezdőkoordinátákat és a `xnov` és `ynov` a mozgás irányát. Az `if` ellenőrzi hogy az `x` és `y` értékei nagyobb vagy egyenlőek e a `mx` és `my` értékeivel vagy kisebb vagy egyenlőek e a 0-nal. Ha teljeseül a feltétel `x` akkor az `xnov` értéke vált előjelet ha az `y` teljesül akkor az `ynov` értéke vált előjelet. A labda indulási koordinátáit az `x` és `y` értékei határozzák meg mozgását az `xnov` és `ynov` értékei és az `x` `y` `xnov` `ynov` változók módosítása.

A feladat `if` nélkül:

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

static void gotoxy(int x, int y)
{
    int i;
    for(i=0; i<y; i++) printf("\n");
    for(i=0; i<x; i++) printf(" ");
    printf("o\n");
}
```

```
}

void usleep(int);
int main(void)
{

    int egyx=1;
    int egyy=-1;
    int i;
    int x=10;
    int y=20;
    int ty[23];
    int tx[80];

    for(i=0; i<23; i++)
        ty[i]=1;

    ty[1]=-1;
    ty[22]=-1;

    for(i=0; i<79; i++)
        tx[i]=1;

    tx[1]=-1;
    tx[79]=-1;

    for(;;)
    {
        for(i=0; i<36; i++)
            printf("_");

        printf("x=%2d", x);
        printf("y=%2d", y);

        for(i=0; i<=35; i++)
            printf("_");

        (void)gotoxy(x,y);

        x+=egyx;
        y+=egyy;

        egyx*=tx[x];
        egyy*=ty[y];

        usleep (200000);
        (void)system("clear");
    }
}
```

Tanulságok, tapasztalatok, magyarázat... A static függvényeket csak abba a fájlba használhatjuk ahol definiáltuk őket így nem probléma ha másik fájlba ugyanolyan nevű függvény van mert a C függvények globálisak így más fájlba használhatjuk őket. A gotoxy függvény a paraméterként megadott (x,y) koordinátára állítja a kurzort. A függvénynek maximum koordinátái vannak x 80 y 25 így csak 80x25 területen fog csak pattogni a labda. Az (1,1) a bal felső sarokba van de a labda az inicializált x változó és y változó indul így a pattogtatáshoz a koordinátákat ellentétesen kell növelni csökkenteni. A 80 maximum miatt ha x=80-ról indítjuk a labdát szegmentálási hibát kapunk. Minimum koordinata nincs így y=1 és x kisebb mint 80-ról indítva a labdát a y=1 soron mozog balról jobbra az x=1 és 80 koordináták között. (0,0) és negatív koordinátákról is szegmentálási hibát kapunk. A szegmentálási hibát kapunk futás közben ha y=25 vagy 24-ről indítjuk a labdát megfelelő x koordinátával ha eléri a program az y=23 koordinátát. Mivel az y értéket az egy az egy pedíg a `ty[y]` vektor értéke határozza meg. A 25-nél indulva a kiírja a (x,25)-re az o-t majd növeli az y-t az egy kezdőértékével -1-el így lesz 24. Aminek lekéri az értéket úgy a `ty[y]`: `ty[24]`-et nem fog találni így a `t[i]`-t keresi majd melynek értéke 1 lesz. Az i értéke a tartozó for meg 0,1,...22 volt de utána lefutott meg 3 for ciklus i változóval újabb értékekkel közte a 24-el. Úgy a `ty[24]`-nek nincs értéke, de van `i=24` és `ty[i]=1`. A `ty[y]` for ciklusba a `ty[i]=1` olyan i-ket kap ami megfelel a feltételnek így ezt nem ellenőrizi mivel nincs utközesbe sem semmilyen logikai i-re vonatkozó feltétellel a `ty[i]=1`-hez tartozó for ciklusba úgy a többi feltételt nem venne figyelembe. A végtelen cikluson kívül eso 2 for ciklusban 1 értéket adunk a nem szélső sorok és oszlopok koordinátáinak a szélsőre -1 lesz. A vektorok nevét a többi globális változónál deklaráltuk. 2 random generált értékkel amit befolyásol az utána levo for ciklus és hogy milyen nevet adtunk a vektornak. Kaphat 0 értéket is ekkor vízszintesen vagy függőlegesen megy a labda egy vonalba. Lehet kiugratni is a kurzormozgató billentyűkkel felugrott -13567-re 500-ról és ekkor felül a vonalig ment balra mert a felső sarkot nem érte el az odairt vonalaktól amit a végtelen ciklusba írunk meg. A jobb oldali vonal volt a 0 attól ment jobbra a 79 értékig. Ez is egy vonalba közlekedett csak mashogyan mert elcsuszott a szélsőkoordináták ahogy irtam a vonalaktól. Ekkor is 0 volt az egy így vízszintesen pattogott. Peldaul Y=28-nál x+1-el pedig érdekes ívet jár a labda. A végtelen ciklusba a 2 for ciklusba 2x35 vonalat írunk `printf` függvény és közé 2x4 helyen pedíg az x és y koordinátákat kiírjuk. Majd a kezdőpontként megadott (x,y) koordinátára állítjuk a labdát a gotoxy-val. Majd az x változó és y változó értéket az egyx változó és egyy változó értékeivel növeljük. Aztán a `tx[x]` vektor és `ty[y]` értékeivel megszorozzuk az egyx változó és egyx változó így sarkokra érve irányt változtat a labda. Majd a `usleep` függvény hívva a megadott microsecondommal késleltetjük a labda minden mozgását és használatához kell a `time.h` header file. A `(void)system("clear")` függvény használatához `include-lni kell a stdlib.h header file-t és torli a konzol tartalmát. Így marad minden lépés után a vonal felül mert torlás nélkül a kurzor lenne a bal alsó sarokba tehát elcsuszna. (1,1)-ről indítva is érdekesen pattog a labda. A gotoxy függvény deklarálásánál a printf függvény megadhatjuk milyen karakter lejtetünk. Az int main() függvény futása után elvégzi a deklarációt a gotoxy függvény vegett a printf függvény bevonalazza és kiírja az x és y-t az első sorba majd a gotoxy függvény hívásakor. Ez több időt vesz igénybe így ezalatt megpróbálja ugyanonnan kiírni mind 3 printf függvény sikertelenül. Majd az első tudja írni az első sor után mert észleli hogy /n van benne új sort kér. Így leírja a 20 Entert majd következik a Szokozos amit nem tud megkezdeni az első sorból mert már odairta a korábbi vonalás for ciklus és a sor vége meg az Enterest írta. Így keresi a 3. printf függvény kiíratási helyet ami mivel Entert tartalmaz A printf függvények használatánál az utolsó printf függvény '\n'-re kell végződnie. A (void)system("clear") minden for ciklus végén torli a terminált így újra pozícionálja a definiált gotoxy.`

A program különleges parancsai a "clear" (mely a egyetlen labdához) és a "usleep" (mely a "sima" pattogáshoz kell).

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

```
#include <stdio.h>

int
main (void)
{
    int m = 0;
    int n = 0x01;
    do
        ++m;
    while (n <= 1);
    printf ("A szo %d bites\n", h);
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat... A 0x01 16-os számrendszerben van x felosztja ketfele 2x4 számjegyre az egyszerűbb atvaltas miatt mert az egyes 4 számjegyu tagok erteke 0 es 15 koze esik ami a 16-os számrendszer számkeszlete. A es F kozti angol abc betukkel jeloljuk a 10 es 15 kozti hexadecimalis számjegyeket. A 0x01 az kettes számrendszerben 0000-0001 ami tizes számrendszerben 1. Tizes számrendszerbe atvaltani ugy lehet hogy a szám vegeleleje fele felirjuk egyesevel a számrendszer amiben van hatvanyait 0-tol a szó hossza -1-ig majd egyenkent megszorozzuk a hozzatartozo szám ertekevel majd összeadjuk a szorzatokat. A while ciklus csinalja az utanna irt utasitast jelen esetben noveli egyesevel az m változo ertekeket amig a while feltetel nem ad vissza hamis ertekeket. A while ciklus felteleben levo n balshift 1 = n fut n=0-ig amikor a feltetel hamis lesz mert akkor már nem tudja shiftelni n-t így a kovetkezo gepiszoba kerül at a shiftelt 1-es. A printf fuggveny "%d" resze azt jelenti hogy veszi a kovetkezo argumentumot es kiirja egész számként. A gepi szó 32 bit így az opreacios rendszerem 32 bites ami max 4GB memoriát támogat. Az hogy hany bites operacios rendszert használhatunk a legfokepp processzor bitszama befolyasolja. A printf fuggveny parameterebe h változo helyett n változo hasznaltam. A BogoMips a Linux operációs rendszerben nyújtott mérés amely relatív módon jelzi hogy a számítógép processzora milyen gyorsan fut.

2.6. Helló, Google!

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

Megoldás videó: https://www.youtube.com/watch?v=P8Kt6Abq_rM

```
#include <stdio.h>
#include <math.h>
void
kiir (double tomb[], int db)
```

```
        {
            int i;
for (i=0; i<db; ++i){</programlisting>
    printf("%f\n",tomb[i]);
}
}

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

void
pagerank(double T[4][4]){
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény
    double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0}; //ezzel szorzok
    int i, j;
    for(;;){
        for (i=0; i<4; i++){
            PR[i]=0.0;
            for (j=0; j<4; j++){
                PR[i] = PR[i] + T[i][j]*PRv[j];
            }
        }

        if (tavolsag(PR,PRv,4) < 0.0000000001)
            break;
        for (i=0;i<4; i++){
            PRv[i]=PR[i];
        }
    }

    kiir (PR, 4);
}

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

    double L1[4][4] = {
        {0.0, 0.0, 1.0/3.0, 0.0},
        {1.0, 1.0/2.0, 1.0/3.0, 0.0},
```

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

double L2[4][4] = {
    {0.0, 0.0, 1.0/3.0, 0.0},
    {1.0, 1.0/2.0, 1.0/3.0, 0.0},
    {0.0, 1.0/2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0/3.0, 1.0}
};

printf("\nEredeti:\n");
pagerank(L);
printf("\nSemmire sem mutat:\n");
pagerank(L1);
printf("\nCsak magára mutat:\n");
pagerank(L2);
printf("\n");
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat... A PageRank algoritmus alapötlete, hogy azok a weblapok jobb minőségűek, amelyekre jobb minőségű lapok mutatnak. A Google alapjai ezen a Pagerank rendszeren nyugszanak.

A pagerank függvény bemenete 4x4-es matrix. Az L matrix oszloposan sztohasztikus minden oszlopanak összege 1, ez szükséges feltétele a helyes PageRank értékek számításának. A PR vektorba menti a kijött eredményt amit a PRv kezdő PageRank értékekből számol ki az L matrix segítségével. PRv kezdőértéke minden honlapnak 1/honlapokszáma. Az aktuális honlapnak aminek PageRank értéket számoljuk úgy a ramutato lapjára ahány lap mutat annyival le kell osztani a PageRank értékeit majd összeadni minden aktuális honlapra mutatóval ugyanígy. Az L-ben tároljuk hogy a mennyivel kell szorozni a PRv-t hogy hány lap mutat rá, csak reciprokot szorzunk. Ezen literációkat szamat addig növeljük míg nem fog sokat változni már a PageRank értéke, ezt a távolság függvénnyel mérjük. Ha nagyon minimális az eltérés akkor break-kel kilép a végtelen for ciklusból és a kiír függvény jön. Egyébként a PRv vektorba átírja a PR értékeit, a PR értékeit pedig a végtelen ciklus első for ciklusa kinullazza. A távolság függvény az újabb literáció és a literáció különbségeinek négyzetösszegének gyökét hasonlítja a 0.0000000001-hez. Mivel az sqrt függvényt használjuk gyökvonashoz így futtatni az -lm kapcsolóval kell és a használatához szükséges beletölteni a math.h header fájlt. A kiír függvény pedig a parameterül adott vektornak a parameterül adott elemszámait írja ki 0-tól a parameterig. A távolság függvény pedig a parameterül adott 2 vektor a 3. parameterül adott egész számmal kisebb elemeivel dolgozik. A printf %f paraméter a double típusú kihatáshoz szükséges. A kiír és a pagerank függvény void típusú mivel nincs visszatérési azot meghívó függvény fele. A távolság double típusú mert nagy nemcsak egész hanem lebegőpontos számokkal számolunk és azt eredményként adja vissza a hívó függvénynek. Globális függvények is. Az L1-ben a D nem mutat semmire így nem növeli csak csökkenti az összértéket így a 0-hoz fognak tartani a PageRank értékek. Az L3-ban a D úgy csökkenti az összértéket minden ciklusban de növeli a sajátját és tart az 1 fele. Az A, B, C összértéke kb arányosan csökken azzal az értékkel amivel a D no ciklusonként. Ha megy egy elemre mutatnak akkor visszakell mutatnia onamgan kívül valamelyikre.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

```
library(matlab)
stp <- function(x) {
  primes = primes(x)
  diff = primes[2:length(primes)]-primes[1:length(primes)-1]
  idx = which(diff==2)
  t1primes = primes[idx]
  t2primes = primes[idx]+2
  rt1plust2 = 1/t1primes+1/t2primes
  return(sum(rt1plust2))
}

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

A Brun-tétel azt mondja, hogy az ikerprímszámok reciprokaiból képzett sor összege véges vagy végtelen sor konvergens, ami azt jelenti, hogy ezek a törtek összeadva egy határt adnak ki pontosan vagy azt át nem lépve növekednek. A program ezenek az ikerprímek összegzett reciprokaikat rajzolja ki.

A program R nyelven van kodolva így futtatáshoz R fordító szükséges, ahol telepítve van a matlab csomag. Ubuntu-ban `install.packages("matlab")` paranccsal tudjuk telepíteni miután beletűnt az R paranccsal az R környezetbe. A program első sorával beletűnt a telepített matlab könyvtárba. R nyelvben a változoknak nem kell típusot adni, az `stp` után írt operátorral adunk értéket. Itt az `stp` változéhoz rendeljük a `function(x)` függvényt így `stp(x)`-el használhatjuk mivel a `function()` beépített függvény így használatához a hozzárendelt névvel tudunk hivatkozni. Letrehozzuk a `primes` vektort, úgy `=` operátorral hozunk létre vektorokat. A `primes` vektor értékei a beépített `primes(x)` függvény visszatérési értékei. A `primes` függvény a megadott `x` paraméterig írja ki a prímeket `x`-et beléve. Tul. nagy paraméter megadása esetén hibajelzettel írja ki amennyit ki tud és kijelzi hányat nem tudott megírni. Majd a `diff` vektorba mentjük a különbségeket a `primes` vektor 2. eleme és 1. elem között az utolsó és az előtti elem között. R nyelvben a vektorok elemeinek a száma 1-től indul. A `diff` meghatározására meg használjuk a `length` beépített függvényt ami megadja az adott vektor elemeinek a számát vagyis a legnagyobb érték hivatkozási számát, sorsszámát. R-ben 2 vektor kivonása paraméterként működik. A `x:y` operátor az `x`-tól `y`-ig menő szabályos sorozat aminek elemeit a `primes` függvény paraméterként egyenként felhasznál. Az `idx` vektor tartalmazza az ikerprímek első tagjának indexét `x`-ig. Ezeket a `t1primes` vektorba írjuk ki, a `t2primes`-be pedig az ikerprímek második tagjait. Az ikerprímek olyan prímszámok melyeknek különbsége 2. A prímszámok pedig olyan pozitív egészs számok melyeknek 2 különböző osztóval van 1 és onmaguk. A `which` beépített függvény ellenőrzi hogy mely indexű elemekre teljesül a feltétel úgy a `diff` vektor melyik eleme 2 és menti ezeket az `idx` vektorba. Az `rt1plust2` vektorba összeadjuk az ikerprímek reciprokait. A `t1primes` reciprokait és `t2primes` reciprokait vektorok elemeit elemeenként adjuk össze és mentjük az `rt1plust2` vektorba. A reciproka a szám 1-gyel osztva. Majd `return`-be adjuk az `rt1plust2` vektor elemeinek az összeget ami az eredeti függvény visszatérési értéke lesz. A `sum` beépített függvény megadja a vektor elemeinek az értékeinek az összeget. Az `x` vektornak megadjuk az elemeit a `seq(x,y,by=z)` függvénnyel ami szabályos sorozatot ad vissza `x`-tól `y`-ig `z`-vel növekedve. A `sapply` függvényt használunk kell `function(x)` függvények, nem beépített függvények

kirajzoltatása esetén. A `sapply(x,FUN)` meghatározza a paraterul adott x koordinatakra a FUN függvényhez rendelt függvény értéket és ezeket az értékeket mentjük az y vektorba. A plot függvény kirajzoltatja egy külön ablakba egy koordinatarendszerbe a megadott vektorokat elemként parositva. A type paramterbe megadhatunk típusokat a "b" azt jelenti hogy korokat és vonalakat is rajzoljon.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan](https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall_paradoxon_kapcsan)

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

for (i in 1:kiserletek_szama) {
  if(kiserlet[i]==jatekos[i]){
    mibol=setdiff(c(1,2,3), kiserlet[i])
  }else{
    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))
  }
  musorvezeto[i] = mibol[sample(1:length(mibol),1)]
}
nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)
for (i in 1:kiserletek_szama) {
  holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvalt[sample(1:length(holvalt),1)]
}
valtoztatesnyer = which(kiserlet==valtoztat)
sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

A Monty Hall probléma lényege hogy amennyiben 3 "ajtó" közül választunk(1 ajtó mögött a "nyeremény"),egyét. Ezután kinyitnak egy ajtót amely üres; a probléma lényege pedig az ,hogy ha új ajtót választunk, akkor nagyobbak az esélyeink a sikerre. A 21 – Las Vegas ostroma filmben is szóra került.

A kiserletek_szama egyelemu vektor melynek erteke meghatározza a kiserletek szamat ami jelen esetben 10000000. A kiserlet vektort a sample függvennyel feltöltjük adatokkal, ez tartalmazza a nyeremeny szamat ami 1 és 3 kozse eso szam egy kiserlet eseten, de kiserletek_szama mennyiséggel kell megismetelni annyi a vektor hossza. A sample(x,y,z) beépített függvény ahol x helyere kerül az amik kozul general a függvény, y helyere hogy milyen hosszan generalja a szamsorozatot, z helyere replace=T vagy replace=F kerül. T-nel megengedett az ismetles a generalas soran, F eseten nem. 1:3 az 1 2 3 szamokat jelentik, ezekbol general kiserletek_szama-nyit. A jatekos vektort ugyanigy a random szamgeneralassal töltjük fel, ez az amit a

jatekos eloszor választ ki. Majd létrehozuk a musorvezeto vektort a beepített vector parancs segítségével. A vector parameterebe megadjuk a létrehozando vektor hosszat a length ertekevel ami a kiserletek szama. A musorvezeto vektort egy for ciklus segítségével töltjük fel adatokkal. A `for(i in 1:kiserletek_szama)` azt jelenti hogy kiserletek_szama-nyiszor hajtódik végre a for ciklus, a benne megadott utasítás. Az utasítás részben töltjük fel a musorvezeto vektort 2 lépésben a mibol vektor segítségével. Eloszor ellenorizzuk a hogy a jatekos a nyeremenyt választotta e az if utasítás segítségével, ha igen akkor a musorvezeto 2 közül választhat mivel a nyeremenyt nem választhatja ki. A választási lehetőségeket a mibol vektorba mentjük amit a `setdiff` függvény segítségével határozzuk meg. A `setdiff` beepített függvény kiszámolja a parameterkent adott adatok különbséget, jelen esetben kivonja az első parameter értékeből a második értéket. Vektort megadhatunk úgy hogy felsoroljuk az elemeit vagy meglevő vektorokat fűzhetünk össze. Az if utasítás else ágba kerül az ha nem a nyeremenyt választja a jatekos, ekkor a musorvezeto csak egyet választhat mivel a választási lehetőségei közt van a nyeremény amit úgye nem választhat. Így ekkor a mibol vektor értéke az 1,2,3 számokból kivesszük a jatekos által választott úgye nem nyeremény számát és a kiserlet azaz a nyeremény számát. Majd a musorvezeto vektort feltöltjük úgye egyenként a for ciklussal a mibol vektor megfelelő elemeivel. Úgye ha egy elem a mibol vektor tehát nem a nyeremenyt választotta a jatekos akkor a mibol vektor első eleme lesz a musorvezeto i-edik értéke úgye mindegyik változónal i-edik elemmel dolgozunk csak a mibol-nel nem a for cikluson belül. Ha pedig a nyeremenyt választotta a jatekos akkor a mibol vektor 2 elemű így a `sample` függvénnyel generalva 1 vagy 2-t dönti el a mibol vektor hanyadik számú eleme lesz a musorvezeto i-edik eleme. A nemváltoztatesnyer vektorba mentjük azokat az indexeket ahol a jatekos eloszor a nyeremenyt választotta. Úgye mindegyik vektorra passzolnak ezek az indexek. Kiíratásnál mivel sok adat volt így soronként 10 elemmel írja ki az elemeket legszelen pedig 10-el no soronként a vektor indexe. A változtat vektort létrehozuk a masodjara választásos esetekre ahol a jatekos újat választ a musorvezeto választása után ebbe mentjük az általa választott számot ami egy lehet mivel az előzőn kívül választ és a musorvezetojet sem választhatja. A musorvezeto vektorhoz hasonlóan megadjuk a vektor méretét ez is ugyanakkorra lesz mivel mindegyik esettel számolunk. Az adatok feltöltése hasonlóan segédváltozóval történik jelen esetben a `holvalt`-tal, de itt úgy nem kell az if mert csak egy eset van. A változtatesnyer vektort a nemváltoztatesnyerhez hasonlóan hozzuk létre, de ez azokat az indexeket tartalmazza ahol a jatekos újabbat választva kiválasztotta a nyeremenyt. Úgye `==` operátort a logikai ellenőrzésekre használjuk, jelen esetben a `which` függvény paramereiben. Az `sprintf` függvény kiírja az első parameterkent adott " " idézőjelek közötti szöveget és közé a % jel megfelelő elhelyezésevel a második parameterkent megadott értéket, a % jel után az i az egész számot jelenti. A length függvények pedig kiírják a megadott vektorok méretét. A `length(nemváltoztatesnyer)+length(valtoztatesnyer)` eredménye a kiserletek_szama lesz. Mivel a musorvezeto mindig kivesz egy nyereményt a háromból így vagy nemváltoztatással vagy változtatással mindenképp nyersz. Úgye pont ellentétesen kell választani hogy így vagy úgy nyerj. Tehát a nagyobb esély arra van hogy nyereményt válassz mert ekkor nyersz változtatással 2/3. Így arra pedig hogy nemváltoztatással nyerj 1/3 esély van mivel jót a nyereményt kell kiválasztanod ehhez. A musorvezetonek a választásosnál van szerepe mert elintézi hogy biztos hogy nyerj. Így az első választás előtt van eldöntve választassz vagy nem. Jatek közben döntve viszont 50-50%-nak tunhet a választás esélye.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás videó:

```
#include<iostream>
int main(){
int v;
std::cin>>v;
for(int i=0;i<v;++i)
std::cout<<"| ";
}
```

Tutoraltam.

A decimális számrendszer a tízes számrendszer amit napjainkban használunk tíz számjegyből áll melyek a 0,1,2,3,4,5,6,7,8,9. Az unáris számrendszer az egyes számrendszer ami egy számjegyből vagy valamilyen szimbólumból is állhat. Az átváltás tízesből egyes számrendszerbe úgy történik hogy annyi szimbólumot írunk amennyi a szám értéke. Az egyes számrendszerből alakult ki a tízes számrendszer mivel így egyszerűbben leírhatjuk a nagyobb számokat. A program C++ nyelvet használ így iostream header file használata szükséges mivel a szabványos be kimeneten dolgozunk. Az std::cin segítségével mentjük a beírt sztringet egy v egész típusú változóba, majd for ciklus segítségével mely addig hatja végre az tasítást míg a feltétel igaz, tehát az i egész típusú változó értéket nullától növeljük eggyel amíg kisebb az értéke mint a v változó értéke. Az utasításban pedig minden ciklusban kiírunk egy | szimbólumot a szabványos kimenetre az std::cout függvény segítségével. Turing géppel rámegyünk a bemenet utolsó számjegyére majd csökkentjük az értéket a számnak eggyel a megfelelő módon és az egyenlőség után írunk egy szimbólumot minden érték csökkenésnél míg a bemenet nem torlodik. A Turing gép a bemenetet feldolgozza, az adott állapotátmenet graf segítségével. Az állapotok között az utasítások végrehajtása szerint lépdel, egy utasításban megadjuk az aktuális bemeneti karaktert mire cserélje és merre lépjen.

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

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

Megoldás videó:

A környezetfüggő nyelvekben a kepesi szabályok alakja lehet $aAb \rightarrow agb$ ahol A nemterminalis, a -t és b -t terminalisok és nemterminalisok alkotják, g -t nemterminalisok és terminalisok alkotják de üres szó is lehet. A kepesi szabályok alakja lehet még $a \rightarrow b$ ahol a -t és b -t nemterminalisok és terminalisok alkotják és a -nak tartalmaznia kell legalább egy nemterminalist és b hossza legalább akkora mint a hossza. A kepesi szabályok alakja lehet még $S \rightarrow a$ ahol S kezdő nemterminalis és a üres szó, ekkor S egyetlen szabály jobboldalon sem szerepelhet. Az a^n az n -edik a , b^n az n -edik b , c^n az n -edik c az $a^n b^n c^n$ azt jelenti hogy abc sorrendben vannak az a, b, c nemterminalisok azonos hatványon, ahányadikon van annyi nemterminalist kell írni, nulladik esetben üres szó lesz.

Nemterminalisok:

S, X, Y

Terminalisok:

a, b, c

Helyettesítési szabályok:

$S \rightarrow \text{uresszo}$

$S \rightarrow abc$

$S \rightarrow aXbc$

$Xb \rightarrow bX$

$Xc \rightarrow Ybcc$

$bY \rightarrow Yb$

$aY \rightarrow aaX$

$aY \rightarrow aa$

Nemterminalisok:

A, B, C

Terminalisok:

a, b, c

Helyettesítési szabályok:

$S \rightarrow A$

$S \rightarrow \text{uresszo}$

$A \rightarrow aAB$

$A \rightarrow aC$

$CB \rightarrow bCc$

$cB \rightarrow Bc$

$C \rightarrow bc$

3.3. Hivatkozási nyelv

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

A BNF nyelvet környezet-független nyelvek leírására használjuk, melyben az alábbi jelöléseket használjuk: \Rightarrow származtatási szabály megadása, kisebb nagyobb kacsacsor szimbólum, $|$ vagy, " " ` ` szöveg. A $\{ \} ?$ közt található kifejezés opcionális tehát az is lehetséges hogy nem létezik. A kódban a változók szimbólumként szerepelnek a megfelelő jelöléssel, mivel egy szimbólumnak egy értéke lehet egyszerre így a vagy jelöléssel definiáljuk $::=$ -vel. A `//` kommentelő jel használata c90-ben hibauzenetet ad vissza, c99-ben pedig működik, ahogy a `for` ciklus sem működik inicializált paraméterekkel c90-ben, csak c99-ben.

```
<Utasitasok> ::= <cimkezett_utasitas>
                | <kifejezesutasitas>
                | <osszetett_utasitas>
                | <kivalaszto_utasitas>
                | <iteracios_utasitas>
                | <vezerlesatado_utasitas>

<cimkezett_utasitas> ::= <azonosito> : <utasitas>
                        | case <allando_kifejezes> : <utasitas>
                        | default : <utasitas>

<kifejezesutasitas> ::= {<kifejezes>}? ;

<osszetett_utasitas> ::= { {<deklaracios_lista>}? {<utasitas_lista>}? }
<deklaracios_lista> ::= <deklaracio> | <deklaracios_lista> <deklaracio>
<utasitas_lista> ::= <utasitas> | <utasitas_lista> <utasitas>

<kivalaszto_utasitas> ::= if ( <kifejezes> ) <utasitas>
                        | if ( <kifejezes> ) <utasitas> else <utasitas>
                        | switch ( <kifejezes> ) <utasitas>

<iteracios_utasitas> ::= while ( <kifejezes> ) <utasitas>
                        | do <utasitas> while ( <kifejezes> ) ;
                        | for ( {<kifejezes>}? ; {<kifejezes>}? ; {<kifejezes>}? ) <utasitas>

<vezerlesatado_utasitas> ::= goto <azonosito> ;
                        | continue ;
                        | break ;
                        | return {<kifejezes>}? ;
```

```
int main()
{
    //a
```

```
for(int i=0; i<10; ++i) {  
}  
}
```

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.4. Saját lexikális elemző

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

```
%{  
#include <stdio.h>  
int realnumbers = 0;  
%}  
%%  
[0-9]*(\.[0-9]+)? {++realnumbers;  
printf("[Valos szam=%s]", yytext);}  
%%  
int main ()  
{  
yylex ();  
printf("Valos szamok szama =%d\n", realnumbers);  
return 0;  
}
```

A program használatához telepíteni kell a flex csomagot, és fordításhoz a lex kulcsszót és a -lfl kapcsolót kell használni. A program első részében C programhoz hasonlóan definiáljuk a használatos változókat és includoljuk a fájlokat, majd a %% jelek közötti részen reguláris kifejezésekkel adjuk meg a szabályokat amelyekre illeszkedő karaktereket keres a program. A [0-9] rész egy nulla és kilenc közötti egész számot jelent, a * karakter használata azt jelenti hogy az előtte lévő kifejezés nulla vagy többször is szerepelhet egymás után, a + karakter pedig hogy egy vagy többször szerepelhet az előtte lévő kifejezés, a \. a tizedes vesszőt jelenti mivel valós számra való illeszkedést keresünk az nem feltétlenül egész szám lesz. A ? karakter azt jelenti hogy az előtte lévő kifejezés nulla vagy egyszer szerepelhet, mivel egész számok is valós számok és egy tizedesrészük van, úgynevezett jelen esetben a ? hatasköre az előtte lévő zárójelek közötti összes kifejezés. Majd egy másik reguláris kifejezéssel számoljuk a realnumbers változó értékében hány valós számot találtunk és kiíratjuk a printf függvény segítségével a talált számot, az yytext változó értéke a talált kifejezés értéke ami jelen esetben a valós szám. Az int main részen meghívjuk az yylex függvényt mely futtatja a szabályokkal a lexikális elemzést majd miután végigment a szövegen kiíratjuk a valós számok számát. A program a szabályokra nem illeszkedő kifejezéseket is megjeleníti a kimeneti csatornán ami alapbeállítás alapján a szabványos kimenet, illetve a szabályok által módosított kifejezéseket is a szövegben.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

A felx program elso reszeben azaz a `%{ }%` reszben definialjuk a program mukodeséhez szukseges valtozokat, strukturakat es include-oljuk a megfelelo header file-okat. Definialunk egy `L337SIZE` macrot mely az `l33d1c7` tomb sorainak szamat, mivel a `sizeof` fuggveny meghatarozza az adott tomb, struktura meretet bajtban, mivel `char` tipusu igy egy bajt. A strukturaban mindegyik valtozo erteke `char` tipusu igy az `l337d1c7` tomb merete annyi ahany eleme van, a chiper struktura merete pedig a dekalalt valtozok merete ami egy sora az `l337d1c7` tombnek igy a hanyados erteke a tomb oszlopainak a szama lesz. Majd a chiper struktura tipusu tombnek az ertekeit adjuk meg mellyen definialjuk hogy az adott karaktert majd milyen karakterekre cserelhetjuk le. A program masodik reszeben azaz `%% %%` reszben kifejezesek hasznalataval adjuk meg a flexeles szabalyait. A `.` karakter egy tetszoleges karakter a bemenetrol kiveve az ujsort melyet keresunk a szovegben, majd egy kifejezesben megadjuk hogy a illeszkedes eseten melyik karakterre cserelje at a beolvasott karaktert. Ehhez inicializaljuk a `found` valtozot nulla kezdoertekkel, melynek erteke illeszkedes eseten egyre valtozik es mivel minden karakter megfelel a szabalynak mivel minden egy tetszoleges karaktert keresunk igy nekunk kell kiiradni ha nem modsitjuk az erteket, mivel automatikusan csak a szabalynak nem megfelelot irna ki. A `for` ciklus segitsegevel minden sorban talalhato karakterre azaz a tomb `c` konstans reszere ellenorizzuk hogy illeszkedik e a beolvasott karakter ami az `yytext` valtozo tarol, mivel kisbetus karaktereket adtunk meg es a karakter nagybetus megfelelojet is ugyanugy kezeljuk igy a `tolower` fuggvennyel kisbetuskent ellenorizzuk az illeszkedest. Majd ha teljesul a feltetel, tehat talaltunk egy illeszkedo karaktert akkor `r` egesz tipusu valtozoba generalunk egy random egesz szamot egy es szaz kozott, mivel a `rand()/RAND_MAX+1.0` egy nulla es egy kozotti lebegopontos szamot general, ugye a `+1.0` azert szukseges hogy ne egeszeket osszunk mert akkor mindig nullat kapnank, de mivel ezt megszoroztuk szazzal igy nulla es szaz kozti szamot kapunk szazat a nullat beleertve mivel egesz tipusura kerekitjuk az osztas utan es csak szazzal szoroztuk elotte a valos szamot. Majd hozzaadunk meg egyet a kapott hanyadoshoz hogy nagyobb szamot kapjunk mivel a felteteleket kilencven feletti szamokra is adtuk, igy a szam mar egy es szaz kozotti lesz. Majd `if` feltelakkal meghatarozzuk a generalt szam alapjan melyik karaktert irjuk ki a kimenetre a beolvasott helyett az `l33d1c7` tomb adott sorahoz tartozo leet valtozo megfelelo ertekevel. Majd a `found` valtozo erteket egyre allitjuk igy nem irjuk ki a mar kiirt karaktert megegyszer, majd a `break` fuggvennyel kilepunk a `for` ciklusbol. Majd egy `if` feltetel segitsegevel ellenorizzuk hogy a `found` valtozo erteke nulla e mert akkor kiiradjuk a beolvasott karaktert, mivel mindegyik karakter megfelel a szabalynak igy nem ir ki egyet sem automatikusan. A `main` fuggvenyben az `scrand` segitsegevel melynek segitsegevel kezdoerteket adunk a `rand` fuggvenynek ami ugye utanna hivodik meg az `yylex` fuggvenyben, a `time(NULL)` fuggveny a program futasa ota eltelt miliszkundumokat adja meg, de mivel igy minden ujra inditaskor nagyabol akkor hivodna meg igy hozzaadjuk az aktualis `processID`-t azaz muveletazonositot ami a `getpid` fuggveny visszateresi erteke, igy nem inicializaljuk ugyanolyan ertekkel a `rand` fuggveny pszedo listajat ahonnan a szamokat generalja az automatikusan generalt `RAND_MAX` valtozo ertekeig. Mivel van visszateresi erteke `main`-nek igy addig hivodik ujra az `yylex` fuggveny amig van bemenet.

```
% {
```



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

#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))

struct cipher {
    char c;
    char *leet[4];
} l337d1c7 [] = {

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

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

```
%}
%%
. {
    int found = 0;
    for(int i=0; i<L337SIZE; ++i)
    {
        if(l337d1c7[i].c == tolower(*yytext))
        {
            int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));
            if(r<91)
                printf("%s", l337d1c7[i].leet[0]);
            else if(r<95)
                printf("%s", l337d1c7[i].leet[1]);
            else if(r<98)
                printf("%s", l337d1c7[i].leet[2]);
            else
                printf("%s", l337d1c7[i].leet[3]);

            found = 1;
            break;
        }
    }
    if(!found)
        printf("%c", *yytext);
}
%%
int
main()
{
    srand(time(NULL)+getpid());
    yylex();
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

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

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

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

**Bugok**

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

i.

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

Ha az elozo signal fuggvényben nem ignoroltuk a billentyuzetrol bemeno utasitasok értelmezését, akkor ezeket a jelkezeslo fuggvény fogja kezelni. Akkor ignoráljuk a signal fuggvény első paramétereként adott szignált, ha az elozo signal fuggvény második paramétere SIG_IGN volt, és a signal fuggvény visszatérési értéke a második paraméterként megadott szignálkezeslo fuggvény elozo értéke.

ii.

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

Egy for ciklus mely végrehajtja az adott utasításokat ötször, a ++i kifejezés először növeli az i változó értékét, aztán állítja be a visszatérési értéket így külön változóban mentve az utasítást növelt értéket kapnánk.

iii.

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

Egy for ciklus mely végrehajtja az adott utasításokat ötször, az i++ kifejezés először megadja a visszatérési értéket, aztán növeli a másik változó értékét így külön változóba mentve az utasítás a növelni kívánt változó eredeti értékét kapnánk vissza, de az érték növelt lenne.

iv.

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

A for ciklus nem jól fogja kitölteni az értékeket, tehát nem 0,1,2,3,4 értékek kerülnek a tombba, hanem az első helyre random érték kerül így tomb[i-1]-et kell használnunk, illetve ++i-t.

v.

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

A for ciklus annyiszor fut amíg i értéke nem lesz egyenlő az n változó értékével avgy a *d++=*s++ egyenlőség nem áll fenn.

vi.

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

A printf függvényel kiírtuk az f függvény paraméterei által visszatartott értéket.

vii.

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

A printf függvényel kiíratjuk az f függvény visszatérési értékét a változóra és az a változó értékét.

viii.

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

A printf függvényel kiíratjuk az f függvény visszatérési értékét az a változó memóriacímére és az a változó értékét.

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat... Az első formulának a jelentése hogy a prímszámok száma végtelen, a második formulája hogy az ikerprímek száma végtelen, a harmadik formulája hogy a prímszámok száma véges, a negyedik formulája hogy a prímszámok száma végtelen.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

```
int a;
```

Az a-t egész típusú változóként deklaráltuk.

- ```
int *b = &a;
```

A b egész típusu mutató változót inicializáljuk az a változó memóriacímevel.

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

Az r egész típusu változó memóriacímenek értékül adjuk az a változót.

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

Létrehozunk egy öt egész típusú elemből álló tömböt.

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

A tr egész típusú tömb ötödik elemének memóriacímet a c változó memóriacímeire állítjuk.

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

Létrehozunk öt egész típusú memóriacímrre mutató tömböt d néven.

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

Definiálunk egy h névű függvényt melynek visszatérési értéke egész típusú mutató.

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

Definiálunk egy l névű függvényre mutató mutató melynek visszatérési értéke egész típusú mutató.

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

Definiálunk egy v névű függvényre mutató mutatót melynek visszatérési értéke egész típusú, parametere egy c névű függvény melynek visszatérési értéke egész típusú és parametere a és b egész típusú változók.

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

Definiálunk egy z névű függvényre mutató mutatót melynek parametere egész típusú és mutatójának visszatérési értéke egész típusú és parametere két egész típusú.

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

## 4. fejezet

# Helló, Caesar!

### 4.1. int \*\*\* háromszögmátrix

Írj egy olyan malloc és free párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tarban!

```
#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;
}
```

Eloszor lefoglaljuk a helyet az alsóharomszög matrixnak majd feltöltjük adatokkal aztán kiírjuk és töröljük a lefoglalt helyet. A matrix minden értékének double a mertet foglalunk le, sorainak double\*-t és az oszlokoknak double\*\*-t. A double\*\* pointer által mutatott memóriaterületre mentünk csak pontosan. Le-trehozzuk az nr változót egész típusúval és 5 értékkel. A tm változót deklaráljuk mutatóként ami egy double típusra mutat. A double \*\*tm is foglal le tarhelyet. A tm pointer típusváltozóra lefoglal a malloc függvény nr-szer ami úgynevezett most 5 sizeof(double\*)-nyi helyet ami a double\* mertet jelenti. A malloc az operációs rendszertől kér memóriát eredetileg void\* mutatóval de itt double\*\* a mutatója a double\*-nak így típus-kenyerítjük a zárójelbe. Ha nem lenne tarhely akkor NULL értéket kap a tm és kilep a program. A sorok oszlopainak for ciklussal foglalunk helyet. Mivel alsóharomszög matrix így a 0-ik sorba 1 double mértéki azaz 8 byte ezen az operációs rendszeren kellessz. Mindegyik sorba egyel több és az utolsóba mi a 4.-ik (1+4)\*8 azaz 40 byte kellessz. Ha elfogy a tarhely akkor itt is NULL értéket kap és kilep a program, -1-et kap a main függvény. Itt minden double-hoz double\* pointer tartozik. Majd 2 egymásba ágyazott for cik-lussal feltöltjük a lefoglalt tarhelyeket elemekkel. Az értékeket soronként adjuk meg a keplet szerint. Majd kiírjuk az értékeket a printf függvény segítségével szokásos és enter megfelelő használatával. A tm[3][0] = 42.0; a 3. sor 0. elemét írja át. A (\*(tm + 3))[1] = 43.0; a 3. sor 1. oszlopának elemét. A \*(tm[3] + 2) = 44.0; a 3. sor 2. oszlopát írja át. A \*(\*(tm + 3) + 3) = 45.0; a 3. sor 3. oszlopát írja át. A \*(tm + 3)[1] = 43.0; a 4. sor 0. oszlopát írja át. A free függvény felszabadítja a malloc által lefoglalt memóriát a tm és tm[i]-kre vonatkozóan.

Megoldás videó:

Megoldás forrása:

Tanulások, tapasztalatok, magyarázat...

## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

int
main (int argc, char **argv)
{
 char kulcs[MAX_KULCS];
 char buffer[BUFFER_MERET];

 int kulcs_index = 0;
 int olvasott_bajtok = 0;

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

 while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
 {
 for (int i = 0; i < olvasott_bajtok; ++i)
 {
 buffer[i] = buffer[i] ^ kulcs[kulcs_index];
 kulcs_index = (kulcs_index + 1) % kulcs_meret;
 }

 write (1, buffer, olvasott_bajtok);
 }
}
```

A `#define` szóval definiáljuk a konstansokat C-ben nevükkel majd értékükkel, ezeknek az értéke a program futása alatt nem változik meg és az egész fájlba érvényesek az értékei. A `main` függvény `argc` parametere a program futtatását tartalmazó parancssori argumentumok számát tartalmazó változó beleértve a program nevet, az `argv` a parancssori argumentumok szavait tartalmazó vektor. C-ben a vektor indexei 0-tól indulnak. A `char` szóval deklaráljuk a karakter típusú változókat. Jelen esetben létrehoztuk a `kulcs` és `buffer` nevű változót 100 és 256 elemmel. C-ben változók létrehozásakor a `nev[x]` eseten `x` a méret, de hivatkozás



nev[x-1]-ig megy nev[0]-tól. Az int szót az egész szám értéke változók deklarálására használjuk. Inicializáltuk 0 értékre a kulcs\_index és olvasott\_bajtok változókat. A kulcs\_meret változó értéket a futtatáskor megadott kulcs hossza. Ezt az strlen függvény segítségével határozzuk meg ami a parameterül adott sztring hosszát adja meg. A sztring az egy parancssori argumentum. Jelen esetben a meghatározando sztring az arg[1] lesz ami a megadott kulcs. Az strncpy(x,y,z) függvény az y által mutatott parancssori argumentumot átmásolja x-be maximum n bajtot. Mukodesukhoz szükséges a string.h header file amit include-olnunk kell. A while ciklus addig fut amíg a feltétel hamis, amíg van olvasható bajt. A read függvény első paramtere az hogy honan olvas, jelen esetben 0 tehát a szabványos bemenetrol olvas. A második parameter hogy hol tárolja a beolvasott bajtokat, jelen esetben a buffer vektorba amit void\*-al típuskenyszerítettünk. A harmadik parameter pedig az hogy hány bajtot olvasson be. A visszatérési értéke a beolvasott bajtok száma, 0 ha nincs beolvasando bajt. A program addig fut amíg a while ciklus fut, az pedig amíg tud beolvasni legalább 1 bajtot a read. Mivel a while feltételben duplán van zerojelezve a tartalom így nem logikai vizsgálatot végez a feltételben, hanem az olvasott\_bajtok változó értéket modisítja a read. A while ciklus addig fut végtelesen ciklusként amíg 0-nál nagyobb egészeket kap vissza, de minden ciklus után a read olvas így változik a feltétel értéke, majd 0-t elér ha nincs úgye olvasando bajt, a while(0) pedig kilep a ciklusból. Mivel 1 bajt 1 karakter mert 8 biten tarunk egy karaktert ezen a rendszeren ezért 256 bajt olvasa 256 karaktert jelent. A char vektor pedig bajtonként tárolja az adatokat, 1 bajt 1 elem a vektorban. A for ciklusban bajtonket olvasunk be sorba a buffer vektor tartalmát és egyenként exorozzuk. Az exor művelet jele ^ operator. Ami 0^0 és 1^1 esetén 0-t, 1^0 és 0^1 esetén 1-t ad eredményül. Így ha mivel fix a kulcs így újból exorozva visszakapjuk az eredeti eredményt, mert minden egyfelekeppen jöhet ki. Így minden karaktert 8 biten exoroz egy 8 bites kulccsal és 8 bites eredményt eredményül kapja az adott indexű buffer vektor. A % operator kiírja a maradékos osztás maradékát ami egész szám, ha nincs meg benne akkor az eredeti számot, ha pont annyiszor van meg akkor 0-t. Úgye szükséges ezzel a módszerrel kinullazni a kulcs\_index-et mivel a kulcs\_meret általában kisebb lesz mint az olvasott\_bajtok mérete. A write függvény parameterit ugyanúgy kell megadni mint a read-t, itt 1-et adunk meg az első paramternek így a szabványos kimenetre ír. A read használatához szükséges az unistd.h header és az stdio.h header pedig mivel a bemeneten és kimeneten dolgozunk. A read függvény második paramtere void típusú pointer kell legyen így a buffert void\*-al típuskenyszerítjük. A buffer a buffer[] sor 0. elemének pointere, tehát a memóriacímét tárolja. A sor elemei sorba vannak így a read ír sorba elemenként egybajtot mert a sor char típusú. Az olvasott\_bajtok változót nemszükséges inicializálni, mert a read-nek mindnig van visszatérési értéke és nem használjuk a while cikluson kívül. A while ciklus azért szükséges hogy tudjuk hány bajtot olvastunk be ami az exor művelet elvégzéséhez szükséges, ezt meglehetősen határozni a sor méreteből, de nagyobb szövegeknel lassabb és nembiztos elég a memória. Viszont nem tudjuk mennyi bajtot olvasunk be és mivel a sor elemeit túl lépve errorrt kapunk ezért ciklust kell használnunk kiveve ha nagy méretű sort használunk de úgy nem biztos működésu.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

```
public class ExorTitkosító {
```

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

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

 while((olvasottBájtok =
 bejövőCsatorna.read(buffer)) != -1) {

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

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

 }

 kimenőCsatorna.write(buffer, 0, olvasottBájtok);

 }

}

public static void main(String[] args) {

 try {

 new ExorTitkosító(args[0], System.in, System.out);

 } catch(java.io.IOException e) {

 e.printStackTrace();

 }

}

}
```

Java program minden sora egy osztályban van és tartalmazni kell a programnak a main függvényt. Az osztálynevek nagykezdobetűvel kezdődnek és a fájl neve az osztály neve, fájlkiterjesztése .java. Javac paranccsal general egy .class kiterjesztésű fájlt amit a névvel futtathatunk a java parancs segítségével ekkor nem kell a név után a kiterjesztést írni. A main függvény pontosabban a public static void main(String[] args) kód. A hibakezelésre használjuk az try és catch szavakat mivel hiba esetén kilep a program. A try-ba megadott részek tesztelődnek a program futása közben és hiba esetén a catch-be megadott kód lefut. Jelen

esetben a try tartalma az ExorTitkosito nevu konstruktor ami szinte maga a program es a catch a bemeneti-kimeneti hibakat kezeli. A throws kifejezessel jelezzuk hogy milyen tipusu hibát dob a konstruktor tipusu fuggvény ezt a fuggvény parameterei es utasításai köze kell deklarálni. A konstruktor nevének meg kell egyeznie az osztály nevével es a main fuggvénybe való hívása a new kifejezéssel történik. Hivaskor elso parameternek egy sztringnek kell lenni mivel ugy deklaráltuk ez jelen esetben a parancssorba irt elso sztring a java ExorTitkosito utan amit az args[0] parancs erteke. A masodik paramter az hogy honnan olvassuk az adatokat a System.in a szabványos bemenet. A harmadik paramter a System.out ami a szabványos kimenet ahova ír a fuggvény. A read fuggvényt a java.io.InputStream elotaggal objektummal használjuk a write fuggvényt a java.io.OutputStream-mel amit atneveztünk bejovoCsatorna-ra es kimenoCsatorna-ra az ExorTitkosito fuggvény deklarációjánál. Deklaráljuk a kulcs nevu sort byte típussal azaz minden eleme 1 bajt lesz erteke a parancssorba megadott kulcs lesz amit a getBytes fuggvény segítségével mentünk a kulcs sorba. A getBytes fuggvény a sztringet kódolja egy bajtsorozatba majd visszaad egy bajtsort, a sztringet az elotte levo rész határozza meg. A buffer nevu sort létrehozzuk byte típussal es értéket adunk egy új byte nevu sort 256 elemmel tehát 256 bajtnyi lesz a sor es a buffer sor mutatóját rá fog mutatni. A kulcsIndex es az olvasottBajtok változókat deklaráljuk egészenk es inicializáljuk 0 értékekkel. A while ciklus fut végtelen ciklusként amíg van beolvasando bajt mivel a read fuggvény visszatérési értéke a beolvasott bajtok száma majd -1 ha már nincs több beolvasando bajt. Tehát a beolvasas bajtonként megy no az olvasottBajtok változó értéke amit felhasználunk az exor művelet elvégzéséhez es a kiíratáshoz. Beolvas egy bajtot exorozza a kulcs megfelelő bajtjával majd kiírja a szabványos kimenetre majd addig fut újból míg van bemenet. A write fuggvény elso parameter hogy hova írjon a masodik parameter hogy mennyi a kezdeti eltolás mértéke a harmadik paramter az hogy hany bajtot írjon ki.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 4.4. C EXOR törő

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

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

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

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

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

int
tisztalta_lehet (const char *titkos, int titkos_meret)
{
 double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

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

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

int
xor_tores (const char kulcs[], int kulcs_meret, char titkos[],
 int titkos_meret)
{
 xor (kulcs, kulcs_meret, titkos, titkos_meret);

 return tisztalta_lehet (titkos, titkos_meret);
}

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

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

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

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

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

 exor (kulcs, KULCS_MERET, titkos, p - titkos);
 }

return 0;
}
```

Az int main függvényt paraméter nélkül deklaráltuk úgy, hogy void-al és definiáltunk konstansokat. Deklaráltuk a kulcs és titkos vektorokat 4096 és 8 méretűnek karakter típusúval, így annyi bajtosak is. A p változót deklaráljuk pointerként, ami a memóriába egy char típusra mutat és inicializáljuk titkos sorneve értékével, ami a sor 0. elemének memóriacímét tárolja. Majd deklaráljuk az olvasott\_bajtok változót. A while ciklust {} nélkül használhatjuk, ha csak a feltétel után lévő egy állítást tartalmazza a ciklus, úgy, jelelve; vagy {}-el jelölve. Ez érvényes a for, if esetén is. Úgy, hogy a read függvény a szabványos bemenetről olvas bajtonként és menti void típusú pointer címére, jelen esetben a p változó címére, amit void\*-al típuskénszerítettünk. Mivel char típusú sorba menti az adatokat, így egy bit egy elem. A while ciklus addig fut, végtelen ciklusként, amíg van beolvasandó bajt, vagy el nem érjük a 4096 bajtot, a felett, mivel nincs több

hely a titkos sorba így kilep error-ral a program. Az olvasott\_bajtok számával növeljük a p pointer típusu változoba mentett memóriacím méretét minden ciklus végrehajtásakor. Pontosan annyival növeli ahány az olvasott\_bajtok aktuális értéke. A read harmadik paramétere ahova a beolvasandó bajtokszámát kell megadni tartalmaz egy feltételes kifejezést jelölés feltétel  $? x : y$ . Ha a teljesül a feltétel akkor a kifejezés értéke x azaz a : előtti rész, ha nem teljesül akkor a kifejezés értéke y azaz a : utáni rész. Így jelen esetben a read harmadik paramétere vagy OLVAS\_BUFFER értéke amit definiáltunk 256-nak vagy titkos ami a titkos sor első vektorának memóriacímébe hozzáadva MAX\_TITKOS értéket amit definiáltunk 4096-nak kivonva a p aktuális értéket. A p memóriacím kezdőértéke megegyezik a titkos memóriacímével hisz mindkettő a titkos sor 0. elemére mutatnak. A feltétel ellenőrzi kisebb-e beolvasott bajtok száma azaz beírt vektor elemeinek a száma amit a p - titkos határoz meg és hozzáadva az OLVASAS\_BUFFER értéket kisebb-e a MAX\_TITKOS értékenél mivel annyi eleme a sornak amibe ír a read függvény. Ugye 4096 az  $16 \cdot 256$  így ha 4096 a beolvasandó szöveg akkor a  $? x : y$  feltételes kifejezés x értéke 15-szor majd az y értéke 1-szer fut le. Ha pedig kevesebb lesz mint 256 valahányszorosa ami kisebb mint 16 akkor beolvassa amennyi van visszatérve értéke az olvasott\_bajtok-nak és növeli a p-t majd próbál +256 bajtot olvasni de 0 bajtot olvas így 0 eredményt ad vissza és kilep a while ciklus. A for ciklus ha tele van a titkos sor akkor hamis feltétellel kilep egyébként a sor üres elemeit töltjük fel \0 értékekkel. Eredetileg a titkos sor azon elemei ahova nem olvastunk be adatot random számokat tartalmaznak szinte minden hivatkozáskor különbözőt. Az egymásba ágyazott for ciklusok megadnak minden kombinációt 8 számjegyre titkosítási kulcs megfejtésére úgy ezzel volt exorozva az kódolatlan szöveg. Eloszor 00000000 kulcs lesz majd 0000001 majd 1-9 között aztán egyelőttel levő számjeggyel. 0-9-ig egyenként az utolsó 0-9-ig előtte levők 0-k és így tovább. Addig generalja a kulcsokat amíg az if feltétel igaz nem lesz és kiírja a kulcsot a szabványos kimenetre. Az if feltétel tartalmazza az exor\_tores függvényt ami egész számot ad vissza ha 1-et akkor teljesül az if feltétel ha 0-t akkor az if utasítás része nem fut le. Ezt a tiszta\_lehet függvény határozza meg az atlagos\_szohossz függvény segítségével. Az atlagos\_szohossz függvénnyel megszámláljuk hány szóköz van a beolvasott szövegben és leosztjuk az összes karakterek számával, mivel ez nem biztos egész szám így double visszatérési értékű a függvény és kiszámításakor a műveletet típuskonverziót adunk double-re. Ez az érték lesz a szohossz változó értéke a tiszta\_lehet függvényben aminek a visszatérési értéket egy összetett logikai feltétel határozza meg. A logikai feltétel logikai és operator ami akkor igaz ha minden komponense nem 0, egyébként 0. A feltételben ellenőrizzük a szohossz 6.0-nál nagyobb és 9.0-nál kisebb és tartalmazza-e a gyakori magyar szavakat a "hogy" "nem" "az" "ha" szavak reszszóként szerepelnek a beolvasott szövegben. Ha megtalálja a keresett szót akkor visszatérve annak első betűjének első előfordulási helyének pointerét, ha pedig nincs a szövegben akkor 0-t ad vissza. Az exor függvény amit az exor\_tores függvény hív meg a titkos sort bajtonként exorozza a kulcs sor bajtjaival sorba 0-tól 7-ig majd ismételve ameddig a titkos sor tart. A const szóval konstans változókat hoztunk létre aminek ha változtatnánk az értéket hibajelzést kapnánk. A strcasestr függvény használatához szükséges a string.h header file és a \_GNU\_SOURCE-t definiálni. Definálni az include-előtt kell. A program minden kulcsot legeneral de csak a feltételnek megfelelőeket írja ki a printf függvény segítségével, minden lépésben 2-szer exorozunk így a titkos sor értékei a beolvasott marad.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 4.5. Neurális OR, AND és EXOR kapu

```
library(neuralnet)
```

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

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

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

plot(nn.or)

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

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

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

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

plot(nn.orand)

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

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

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

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

plot(nn.exor)

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

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



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

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. <-
 output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

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

## R

A neuralis halokat nem programozzuk hanem tanítjuk. Az első sorba belepünk a neuralnet könyvtárba. Majd létrehozuk a c vektorokat és elnevezzük a1, a2, OR-nak hogy tudjunk rájuk hivatkozni. Ezzel deklaráltuk az a1, a2, OR változókat a megfelelő értékekkel. 0 és 1 az igaz hamis OR, AND, EXOR pedig a logikai műveletek amelyeket beakarunk tanítani. Ugye az OR azaz a megengedő vagy csak hamis hamis értékeknel ad vissza hamisat egyébként igazat. Az AND azaz az és csak igaz és igaz értékeknel ad vissza igaz értéket egyébként hamis. Az EXOR azaz a kizáró vagy pedig hamis és hamis értékeknel hamis, igaz és igaz értékeknel hamis, igaz hamis és hamis igaz értékeknel pedig igaz. Az exor.data névvel létrehozunk egy matrixot az adott a1, a2, OR vektorokból. Majd nn.or néven létrehozuk a beépített neuralnet függvényt amely a megfelelően megadott paraméterekkel kiszámolja a szükséges súlyokat egy aktivációs függvény segítségével és a hibáról is kapunk információt. A súlyok mellett van egy eltolássúly is aminek ugye akkor van jelentősége ha a bemeneti neuronok értékei 0-k, mert akkor az eltolóssúly értékeiben lesz majd nem 0 a függvény. Ezzel az eltolóssúllyal nemcsak 0 körüli zérushelyű függvényeket használhatunk. Az aktivációs függvény a súlyozott összegekkel számol és visszaad egy értéket 0 és 1 között. Jelen esetben az aktivációs függvény  $1/(1+\exp(-x))$  de más függvényt is használhatunk például a hiperbolikus tangens. A veszteségfüggvény kiszámolja a hibát az eredeti kimenet és a várt kimenet ismertetében. Jelen esetben az  $1/2*(y-x)^2$  ami az átlagos négyzetes eltérésként számolja ki a hibát. A neuralnet első paraméterében megadunk egy formulát aminek R-ben az elemeket a ~ operátorral választjuk el, bal oldalra kell az eredményt írni, jobbra amikből kijön majd. Az egyes oldalakon a változóneveket + operátorral kapcsoljuk össze ahogy az OR+AND~a1+a2. A ~ operátor baloldalon a bemenő neuronok vannak, a jobb oldalon a kimenet. A második paraméterként az adatot amivel dolgozik a függvény, ami jelen esetben egy matrix. A hidden értékeként megadhatjuk hány rejtett neuron legyen, alapértéke 1. Az EXOR-nal többretegu neuronokkal kell számolni, mert rejtett neuronok nélkül nagy hibával számol. A hidden értéket megadhatjuk vektorral is ekkor ahány elemű annyi sor és az értékei az oszlopok száma soronként így kisebb hibát kaphatunk ahogy a c(6,4,6) vektorral is kaptunk. A logical.output-nak logikai értéket kell adni TRUE vagy FALSE-t. TRUE az alapértelmezett érték ekkor az aktivációs függvényt nem alkalmazza a kimenő neuronokra. A stepmax-ba megadhatjuk hány lepest vezethet el maximum a neurális hálózat képzésekor. Ennek alapértéke  $1e+05$  azaz 100000. A threshold értéke meghatározza a részleges hiba küszöbértéket, amit ha átlép megáll a számítás. A plot függvénnyel kirajzoltatjuk az nn.exor-t. A compute első paramtere a függvény neve amellyel számoltuk a neurális hálót. A második paramtere tartalmazza azt a matrixot amelyből számoltuk a neurális hálót. Csak data.frame-ba tudunk létrehozni matrixot vektor nevekből, a többi általunk adott név tetszőleges. Ezzel ellenőrizhetjük a kapott súlyokat és az eredeti értékekkel számolva mennyi jön ki EXOR, AND, OR helyett, ez a ne.result matrixba van.

Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

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

Tanulságok, tapasztalatok, magyarázat...



## 4.6. Hiba-visszaterjesztéses perceptron

C++

```
#ifndef QL_HPP
#define QL_HPP

#include <iostream>
#include <cstdint>
#include <map>
#include <iterator>
#include <cmath>
#include <random>
#include <limits>
#include <fstream>

class Perceptron
{
public:

 Perceptron (int nof, ...)
 {
 n_layers = nof;

 units = new double*[n_layers];
 n_units = new int[n_layers];

 va_list vap;

 va_start (vap, nof);

 for (int i {0}; i < n_layers; ++i)
 {
 n_units[i] = va_arg (vap, int);

 if (i)
 units[i] = new double [n_units[i]];
 }

 va_end (vap);

 weights = new double**[n_layers-1];

#ifdef RND_DEBUG
 std::random_device init;
 std::default_random_engine gen {init() };
#else
 std::default_random_engine gen;
#endif
 }
};
```

```
std::uniform_real_distribution<double> dist (-1.0, 1.0);

for (int i {1}; i < n_layers; ++i)
{
 weights[i-1] = new double *[n_units[i]];

 for (int j {0}; j < n_units[i]; ++j)
 {
 weights[i-1][j] = new double [n_units[i-1]];

 for (int k {0}; k < n_units[i-1]; ++k)
 {
 weights[i-1][j][k] = dist (gen);
 }
 }
}

Perceptron (std::fstream & file)
{
 file >> n_layers;

 units = new double*[n_layers];
 n_units = new int[n_layers];

 for (int i {0}; i < n_layers; ++i)
 {
 file >> n_units[i];

 if (i)
 units[i] = new double [n_units[i]];
 }

 weights = new double**[n_layers-1];

 for (int i {1}; i < n_layers; ++i)
 {
 weights[i-1] = new double *[n_units[i]];

 for (int j {0}; j < n_units[i]; ++j)
 {
 weights[i-1][j] = new double [n_units[i-1]];

 for (int k {0}; k < n_units[i-1]; ++k)
 {
 file >> weights[i-1][j][k];
 }
 }
 }
}
```

```
}

double sigmoid (double x)
{
 return 1.0/ (1.0 + exp (-x));
}

double operator() (double image [])
{
 units[0] = image;

 for (int i {1}; i < n_layers; ++i)
 {
#ifdef CUDA_PRCPS

 cuda_layer (i, n_units, units, weights);

#else

#pragma omp parallel for
 for (int j = 0; j < n_units[i]; ++j)
 {
 units[i][j] = 0.0;

 for (int k = 0; k < n_units[i-1]; ++k)
 {
 units[i][j] += weights[i-1][j][k] * units[i-1][k];
 }

 units[i][j] = sigmoid (units[i][j]);
 }

#endif

 }

 return sigmoid (units[n_layers - 1][0]);
}

void learning (double image [], double q, double prev_q)
{
 double y[1] {q};

 learning (image, y);
}
```



```
 }
 }
}

for (int i {0}; i < n_layers-1; ++i)
{
 delete [] backs[i];
}

delete [] backs;
}

~Perceptron()
{
 for (int i {1}; i < n_layers; ++i)
 {
 for (int j {0}; j < n_units[i]; ++j)
 {
 delete [] weights[i-1][j];
 }

 delete [] weights[i-1];
 }

 delete [] weights;

 for (int i {0}; i < n_layers; ++i)
 {
 if (i)
 delete [] units[i];
 }

 delete [] units;
 delete [] n_units;
}

void save (std::fstream & out)
{
 out << " "
 << n_layers;

 for (int i {0}; i < n_layers; ++i)
 out << " " << n_units[i];

 for (int i {1}; i < n_layers; ++i)
 {
 for (int j {0}; j < n_units[i]; ++j)
 {
```

```
 for (int k {0}; k < n_units[i-1]; ++k)
 {
 out << " "
 << weights[i-1][j][k];

 }
 }

}

private:
 Perceptron (const Perceptron &);
 Perceptron & operator= (const Perceptron &);

 int n_layers;
 int* n_units;
 double **units;
 double ***weights;

};
#endif
```

A program nem tartalmaz main függvényt így használatához másik fájlban irt main függvény és megfelelően megadott adatok függvény paraméterek szükségesek. A program tartalma a Perceptron osztály amiben definiálunk függvényeket és változókat. Egy objektumnak van private és public része is. Eloszor a private rész fut le, ehhez nincs kulso hozzáférés. A private részben létrehozunk egy masolo konstruktort és egy segéd konstruktort majd deklaráltunk egy változót és 3 pointert. Az egy \*-os pointert 1 dimenziós tömbökhöz azaz sorokhoz használjuk, a \*\*-os pointert 2 dimenziós tömbökhöz azaz matrixokhoz, a \*\*\*-os pointert 3 dimenziós tömbökhöz használjuk. Ezek a számításokhoz szükségesek. A masolo konstruktorral a masolt objektumnak és magának külön memóriát foglal, a segéd konstruktor ugyanazt használja, jelen esetben a masolo konstruktort. Mivel private részben definiáltak így nem használhatjuk őket más programba amivel használjuk ezt a kódot. A felesleges újradefinálás elkerülése érdekében ami ugye növelne a futási időt definiáltuk a QL\_HPP-t, ami újradefinálás esetén #endif-re lép és az #ifndef-nel ellenőrzi volt-e már definiálva. A Perceptron szerepel parametizált osztályként melynek értékeit objektum létrehozásakor adjuk meg, paraméterei eltérő mennyiségű egész típusú változók lehetnek attól függően hány rejtett neuront használunk. Mivel hiba-visszajelző preceptronként használjuk így minimum 4 paramétert kell megadnunk. Az első paraméter a rejtett neuronok száma, a második a bemeneti neuronok száma azaz adatok száma, a harmadik paraméter a rejtett neuronok száma, az utolsó paraméter a kimeneti neuronok számát adja meg. A variadikus függvény teszi lehetővé hogy eltérő paraméterű legyen a konstruktor. Használatához be kell foglalni a cstdarg.h header fájlt. A va\_list információt tárol a lista aktuális állásáról, nevet kell neki adni amit az va\_start, va\_arg és va\_end használ. A va\_start inicializálja a variable argument listát a legutolsó megadott konstruktor változóval, a konstruktor deklarálásánál legalább egy változót inicializálni kell és utána ... szükséges a függvény használatához. A va\_arg paraméterében megadjuk milyen típusokat inicializálnak. A va\_end a végét jelzi a variable arguments listának. Az n\_units soroké a va\_arg által visszatartott értékek, azaz majd egy külső objektum bizonyos paraméterei lesznek, mert megadjuk a konstruktor első paraméterével. Az units matrix 1.-től a konstruktor első paraméteré-1-ig indexű sorainak az oszlopainak double mennyiségeit, helyet fog-

lalunk. A `#ifndef` neve már volt definiálva, akkor az `#else` utáni rész fut az `#endif`-ig. Ha pedig még nem volt definiálva akkor az `#else`-ig lefut, az `#else` nem. A tartalma az hogy az `std::uniform_real_distribution` kisebbkacsacsor `double` nagyobbkacsacsor `dist ( -1.0, 1.0 )`; sor segítségével `-1.0` és `1.0` közötti számokat `random` `general double` típusban. Ennek segítségével adjuk meg a súlyok kezdőértékeit. A súlyokat és a retek számát és az egyes retek neuronjainak számát külső fájlból is beolvashatjuk. A `sigmoid` függvény az aktivációs függvény `double` értékekkel számol, ez a súlyozott értékek összegeiből kiszámol egy `0` és `1` közé eső számot. A `()` operátor túlterheléssel funktort hozunk létre így egy `Perceptron` objektumot típusként megadva számolhatunk az objektum és a funktor paramterének értékeivel és a számolt érték változások mentődnek és adhatunk visszatérési értéket is. A funktor paramtere egy `2` dimenziós tömb ami a tomorítendő kép pixel adatait varja paramterül. A pixel adatokat `3d` tömbként olvasva írjuk `2` dimenziós tömbbe sor vagy oszlopfolytonosan `for` ciklusok segítségével. A pixeladatokat az `image` `2` dimenziós tömb tartalmazza melynek a mutatója az `image` és ezt értékel adjuk az `units[0]` `2` dimenziós tömb `0.` sorának mutatójának így az `units[0]` az `image` értékeit használja számításokhoz. A `#pragma omp parallel for` parancsot arra használjuk hogy több szálon számoljon a processzor. Az `units` `2` dimenziós tömb `1.` sorától a `layers-1.` sorig számoljuk ki a súlyozott összegek aktivációs függvény értékeit. A funktor visszatérési értéke a tanításhoz szükséges, ez lesz a tenyleges érték. A vart értéket a `learning` függvény második paramterének adjuk meg, amit a `y` változóba ment. A `learning`-be van agyazva megegy `learning` függvény amivel meghatározzuk a megfelelő súlyokat hogy a vart értéket kapjuk. Ehhez hibakkal számolunk és bővítjük a tenyleges súlyokat. A hibát a `backs` `2` dimenziós tömb segítségével számoljuk ki ebbe számolunk az utána lévő `units` érték segítségével. Majd a súlyhoz hozzáadjuk a hozzátartozó `backs*konstans` érték\*következő `units` értéket és így megkapjuk a megfelelő súlyt. Az `1-sigmoid(units)`-nal a `simoid` nélkül `1` lehetne így kinullaznánk a súlyokat, de újra `sigmoid`-olva tart az érték csökken folyamatosan. Negatív érték esetén a második `sigmoid` az `0.5` lesz utána egyenletesen `no` ez az érték. A neuron hibáját a `2` felekeppen számoljuk ki, ha kimeneti neuron akkor a vart-tenyleges hibával számolnu, egyébként a megfelelő kiszámolt súly és az `units` és konstans szorzatot összeadva. A függvény végén töröljük a `backs` tömbnek foglalt helyeket a `delete` függvény segítségével, amit a `new`-val foglaltunk. A destruktork jele `~` és neve az osztály neve törli az objektumot és jelen esetben a foglalt memóriacímeket is, ez a program végén fut le. A `save` függvénnyel kimentethjük a retek számát, az egyes retek neuronjainak a számát és a súlyokat.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

```
#include <stdio.h>
#include <png.h>
#include <sys/times.h>
#include <stdlib.h>

#define MERET 600
#define ITER_HAT 32000

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

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

 // számítás adatai
 float a = -2.0, b = .7, c = -1.35, d = 1.35;
 int szelesseg = MERET, magassag = MERET, iteraciosHatar = 255;

 // a számítás
 float dx = (b - a) / szelesseg;
 float dy = (d - c) / magassag;
 float reC, imC, reZ, imZ, ujreZ, ujimZ;
 // Hány iterációt csináltunk?
 int iteracio = 0;
 // Végigzongorázzuk a szélesség x magasság rácsot:
 for (int j = 0; j < magassag; ++j)
 {
 //sor = j;
 for (int k = 0; k < szelesseg; ++k)
```



```
{
 // c = (reC, imC) a rács csomópontjainak
 // megfelelő komplex szám
 reC = a + k * dx;
 imC = d - j * dy;
 // z_0 = 0 = (reZ, imZ)
 reZ = 0;
 imZ = 0;
 iteracio = 0;
 // z_{n+1} = z_n * z_n + c iterációk
 // számítása, amíg |z_n| < 2 vagy még
 // nem értük el a 255 iterációt, ha
 // viszont elértük, akkor úgy vesszük,
 // hogy a kiindulási c komplex számra
 // az iteráció konvergens, azaz a c a
 // Mandelbrot halmaz eleme
 while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
 {
 // z_{n+1} = z_n * z_n + c
 ujureZ = reZ * reZ - imZ * imZ + reC;
 ujimZ = 2 * reZ * imZ + imC;
 reZ = ujureZ;
 imZ = ujimZ;

 ++iteracio;
 }

 kepadat[j][k] = iteracio;
}

times (&tmsbuf2);
printf("%ld\n", tmsbuf2.tms_utime - tmsbuf1.tms_utime
 + tmsbuf2.tms_stime - tmsbuf1.tms_stime);

delta = clock () - delta;
printf("%f sec\n", (float) delta / CLOCKS_PER_SEC);
}

int
main (int argc, char *argv[])
{
 if (argc != 2)
 {
 printf("Hasznalat: ./mandelpng fajlnev");
 return -1;
 }
}
```

```
 int depth = 8;
 FILE *fp = fopen(argv[1], "wb");
if(!fp)
 return -1;

png_structp png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, ↵
 NULL, NULL);
if(!png_ptr)
 return -1;

png_infop info_ptr = png_create_info_struct(png_ptr);
if(!info_ptr)
{
 png_destroy_write_struct(&png_ptr, (png_infopp) NULL);
 return -1;
}

if (setjmp(png_jmpbuf(png_ptr)))
{
 png_destroy_write_struct(&png_ptr, &info_ptr);
 fclose(fp);
 return -1;
}

png_init_io(png_ptr, fp);

png_set_IHDR(png_ptr,
 info_ptr,
 MERET,
 MERET,
 depth,
 PNG_COLOR_TYPE_RGB,
 PNG_INTERLACE_NONE,
 PNG_COMPRESSION_TYPE_DEFAULT,
 PNG_FILTER_TYPE_DEFAULT);

png_write_info(png_ptr, info_ptr);

char sor[3*MERET];

int kepadat[MERET][MERET];

mandel(kepadat);

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

```
for(int z = k*3; z < k*3+3; ++z) {
 sor[z] = kepadat[j][k]
}
}
png_write_row(png_ptr, row);
}

printf("%s mentve\n", argv[1]);
}
```

A program használatához Ubuntu-ban szükséges telepíteni az libpng-dev csomagot és fordításkor linkelni a fájlnevet után a `libpng-config --ldflags` szöveget. Eloszor ellenőrizzük egy feltételben hogy futtatáskor megadtuk-e a képszerző nevét azaz 2 argumentumot adtunk-e meg. Jelen esetben ha 255 karakterrel azaz bajt-nál hosszabb nevet adunk meg lefut a program kilep mivel nem tud létrehozni fájlt olyan névvel így megnyitni sem. Mivel fájllal dolgozunk így eloszor meg kell adni egy FILE típusú pointerbe amit fp-nek neveztünk a fájlnak a memóriacímét, ami a fájlkezelő függvények használatához szükséges és az stdio.h fájlt include-olni kell. A fájl memóriacímét az fopen függvény adja meg ami meg is nyitja az első paraméterül adott fájlt, jelen esetben a második paraméter a wb amiben a w azt jelenti hogy ha létezik olyan névű fájl amit az első paraméterben megadtunk akkor kiírja ha nem létezik akkor létrehoz egyet, a b pedig a bináris fájlok kompatibilitásához szükséges. A png fájlkezeléshez szükséges értéket adni a png\_structp és png\_info típusú mutatóknak, amelyeket a png.h fájl kezeli és információt tárolnak az íráshoz és a képről. A setjmp függvény a hibakezeléshez szükséges, az fclose bezárja a fájlt, a png\_destroy\_write\_struct pedig a paraméterként adott mutatókat törli. Hiba esetén, úgynevezett ha nem tud memóriát foglalni valamelyik png függvény akkor 0 értéket ad vissza amit nem jön létre memóriacím és az if feltétel hamis és kilep a program mivel visszatérési értéket kap jelen esetben -1-et. A png\_init\_io függvény a bemenet és kimenetet hozza létre. A png\_set\_IHDR függvényben beállítjuk a kép szélességét, magasságát, bit mélységét és a színkodolást. A png\_write\_info függvénnyel beírjuk az info és a write struktúrákat a képbe. Deklaráljuk a kepadat 2 dimenziós MERETxMERET-es tömböt melynek értékeit a mandel függvény határozza meg és melynek segítségével meghatározzuk a kép pixeleit. A png\_write\_row függvény segítségével írjuk a kép pixeleit a fájlba a memóriacímek segítségével. A printf függvény pedig kiírja a kép nevét és hogy mentve. A png függvények logikai sorrendben működnek csak. A clock függvény megadja az órajelek számát a program kezdete óta, ennek az értéknek a mutatója a colock\_t, jelen esetben delta változóba mentettük az értéket. Mivel csak a mandel függvény használata alatti értéket nezzük így kivonjuk a függvény végén mertből a függvény elején mért értéket és mivel a másodperceket számoljuk ki így le kell osztani a CLOCKS\_PER\_SEC konstánssal ami úgy az órajelek száma másodpercenként. Tizedestört alakot számolunk így típuskenyszerítjük float-ra a delta változót. A times függvény megadja az aktuális processz időt amit a tms struktúra tms1 és tms2 változóba mentünk memóriacímük alapján. A tms\_utime a felhasználói idő, a tms\_stime a rendszer idő ezeket . -tal érhetjük mivel struktúra tagok. Ezek használatához a sys/times.h fájlt includolni kell. Az alapértelmezett iterációhatárt 84 másodperc volt átlépni minden literációt végrehajtva. 1 literációval átlépve pedig 0.1 másodperc volt. A színezéskor úgynevezett a 0 0 0 a fekete szín, ez akkor van ha nem volt az adott kepadat[][] koordináta iteráció, egyébként mivel -1 255-nek felel meg fehér lesz mivel az ITER\_HAT amivel osztunk nagyobb mint a számláló így -1 -nel nagyobb negatív számot kapunk amit -1-re kerekít. Jelen esetben fordítva színez alapbeállítások mellett így hatékonyabb. A memóriafoglaláskor a fordító szegmentálási hiba jelezhet így futtatni a -fno-stack-protector paranccsal kell, mert malloc helyett 2 dimenziós tömböt használtam. Az a, b, c, d változók értékeinek változtatásával nagyíthatunk vagy kicsinyíthetünk a képen. A dx és dy változók felosztjuk 600 egységre az a b és c d

változók közti távolságot, 600 egység után jutunk el a-ból be és d-ből c-be amit a  $reC$  és  $imC$  változókba számolunk. Ezek segítségével számoljuk ki a  $z_{n+1}=z_n*z_n+c$  komplex számot az  $(a,b)*(c,d)=(ac-bd,bc+ad)$  képlet segítségével úgy jelen esetben  $(a,b)$  és  $(c,d)$  megegyezik. A while feltételben ellenőrizzük hogy a következő  $z_{n+1}$  kisebb lesz-e mint 4 és hogy elertük-e az iteráció határát. Jelen esetben ha az első feltétel miatt lép ki fekete, ha a második miatt fehér lesz a pixel. Vannak más színek is csak kisebb számban például 120 és 130 RGB között van 109 pixel a 360000 közül, mivel mindhárom számjegy megegyezik az RGB kódban így a fekete és fehér közötti árnyalat lesz azaz szürke.

Megoldás videó:

Megoldás forrása:

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

```
#include <iostream>
#include "png++/png.hpp"
#include <complex>
using namespace std;
using namespace png;
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
 {
 cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d" << ↵
 endl;

 return -1;
 }
}
```

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

double dx = (b - a) / szelesseg;
double dy = (d - c) / magassag;

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

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

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

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

 ++iteracio;
 }

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

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

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

A program elejen egy if feltetellel ellenorizzuk a futtataskor a parancssori argumentumok szamat, ha ez nem 9 akkor hibauzenettel kilep a program. Az nulladik argumentum ugye a futtatashoz hasznalt nev, a elso a

fajlnev, a második a szélesség, a harmadik a magasság, az negyedik az iterációs határ, az öt, hat, het, nyolcadik az a, b, c, d változók értékei. Ha teljesül a feltétel akkor mentjük az értékeket a megfelelő változókba, melyeknek adtunk kezdőértékeket hogy lassuk mit kell megadni, de a kimenetre nem írtuk ki ezeket mivel csak szövegszerkeztőbe másolva fordítunk. A változókat a szabványos bemenetre adott adatokkal töltjük fel az atoi és atof függvények segítségével. Az atoi a parameterul kapott sztringet egész típusra konvertálja, az atof pedig dupla pontosságu lebegőpontos valós számokra. A :: operátort akkor használjuk ha valaminek az elemeit használjuk például osztálynak vagy std és png. Ezeket elhagyhatjuk ha használjuk a using namespace std és a using namespace png kódokat a program elején. Az image kisebbkacsacsor rgb\_pixel nagyobbkacsacsor kep(szélesség, magasság) paranccsal létrehozunk egy RGB szinkodolasu, 8 bitmelysegu kep nevű szélesség\*magasság pixelszámú fájlt. Aztán beállítjuk az dx és dy változokkal az x és y tengely koordinátáit és létrehozuk a szükséges változókat a kezdőértékükkel. Majd kiírjuk a cout segítségével a Szamitas szót a szabványos kimenetre és újsorba pedig százalekosan a számítás, azaz a külső for ciklus hanyadik sorban jár a sorok számából százalekosan. Ahhoz hogy ez a százalekos érték frissüljön a szabványos kimeneten is használjuk a flush kódot. A reC változóban számoljuk melyik értékkel járunk az a és b változó között, a dx változó segítségével melyben felosztottuk egynelesten a távolságukat és a k változóval melyben az oszlop szám van. Az imC változóban pedig a d változótól a c változóig ugyanúgy csak itt csökkentjük az értéket, a reC változót pedig növeltük így a balfelsősarokban lehet a koordináta-rendszer és az alatta lévő területen számolunk. A complex függvény segítségével létrehozunk double típusú komplex számokat, jelen esetben a c változóba, a változó első parametere a valós rész, a második a képzetes rész amiből létrehozunk egy komplex számot. Létrehozuk a zn változót 0 értékű komplex számként double típussal és az iteráció változó értékét 0-ra állítjuk. A komplex számokat double, float, long double típussal használhatjuk. A while ciklussal minden pixelnek kiszámoljuk a literációját, ami úgy minimum 1 ha jól vettük fel az a, b, c, d változók értékeit. A while ciklusban számoljuk az iterációt amíg a kapott z\_n komplex szám abszolútértéke kisebb mint 4 vagy el nem érjük az iterációs határt. A komplex számok abszolútértékét az abs függvénnyel határozzuk meg. Minden ciklus után az iteráció segítségével beállítjuk a pixel színeit a kep.set\_pixel függvény segítségével, ahol a . előtt a fájlnev van amin állítunk, az első parameter a sorszám, a második az oszlopszám, a harmadik adjuk meg a pixel színt RGB kódban. A % operátor a maradékos osztás jele, amely maradékot adja eredményül. Mivel a harmadik szín 0 így kékek értéke 0 és a színek a piros és zöld keverékeiből állnak elő mert azokat kiszámoljuk. A kep.write függvénnyel a . előtt lévő fájlnev adataiból létrehozuk a parameterként adott képet. Majd kiírjuk hogy mentve a szabványos kimenetre.

Megoldás videó:

Megoldás forrása:

### 5.3. Biomorfok

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

int main (int argc, char *argv[])
{
 int szélesség = 1920;
 int magasság = 1080;
 int iterációsHatar = 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 > image (szelesseg, magassag);
double dx = (xmax - xmin) / szelesseg;
double dy = (ymax - ymin) / magassag;
std::complex<double> cc (reC, imC);
std::cout << "Szamitas\n";
// j megy a sorokon
for (int j = 0; j < magassag; ++j)
{
 // k megy az oszlopokon
 for (int k = 0; k < szelesseg; ++k)
 {
 double reZ = xmin + k * dx;
 double imZ = ymax - j * dy;
 std::complex<double> z_n (reZ, imZ);
 int iteration = 0;
 for (int i=0; i < iteraciosHatar; ++i)
 {
 //z_n = std::pow(z_n, 3) + cc;
 //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
 //z_n=std::pow(z_n,z_n)+std::pow(z_n,6);
 z_n=(1.0-std::pow(z_n,3)/6.0)/std::pow((z_n-std::pow(z_n ,2.0)/2.0),2)+cc;
 if(std::real (z_n) > R || std::imag (z_n) > R)
 {
 iteration = i;
 break;
 }
 }
 }
}
```

```
}
}
image.set_pixel (k, j,
png::rgb_pixel ((iteration*20)%255, (iteration*40)%255, (iteration*60)%255 ←
));
}
int Szazalek = (double) j / (double) magassag * 100.0;
std::cout << "\r" << Szazalek << "%" << std::flush;
}
image.write (argv[1]);
std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
}
```

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat... A program elején deklaráljuk a program működéséhez szükséges változókat, inicializáljuk is őket hogy lássuk mivel működik jól. Majd egy if feltetellel ellenőrizzük a szabványos bemeneten 12 argumentum van-e, mivel logikai feltétel így `==` használunk. Ha nem teljesül a feltétel kilep a program a megadott hibauzenettel, ha teljeseül akkor a megfelelő változókba mentjük az adatokat a megfelelő formátumban az `atof` és `atoi` függvények segítségével. Az `argc` a parancssori argumentumok számát tartja, az `argv[x]` a `x`. parancssori argumentum értékét 0-tól `argc`-ig, a 0-t belértve. Az `atof` a paraméterként kapott sztring típusát `double` típusúra, az `atoi` pedig egész típusúra konvertálja. Az `image` kisebbkacsacsor `rgb_pixel` nagyobbkacsacsor `image ( szélesség, magasság )` parancs segítségével fogunk elkészíteni egy RGB szinkodolással szélesség\*magasság méretű képet. A `dx` és `dy` változókban felosztjuk az `x` és `y` tengelyt egységes közökre ami az `xmin` és `xmax` és `ymín` és `ymax` közötti értékekből áll. Letrehozzuk a `complex` függvény segítségével a `cc` komplex számot, melynek valós értéke az `reC`, képzetes részét az `imC` változóból számítjuk. Majd kiírjuk a szabványos kimenetre a Számítás szót, alatta levo sorba pedig %-os értéket a számításnak, mely a `flush` segítségével minden sor százalek változaskor `no`, amit az ad meg hogy hanyadik sorban jár a számítás százalekosan. Minden koordinátára kiszámoljuk a `zn` komplex számot a `reZ` és `imZ` változókban, amik arányosan lépnek a koordináta-rendszerben. Az iteráció értéket minden `for` ciklus előtt 0-ra állítjuk így ha azzal lép ki a ciklusból hogy elérte az `iteraciosHatar`-t az iteráció értéke 0 lesz, ha pedig hamarabb kilep mert vagy a valós vagy a képzetes része a `z_n` komplex számnak nagyobb mint `R` akkor az iteráció száma egyenlő `i`-vel az addig végrehajtott iterációk számával. Az `i` helyett iteráció-t használva sem egyszerűbb mert nem lokális az iteráció a cikluson kívül. Egy komplex szám valós részét a `real` függvény, a képzetes részét pedig az `imag` függvény határozza meg, melyek `std` osztálybeliek ahogy még több is így a program elejére írhatjuk a `using namespace std` parancsot a szegementálási hibák elkerülésének érdekében `::`-tal kapcsolatban. A `z_n` változó keplete határozza meg a kirajzolt formát, melyek jelen esetben a megfelelően megadott értékekkel biológiai egysejtűekre hasonlítanak majd. A `z_n` kepletében használjuk a `pow` függvényt, ami az első paraméterként adott változót emeli a második paraméterként adott hatványra. Minden koordináta-hoz tartozó pixelnek beállítjuk a színt a `set_pixel` függvény segítségével, jelen esetben minharom színt elteroen hogy színes képeket kapjunk, mivel azonos értékekkel árnyalat jönne ki. A `write` függvény segítségével mentjük a képet a megadott néven, majd kiírjuk a szabványos kimenetre a fájlnevet és hogy mentve. `for` ciklusnál a feltétel részen nem változtat az utasítás rész, de a `while`-nal változtat. A program működik `while` ciklussal is a `status` változó használatával.



## 5.4. A Mandelbrot halmaz CUDA megvalósítása

```
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>

#include <sys/times.h>
#include <iostream>

#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
 // Végigzongorázza a CUDA a szélesség x magasság rácsot:
 // most éppen a j. sor k. oszlopában vagyunk

 // számítás adatai
 float a = -2.0, b = .7, c = -1.35, d = 1.35;
 int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

 // a számítás
 float dx = (b - a) / szelesseg;
 float dy = (d - c) / magassag;
 float reC, imC, reZ, imZ, ujreZ, ujimZ;
 // Hány iterációt csináltunk?
 int iteracio = 0;

 // c = (reC, imC) a rács csomópontjainak
 // megfelelő komplex szám
 reC = a + k * dx;
 imC = d - j * dy;
 // z_0 = 0 = (reZ, imZ)
 reZ = 0.0;
 imZ = 0.0;
 iteracio = 0;
 // z_{n+1} = z_n * z_n + c iterációk
 // számítása, amíg |z_n| < 2 vagy még
 // nem értük el a 255 iterációt, ha
 // viszont elértük, akkor úgy vesszük,
 // hogy a kiindulási c komplex számra
 // az iteráció konvergens, azaz a c a
 // Mandelbrot halmaz eleme
 while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
 {
 // z_{n+1} = z_n * z_n + c
 ujreZ = reZ * reZ - imZ * imZ + reC;
```

```
 ujmZ = 2 * reZ * imZ + imC;
 reZ = ujreZ;
 imZ = ujmZ;

 ++iteracio;

 }
 return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{

 int j = blockIdx.x;
 int k = blockIdx.y;

 kepadat[j + k * MERET] = mandel (j, k);

}
*/

__global__ void
mandelkernel (int *kepadat)
{

 int tj = threadIdx.x;
 int tk = threadIdx.y;

 int j = blockIdx.x * 10 + tj;
 int k = blockIdx.y * 10 + tk;

 kepadat[j + k * MERET] = mandel (j, k);

}

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

 int *device_kepadat;
 cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

 // dim3 grid (MERET, MERET);
 // mandelkernel <<< grid, 1 >>> (device_kepadat);

 dim3 grid (MERET / 10, MERET / 10);
 dim3 tgrid (10, 10);
```

```
mandelkernel <<< grid, tgrid >>> (device_kepadat);

cudaMemcpy (kepadat, device_kepadat,
 MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
cudaFree (device_kepadat);
}

int
main (int argc, char *argv[])
{
 // Mérünk időt (PP 64)
 clock_t delta = clock ();
 // Mérünk időt (PP 66)
 struct tms tmsbuf1, tmsbuf2;
 times (&tmsbuf1);

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

 int kepadat[MERET][MERET];

 cudamandel (kepadat);

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

 for (int j = 0; j < MERET; ++j)
 {
 //sor = j;
 for (int k = 0; k < MERET; ++k)
 {
 kep.set_pixel (k, j,
 png::rgb_pixel (255 -
 (255 * kepadat[j][k]) / ITER_HAT,
 255 -
 (255 * kepadat[j][k]) / ITER_HAT,
 255 -
 (255 * kepadat[j][k]) / ITER_HAT));
 }
 }
 kep.write (argv[1]);

 std::cout << argv[1] << " mentve" << std::endl;

 times (&tmsbuf2);
 std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
```

```
+ tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

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

}
```

A programot parhuzamosan programoztuk a jobb futási idő elérésének érdekében, ez akár 54-szerese is lehet az eredeti verziónak amely nem tartalmazott parhuzamosítást azaz szekvencialisan egymás után hajtottak végre a számítások. Mivel a működéshez szükséges GPU-s kártya a számítógéphez így nem minden gépen működik. Az ilyen CUDA fájlok használatához a kiterjesztésük .cu kell hogy legyen és a fordításkor nvcc-t használunk a g++ helyett. Az nvdia-cuda-toolkit csomag telepítése is szükséges Ubuntu-t használva a működéshez. A parhuzamosítás ennel a programnal nem volt bonyolult mivel minden pixel színet egymástól függetlenül határozzuk meg. Mivel minnél több szálon számítunk annál hamarabb kiszámoljuk így a lehető legtöbb szálhoz hozunk létre tehát annyit ahány pixel van, jelen esetben 600\*600-at. A CUDA-ban van `__global__` és `__device__` típusú függvény. Az előbbi kernel függvénynek is nevezzük, ezt a hárombalra nyíllal és háromjobbnyíllal operátor ele írjuk és ennek parametere ezen nyíllak után lesz, az operátor belsejébe pedig a az indítandó blokkok száma azaz grid változó értéke és a blokkonként indított szálak száma azaz a tgrid változó értéke. A `__device__` típusú függvényt pedig a `__global__` típusú függvénnyel hívhatunk meg, jelen esetben a mandel függvény lesz ilyen típusú és egész típusú is mivel egész számú visszatérési értéket kapunk tőle az egész pixelekhez tartozó iterációt. A mandel függvény paraméterei a k és j egész számú változók, melyeket úgy a `__global__` típusú visszatérési érték nélkül függvény mandelkernel függvény ad meg amelyet úgy futtatunk parhuzamosan. A mandelkernel függvényben határozzuk meg a j és a k változók értékeit a  $\text{blockIdx.x} * \text{blockDim.x} + \text{threadIdx.x}$  képlet segítségével a k-t ugyanígy csak .y-t használunk a .x helyett mivel az y koordinátákra vagyunk kíváncsiak. A blockIdx a blokkok koordinátáit adja meg a ráciban jelen esetben 2 dimenziós rács így x és y koordinátái vannak, a threadIdx a szál koordinátáit adja meg a blokkban mivel jelen esetben a blokk is két dimenziós így x és y koordinátákat. Mivel a CUDA egy dimenziós tombot használ memóriaként így 1 dimenziós tombba mentjük a meghatározott iterációkat a  $\text{kepadat}[j + k * \text{MERET}]$  tombba. A CUDA a balfelső sarokból lefele osztja be a koordinátákat ahogy a mandel függvényben is tettük, azonban a CUDA elso azaz k-ból számolt x koordinátája az oszlopok sorszámát a programunkban viszont a sorokat az y koordináta a j-ből hasonlóan, tehát fordítva kell használni a programunkban és a kimeteskör is a kepadat tomb elem meghatározásakor. Azonban írhattuk volna mandel(k,j)-vel és akkor maradt volna ezen két dolog az eredeti. Úgy a memóriát a CUDA számításokhoz a cudaMalloc függvény segítségével foglalunk úgy minden iterációnak egy egész értéket. A cudaMalloc mutatora mutatót használ a GPU memóriája miatt. A dim3 koddal hozunk létre szálakat és blokkokat a rácson belül, jelen esetben két két dimenzióst. Jelen esetben a háromnyíllal operátor értékeit megadhattuk volna a gridDim.x vagy gridDim.y és blockDim.x és blockDim.y kódokkal is mivel negyzetes két dimenziós tomb a blokkok és a szálak. A gridDim a rács méretét adja meg amit a blokkokból számol ki így gridDim.x az úgy a jelen esetben 60x60-as negyzetet oldalhosszat azaz 60-at jelent, a threadDim ugyanígy csak a blokkok méretét határozza meg a szálakból ami 100x100-as negyzet így a threadDim.x értéke 100 lesz. A cudaMemcpy segítségével másoljuk át a kepadat változóba a device\_kepadat változó értékeit úgy a függvény első parametere hogy honnan a második hogy hova a harmadik hogy hány bajtot a negyedik hogy milyen módon, jelen esetben a host-ról a device-ra. A host az eredeti kód rész a device a CUDA-s. Egydimenziós tombból két dimenziósba másolunk, de ezt a gép megoldja mivel a tombok mérete megegyezik. A cudaMemcpy paranccsal felszabadítjuk a parameterül adott mutató által foglalt memóriát, ami jelen esetben a cudaMalloc által foglalt memória volt.

Megoldás videó:

Megoldás forrása:

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

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

```
//main.cpp
#include <QApplication>
#include "frakablak.h"

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

 FrakAblak w1;
 w1.show();

 /*
 FrakAblak w1,
 w2(-.08292191725019529, -.082921917244591272,
 -.9662079988595939, -.9662079988551173, 600, 3000),
 w3(-.08292191724880625, -.0829219172470933,
 -.9662079988581493, -.9662079988563615, 600, 4000),
 w4(.14388310361318304, .14388310362702217,
 .6523089200729396, .6523089200854384, 600, 38655);
 w1.show();
 w2.show();
 w3.show();
 w4.show();
 */
 return a.exec();
}
```

```
//frakablak.h
#ifndef FRAKABLAH_H
#define FRAKABLAH_H

#include <QMainWindow>
#include <QImage>
#include <QPainter>
#include <QMouseEvent>
#include <QKeyEvent>
#include "frakaszal.h"
```

```
class FrakSzal;

class FrakAblak : public QMainWindow
{
 Q_OBJECT

public:
 FrakAblak(double a = -2.0, double b = .7, double c = -1.35,
 double d = 1.35, int szelesseg = 600,
 int iteraciosHatar = 255, QWidget *parent = 0);
 ~FrakAblak();
 void vissza(int magassag , int * sor, int meret) ;
 void vissza(void) ;
 // A komplex sík vizsgált tartománya [a,b]x[c,d].
 double a, b, c, d;
 // A komplex sík vizsgált tartományára feszített
 // háló szélessége és magassága.
 int szelesseg, magassag;
 // Max. hány lépésig vizsgáljuk a $z_{n+1} = z_n * z_n + c$ iterációt?
 // (tk. most a nagyítási pontosság)
 int iteraciosHatar;

protected:
 void paintEvent(QPaintEvent*);
 void mousePressEvent(QMouseEvent*);
 void mouseMoveEvent(QMouseEvent*);
 void mouseReleaseEvent(QMouseEvent*);
 void keyPressEvent(QKeyEvent*);

private:
 QImage* fraktal;
 FrakSzal* mandelbrot;
 bool szamitasFut;
 // A nagyítandó kijelölt területet bal felső sarka.
 int x, y;
 // A nagyítandó kijelölt terület szélessége és magassága.
 int mx, my;
};

#endif // FRAKABLAH_H
```

```
//frakszal.h
#ifndef FRAKSZAL_H
#define FRAKSZAL_H

#include <QThread>
#include <cmath>
```

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

class FrakAblak;

class FrakSzal : public QThread
{
 Q_OBJECT

public:
 FrakSzal(double a, double b, double c, double d,
 int szelesseg, int magassag, int iteraciosHatar, FrakAblak * ←
 frakAblak);
 ~FrakSzal();
 void run();

protected:
 // A komplex sík vizsgált tartománya [a,b]x[c,d].
 double a, b, c, d;
 // A komplex sík vizsgált tartományára feszített
 // háló szélessége és magassága.
 int szelesseg, magassag;
 // Max. hány lépésig vizsgáljuk a $z_{n+1} = z_n * z_n + c$ iterációt?
 // (tk. most a nagyítási pontosság)
 int iteraciosHatar;
 // Kinek számolok?
 FrakAblak* frakAblak;
 // Soronként küldöm is neki vissza a kiszámoltakat.
 int* egySor;
};

#endif // FRAKSZAL_H
```

```
//frakablak.cpp
#include "frakablak.h"

FrakAblak::FrakAblak(double a, double b, double c, double d,
 int szelesseg, int iteraciosHatar, QWidget *parent)
 : QMainWindow(parent)
{
 setWindowTitle("Mandelbrot halmaz");

 szamitasFut = true;
 x = y = mx = my = 0;
 this->a = a;
 this->b = b;
 this->c = c;
```

```
this->d = d;
this->szelesseg = szelesseg;
this->iteraciosHatar = iteraciosHatar;
magassag = (int)(szelesseg * ((d-c)/(b-a)));

setFixedSize(QSize(szelesseg, magassag));
fraktal= new QImage(szelesseg, magassag, QImage::Format_RGB32);

mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
 iteraciosHatar, this);
mandelbrot->start();
}

FrakAblak::~FrakAblak()
{
 delete fraktal;
 delete mandelbrot;
}

void FrakAblak::paintEvent(QPaintEvent*) {
 QPainter qpainter(this);
 qpainter.drawImage(0, 0, *fraktal);
 if(!szamitasFut) {
 qpainter.setPen(QPen(Qt::white, 1));
 qpainter.drawRect(x, y, mx, my);
 }
 qpainter.end();
}

void FrakAblak::mousePressEvent(QMouseEvent* event) {

 // A nagyítandó kijelölt területet bal felső sarka:
 x = event->x();
 y = event->y();
 mx = 0;
 my = 0;

 update();
}

void FrakAblak::mouseMoveEvent(QMouseEvent* event) {

 // A nagyítandó kijelölt terület szélessége és magassága:
 mx = event->x() - x;
 my = mx; // négyzet alakú

 update();
}
```



```
void FrakAblak::mouseReleaseEvent(QMouseEvent* event) {

 if(szamitasFut)
 return;

 szamitasFut = true;

 double dx = (b-a)/szelesseg;
 double dy = (d-c)/magassag;

 double a = this->a+x*dx;
 double b = this->a+x*dx+mx*dx;
 double c = this->d-y*dy-my*dy;
 double d = this->d-y*dy;

 this->a = a;
 this->b = b;
 this->c = c;
 this->d = d;

 delete mandelbrot;
 mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
 iteraciosHatar, this);
 mandelbrot->start();

 update();
}

void FrakAblak::keyPressEvent(QKeyEvent *event)
{

 if(szamitasFut)
 return;

 if (event->key() == Qt::Key_N)
 iteraciosHatar *= 2;
 szamitasFut = true;

 delete mandelbrot;
 mandelbrot = new FrakSzal(a, b, c, d, szelesseg, magassag, ←
 iteraciosHatar, this);
 mandelbrot->start();

}

void FrakAblak::vissza(int magassag, int *sor, int meret)
{
 for(int i=0; i<meret; ++i) {
 QRgb szin = qRgb(0, 255-sor[i], 0);
```

```
 fraktal->setPixel(i, magassag, szin);
 }
 update();
}
```

```
void FrakAblak::vissza(void)
{
 szamitasFut = false;
 x = y = mx = my = 0;
}
```

```
// frakszal.cpp
#include "frakszal.h"
```

```
FrakSzal::FrakSzal(double a, double b, double c, double d,
 int szelesseg, int magassag, int iteraciosHatar, ←
 FrakAblak *frakAblak)
```

```
{
 this->a = a;
 this->b = b;
 this->c = c;
 this->d = d;
 this->szelesseg = szelesseg;
 this->iteraciosHatar = iteraciosHatar;
 this->frakAblak = frakAblak;
 this->magassag = magassag;

 egySor = new int[szelesseg];
}
```

```
FrakSzal::~~FrakSzal()
{
 delete[] egySor;
}
```

```
// A szál kódját a Javát tanítokhoz írt Java kódomból vettem át
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
// mivel itt az algoritmust is leírtam/lerajzoltam, így meghagytam
// a kommenteket, hogy a hallgató könnyen hozzáolvashassa az "elméletet",
// ha érdekli.
```

```
void FrakSzal::run()
{
 // A [a,b]x[c,d] tartományon milyen sűrű a
 // megadott szélesség, magasság háló:
 double dx = (b-a)/szelesseg;
 double dy = (d-c)/magassag;
 double reC, imC, reZ, imZ, ujreZ, ujimZ;
 // Hány iterációt csináltunk?
```

```
int iteracio = 0;
// Végigzongorázzuk a szélesség x magasság hálót:
for(int j=0; j<magassag; ++j) {
 //sor = j;
 for(int k=0; k<szelesseg; ++k) {
 // c = (reC, imC) a háló rácspontjainak
 // megfelelő komplex szám
 reC = a+k*dx;
 imC = d-j*dy;
 // z_0 = 0 = (reZ, imZ)
 std::complex<double> c(reC, imC);

 reZ = 0;
 imZ = 0;
 std::complex<double> z_n(reZ, imZ);
 iteracio = 0;
 // z_{n+1} = z_n * z_n + c iterációk
 // számítása, amíg |z_n| < 2 vagy még
 // nem értük el a 255 iterációt, ha
 // viszont elértük, akkor úgy vesszük,
 // hogy a kiindulási c komplex számra
 // az iteráció konvergens, azaz a c a
 // Mandelbrot halmaz eleme
 /*
 while(reZ*reZ + imZ*imZ < 4 && iteracio < iteraciosHatar) {
 // z_{n+1} = z_n * z_n + c
 ujureZ = reZ*reZ+ std::atan(reZ*reZ - imZ*imZ) + std::sqrt(reC);
 ujimZ = 2*reZ*imZ+std::atan(2*reZ*imZ) + imC;
 reZ = ujureZ;
 imZ = ujimZ;
 ++iteracio;
 }
 */
 while(std::abs(z_n) < 4 && iteracio < iteraciosHatar) {
 z_n = z_n * z_n + c;

 ++iteracio;
 }

 // ha a < 4 feltétel nem teljesült és a
 // iteráció < iterációsHatár sérülésével lépett ki, azaz
 // feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c
 // sorozat konvergens, azaz iteráció = iterációsHatár
 // ekkor az iteráció %= 256 egyenlő 255, mert az esetleges
 // nagyítások során az iteráció = valahány * 256 + 255

 iteracio %= 256;

 //a színezést viszont már majd a FrakAblak osztályban lesz
 egySor[k] = iteracio;
 }
}
```

```
 }
 // Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.
 frakAblak->vissza(j, egySor, szelesseg);
}
frakAblak->vissza();

}
```

A program mukodesehez 2 header filet es 3 cpp fajlt keszitettunk melyek futtatashoz a qt keretrendszer es a q4-make szukseges. A QApplication osztalynak minden applikacio létrehozaskor eloszor létre kell hozni egy objektumot, amit jelen esetben a-nak neveztunk. Létrehozunk egy w1 objektumot a FrakAblak osztalynak így az alapparameterekkel fog mukodni, de megadhatunk mas parametereket is. A show fuggvennyel megjelenitjuk az adott objektum által keszített képet. Az exec fuggvény felfuggeszti program futast egy vegtelen amig jelen esetben ugye be nem zarjuk az ablakot, ekkor a visszateresi erteke 0 lesz. E fuggvény hasznalata nelkul miutan kirajzolna a képet az alapbeallitasokkal es kilépne a program, de mivel az exec fuggvény visszaep a fo programreszbe es var ugye ameddig nem zarjuk be az ablakot, így addig nagyitathatunk a billentyu es eger utasitas feldolgozo fuggvényekkel. A header fajok elejen az #ifndef definialasi felttel ellenroiz hogy volt e mar definialva az adott nevu keret ha volt akkor az #endif-hez ugrik, ha nem akkor a kovetkezo sorra, ahol a #define paranccsal definialjuk az adott nevu keretet az #endif-ig. A frakablak.h fajlba létrehozuk a FrakAblak osztalyt melyben hivatkozunk a QMainWindow osztalyra ami a fo ablak es amit megjelenitunk majd a FrakAblak osztalyra hivatkozott show fuggvennyel. A Q\_OBJECT szo hasznalata a billentyu es eger utasitasos fuggvények hasznalata miatt szukseges. Az osztaly elemei alapvetoen privat reszbe tartoznak, de lehetnek publikusak is ha úgy definialjuk oket. Jelen esetben a FrakAblak a foosztaly es a QMainWindow pedig a szarmaztatott osztaly, ekkor a privat osztalyokat csak a foosztalyok erhetik el, a vedett osztalyokat a foosztaly es a szarmaztatott osztalyok erhetik el, a publikust minden osztaly eleri. Létrehozunk egy parametizált konstruktort a FrakAblak osztalyhoz amelynek parametereit kezdertekekkel adjuk meg hogy mukodjon a w1, az utolso parametere a QWidget \*parent = 0, amely mutatót hoz létre a QWidget-nek parent neven 0 kezdertekekkel, tehát nemszukseges neki ertekeket adni objektum létrehozaskor. A Q\_OBJECT es ez szukseges majd a szulo-gyerek kapcsolat kialakitasahoz. Létrehozuk a FrakAblak destruktort is, a hasznalni kívánt fuggvényeket, változókat deklaráljuk, a nagyitashoz szukseges beepített fuggvényeket es az osztalyokra mutató mutatókat. Mivel nem tudni melyik header file fut le hamarabb a ketto kozul így mindketto elejen létrehoztunk egy osztalyt mert mutatót hozunk létre bennuk a masikra es szekvencialisan futnak eloszor valamelyikuk. A frakszal.h fajlba a FrakSzal osztaly szarmaztatott osztalya a QThread, ami olyan modszert biztosít a szalak kezelesehez amely fuggetlen az adott programhoz szukseges hardver- es a szoftverkornyezettol, mivel szegemnséket hasznal így a Q\_OBJECT szót is hasznaljuk. Létrehozuk a FrakSzal konstruktort a megfelelo parameterekkel melyek definialt változok es frakAblak nevu mutató a FrakAblak osztalyra. Dekaráljuk a Frakszal destruktort, a hasznalando fuggvényeket es változókat es egy egesz tipusu egySor mutatót. Mivel a QWidget \*parent = 0-nek nem adunk meg ertekeket így ablak lesz, amit a show fuggvennyel jelenitunk meg. A : QMainWindow(parent) teszi lehetove hogy a foablakkal modositására modosuljon a gyermek ablaka, így bezaraskor bezarul az is. A setWindowTitle fuggvennyel beallithatjuk az ablak címet jelen esetben Mandelbrot halmaz-ra. A this->a = a; esetekben a this mutató teszi lehetove hogy az objektumban használt a változo ertekeket hasznaljuk kulso változokban is mas szamitasokhoz, ehhez ugyanaz a változonev szuksegges a megfelelo konstruktori változoknak es a kulso változoknak. Ahhoz hogy szabalyos ábrát kapjunk es szabalyosan szamoljuk ki a pixel ertekeket a szamitashoz használt oldalarnyoknak meg kell egyeznie a kép pixel oldalarnyaival, az oldalarny a szelesseg es magassag hanyadosa. Mivel a pixel magassagát kepletesen kiszamoljuk, így bármely tetszolegesen meg-

adott másik három adattal működik. A `setFixedSize(QSize(szelesseg, magassag));` függvénnyel beállítjuk az ablak méretét fixre hogy ne változzon a program működése alatt. Letrehozunk egy `QImage` objektumot fraktal néven, a `QImage` konstruktor adataiban megadjuk a pixel szélességet és magasságát a formálendő képnek és a szinkodolást, ami jelen esetben `RGB32` lesz, a méretek pedig az ablak méretei. Letrehozunk mandelbrot néven egy `FrakSzal` objektumot, a konstruktor adatait melyben szerepel a `this` mutató, ami tárolja azt aktuális objektum memóriacímét, jelen esetben ugye a mandelbrot értéke. A `->` operátort akkor használjuk ha mutatóval hivatkozunk egy objektum függvényére vagy változójára. A `start` beépített függvény elkezd a szálak végrehajtását a `run` függvény hívásával. Letrehozzuk a `FrakAblak` destruktort amely akkor hívódik meg ha kilepünk a programból illetve ha használjuk a `delete` függvényt melyet tartalmaz is. A `delete` függvényt objektumok és sorok által foglalt memória törlésére használjuk, sorok törlése esetén `delete[]` formában használjuk. A `QPainter` osztály segítségével újrarajzoljuk a képet a fraktal objektum alapján. Az `if` feltételben megadjuk hogy ha `szamitasFut` hamis akkor rajzol egy teglalapot aminek az `(x,y)` koordinata lesz a bal felső sarka és `mx` a szélessége és `my` a magassága, az oldalonala pedig fehér és 1 pixel vastagságú. Ha egy objektum függvényeit vagy változóit az objektumnévvel használjuk. operátort használunk ahogy jelen esetben a `QPainter` osztály `qpainter` objektumával. A `szamitasFut` igaz-hamis értékű változó akkor lesz hamis amikor az egerrel való kijelölést végezzük, és ez a `PaintEvent`-es függvény rajzolja ki minden `update` függvényhíváskor az ablak tartalmát, mivel újrarajzolást is végzünk így az `end` függvényt használjuk. Az `x` és `y` függvények visszáadják az `x` és `y` koordinátákat az egernek, amik ugye a `QMouseEvent` osztályban vannak aminek a mutatója az `event` és a `->` operátorral használjuk a függvényeket. A `MouseEvent` osztályt kétszer használva kiszámoljuk a kijelölt négyzet oldalait a bal felső sarkanak mentett koordinátaival. A kijelöléskor a kijelölt részeknél kattintást végzél csak egy folyamatként. Az `mx` és `my`-t 0-ra állítjuk hogy újabbnagyításkor nerajzolja ki kattintásra az előző nagyításkor használt értékekkel egy négyzetet. A `mouseReleaseEvent` gyerekosztályában a `this` mutató segítségével definiáljuk a megfelelő változó értékeit, a `dx` és `dy` ugye a beosztások száma az eredeti marad, az `x` és `y` értékeit állítjuk be az alsó szélső értékeknek, az `mx+dx` és `my+dy` értéket pedig felső értékeknek, ezek azonban 0 és magasság illetve szélesség közöttiek, de a számítások az `a`, `b`, `c`, `d` változókat nem feltétlenül 0-nak adjuk meg. A `delete` mandelbrot-tal ugye hívjuk a konstruktort, ami ugye tróli a mandelbrot objektumot is és újat hozunk létre, majd a `start` függvényt használjuk. A `keyPressEvent` függvényben is ha a `szamitasFut` igaz akkor kilep a függvény. A `key` függvény értéke, azaz a lenyomott billentyűzet a `N` az `iteraciosHatar`-t a kétszeresére noveli, így színesebb lesz a kép, ezzel elesítünk rajta. A számítás hasonlóan megy végbe az egerrel kapcsolattal töröljük a régi objektumot és létrehozunk egy újat. A vissza függvényt definiáltuk paraméterrel és paraméter nélkül is. A paraméteres vissza függvényel beállítjuk a megfelelő pixelszíneket az RGB kód alapján a `for` ciklus segítségével mivel az egyes oszlopok minden sorával és a hozzátartozó egy `Sor` tomb oszlopaival ter vissza a vissza függvény, a paraméter nélküli vissza függvény pedig `false`-ra azaz hamisra állítja a `szamitasFut` változó értékét és az `x`, `y`, `mx`, `my` változók értékét 0-ra állítja. A `frakszal.cpp` fájlban a `FrakSzal` konstruktorban beállítjuk hogy a lokális változó értékeit ugye a `this` mutatóval és létrehozunk egy `Sor` néven egy két dimenziós tombot szélesség mennyiségű egész típusú helyet foglalva a `new` operátor segítségével. Letrehozzuk a `FrakSzal` destruktort amelyben töröljük a `delete` operátor segítségével az egy `Sor` által foglalt memóriahelyeket. A `run` beépített függvényben a Mandelbrot halmaz számítását végezzük ugye az iteráció meghatározásával, az iteráció `%= 256`; alapján az iteráció bármely `iteraciosHatar` esetén 0 és 256 közötti egész szám, mivel a `%` jel a maradékos osztás jele. A kiszámolt iterációként mentjük az egy `Sor` tomb sorszámmal megegyező elemeibe. Miután minden pixelhez kiszámoltuk a megfelelő iterációkat meghívódik a paraméter nélküli vissza függvény.

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

## 5.6. Mandelbrot nagyító és utazó Java nyelven

```
//MandelbrotHalmaz.java
public class MandelbrotHalmaz extends java.awt.Frame implements Runnable {
 /** A komplex sík vizsgált tartománya [a,b]x[c,d]. */
 protected double a, b, c, d;
 /** A komplex sík vizsgált tartományára feszített
 * háló szélessége és magassága. */
 protected int szélesség, magasság;
 /** A komplex sík vizsgált tartományára feszített hálónak megfelelő kép ←
 */
 protected java.awt.image.BufferedImage kép;
 /** Max. hány lépésig vizsgáljuk a $z_{n+1} = z_n * z_n + c$ iterációt?
 * (tk. most a nagyítási pontosság) */
 protected int iterációsHatár = 255;
 /** Jelzi, hogy éppen megy-e a számítás? */
 protected boolean számításFut = false;
 /** Jelzi az ablakban, hogy éppen melyik sort számoljuk. */
 protected int sor = 0;
 /** A pillanatfelvételek számozásához. */
 protected static int pillanatfelvételSzámláló = 0;
 /**
 * Létrehoz egy a Mandelbrot halmazt a komplex sík
 * [a,b]x[c,d] tartománya felett kiszámoló
 * <code>MandelbrotHalmaz</code> objektumot.
 *
 * @param a a [a,b]x[c,d] tartomány a koordinátája.
 * @param b a [a,b]x[c,d] tartomány b koordinátája.
 * @param c a [a,b]x[c,d] tartomány c koordinátája.
 * @param d a [a,b]x[c,d] tartomány d koordinátája.
 * @param szélesség a halmazt tartalmazó tömb szélessége.
 * @param iterációsHatár a számítás pontossága.
 */
 public MandelbrotHalmaz(double a, double b, double c, double d,
 int szélesség, int iterációsHatár) {
 this.a = a;
 this.b = b;
 this.c = c;
 this.d = d;
 this.szélesség = szélesség;
 this.iterációsHatár = iterációsHatár;
 // a magasság az (b-a) / (d-c) = szélesség / magasság
 // arányból kiszámolva az alábbi lesz:
 this.magasság = (int)(szélesség * ((d-c)/(b-a)));
 // a kép, amire rárajzoljuk majd a halmazt
 kép = new java.awt.image.BufferedImage(szélesség, magasság,
 java.awt.image.BufferedImage.TYPE_INT_RGB);
 // Az ablak bezárásakor kilépünk a programból.
```

```
addWindowListener(new java.awt.event.WindowAdapter() {
 public void windowClosing(java.awt.event.WindowEvent e) {
 setVisible(false);
 System.exit(0);
 }
});
// A billentyűzetről érkező események feldolgozása
addKeyListener(new java.awt.event.KeyAdapter() {
 // Az 's', 'n' és 'm' gombok lenyomását figyeljük
 public void keyPressed(java.awt.event.KeyEvent e) {
 if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
 pillanatfelvétel();
 // Az 'n' gomb benyomásával pontosabb számítást végzünk.
 else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
 if(számításFut == false) {
 MandelbrotHalmaz.this.iterációsHatár += 256;
 // A számítás újra indul:
 számításFut = true;
 new Thread(MandelbrotHalmaz.this).start();
 }
 // Az 'm' gomb benyomásával pontosabb számítást végzünk ↔
 '
 // de közben sokkal magasabbra vesszük az iterációs
 // határt, mint az 'n' használata esetén
 } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_M) {
 if(számításFut == false) {
 MandelbrotHalmaz.this.iterációsHatár += 10*256;
 // A számítás újra indul:
 számításFut = true;
 new Thread(MandelbrotHalmaz.this).start();
 }
 }
 }
});
// Ablak tulajdonságai
setTitle("A Mandelbrot halmaz");
setResizable(false);
setSize(szélesség, magasság);
setVisible(true);
// A számítás indul:
számításFut = true;
new Thread(this).start();
}
/** A halmaz aktuális állapotának kirajzolása. */
public void paint(java.awt.Graphics g) {
 // A Mandelbrot halmaz kirajzolása
 g.drawImage(kép, 0, 0, this);
 // Ha éppen fut a számítás, akkor egy vörös
 // vonallal jelöljük, hogy melyik sorban tart:
 if(számításFut) {
```

```
 g.setColor(java.awt.Color.RED);
 g.drawLine(0, sor, getWidth(), sor);
 }
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
 paint(g);
}
/** Pillanatfelvételek készítése. */
public void pillanatfelvétel() {
 // Az elmentendő kép elkészítése:
 java.awt.image.BufferedImage mentKép =
 new java.awt.image.BufferedImage(szélesség, magasság,
 java.awt.image.BufferedImage.TYPE_INT_RGB);
 java.awt.Graphics g = mentKép.getGraphics();
 g.drawImage(kép, 0, 0, this);
 g.setColor(java.awt.Color.BLUE);
 g.drawString("a=" + a, 10, 15);
 g.drawString("b=" + b, 10, 30);
 g.drawString("c=" + c, 10, 45);
 g.drawString("d=" + d, 10, 60);
 g.drawString("n=" + iterációsHatár, 10, 75);
 g.dispose();
 // A pillanatfelvétel képfájl nevének képzése:
 StringBuffer sb = new StringBuffer();
 sb = sb.delete(0, sb.length());
 sb.append("MandelbrotHalmaz_");
 sb.append(++pillanatfelvételSzámláló);
 sb.append("_");
 // A fájl nevébe belevevesszük, hogy melyik tartományban
 // találtuk a halmazt:
 sb.append(a);
 sb.append("_");
 sb.append(b);
 sb.append("_");
 sb.append(c);
 sb.append("_");
 sb.append(d);
 sb.append(".png");
 // png formátumú képet mentünk
 try {
 javax.imageio.ImageIO.write(mentKép, "png",
 new java.io.File(sb.toString()));
 } catch (java.io.IOException e) {
 e.printStackTrace();
 }
}
/**
 * A Mandelbrot halmaz számítási algoritmus.
```



```
* Az algoritmus részletes ismertetését lásd például a
* [BARNSELY KÖNYV] (M. Barnsley: Fractals everywhere,
* Academic Press, Boston, 1986) hivatkozásban vagy
* ismeretterjesztő szinten a [CSÁSZÁR KÖNYV] hivatkozásban.
*/
public void run() {
 // A [a,b]x[c,d] tartományon milyen sűrű a
 // megadott szélesség, magasság háló:
 double dx = (b-a)/szélesség;
 double dy = (d-c)/magasság;
 double reC, imC, reZ, imZ, ujreZ, ujimZ;
 int rgb;
 // Hány iterációt csináltunk?
 int iteráció = 0;
 // Végigzongorázzuk a szélesség x magasság hálót:
 for(int j=0; j<magasság; ++j) {
 sor = j;
 for(int k=0; k<szélesség; ++k) {
 // c = (reC, imC) a háló rácspontjainak
 // megfelelő komplex szám
 reC = a+k*dx;
 imC = d-j*dy;
 // z_0 = 0 = (reZ, imZ)
 reZ = 0;
 imZ = 0;
 iteráció = 0;
 // z_{n+1} = z_n * z_n + c iterációk
 // számítása, amíg |z_n| < 2 vagy még
 // nem értük el a 255 iterációt, ha
 // viszont elértük, akkor úgy vesszük,
 // hogy a kiindulási c komplex számra
 // az iteráció konvergens, azaz a c a
 // Mandelbrot halmaz eleme
 while(reZ*reZ + imZ*imZ < 4 && iteráció < iterációsHatár) {
 // z_{n+1} = z_n * z_n + c
 ujreZ = reZ*reZ - imZ*imZ + reC;
 ujimZ = 2*reZ*imZ + imC;
 reZ = ujreZ;
 imZ = ujimZ;

 ++iteráció;
 }
 // ha a < 4 feltétel nem teljesült és a
 // iteráció < iterációsHatár sérülésével lépett ki, azaz
 // feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c
 // sorozat konvergens, azaz iteráció = iterációsHatár
 // ekkor az iteráció %= 256 egyenlő 255, mert az esetleges
 // nagyítások során az iteráció = valahány * 256 + 255
 iteráció %= 256;
 }
 }
}
```

```
 // így a halmaz elemeire 255-255 értéket használjuk,
 // azaz (Red=0,Green=0,Blue=0) fekete színnel:
 rgb = (255-iteráció) |
 ((255-iteráció) << 8) |
 ((255-iteráció) << 16);
 // rajzoljuk a képre az éppen vizsgált pontot:
 kép.setRGB(k, j, rgb);
 }
 repaint();
}
számításFut = false;
}
/** Az aktuális Mandelbrot halmaz [a,b]x[c,d] adatai.
 * @return double a */
public double getA() {
 return a;
}
/** Az aktuális Mandelbrot halmaz [a,b]x[c,d] adatai.
 * @return double b */
public double getB() {
 return b;
}
/** Az aktuális Mandelbrot halmaz [a,b]x[c,d] adatai.
 * @return double c */
public double getC() {
 return c;
}
/** Az aktuális Mandelbrot halmaz [a,b]x[c,d] adatai.
 * @return double d */
public double getD() {
 return d;
}
/** Az aktuális Mandelbrot halmaz feletti rács adatai.
 * @return int szélesség */
public int getSz() {
 return szélesség;
}
/** Az aktuális Mandelbrot halmaz feletti rács adatai.
 * @return int magasság */
public int getM() {
 return magasság;
}
/** Az aktuális Mandelbrot halmazt tartalmazó kép.
 * @return BufferedImage kép */
public java.awt.image.BufferedImage kép() {
 return kép;
}
/** Példányosít egy Mandelbrot halmazt kiszámoló obektumot. */
public static void main(String[] args) {
 // A halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35] tartományában
```

```
 // keressük egy 400x400-as hálóval:
 new MandelbrotHalmaz(-2.0, .7, -1.35, 1.35, 600, 255);
 }
}
```

```
//MandelbrotHalmazNagyító.java
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
 /** A nagyítandó kijelölt területet bal felső sarka. */
 private int x, y;
 /** A nagyítandó kijelölt terület szélessége és magassága. */
 private int mx, my;
 /**
 * Létrehoz egy a Mandelbrot halmazt a komplex sík
 * [a,b]x[c,d] tartománya felett kiszámoló és nygítani tudó
 * <code>MandelbrotHalmazNagyító</code> objektumot.
 *
 * @param a a [a,b]x[c,d] tartomány a koordinátája.
 * @param b a [a,b]x[c,d] tartomány b koordinátája.
 * @param c a [a,b]x[c,d] tartomány c koordinátája.
 * @param d a [a,b]x[c,d] tartomány d koordinátája.
 * @param szélesség a halmazt tartalmazó tömb szélessége.
 * @param iterációsHatár a számítás pontossága.
 */
 public MandelbrotHalmazNagyító(double a, double b, double c, double d,
 int szélesség, int iterációsHatár) {
 // Az ős osztály konstruktorának hívása
 super(a, b, c, d, szélesség, iterációsHatár);
 setTitle("A Mandelbrot halmaz nagyításai");
 // Egér kattintó események feldolgozása:
 addMouseListener(new java.awt.event.MouseAdapter() {
 // Egér kattintással jelöljük ki a nagyítandó területet
 // bal felső sarkát vagy ugyancsak egér kattintással
 // vizsgáljuk egy adott pont iterációit:
 public void mousePressed(java.awt.event.MouseEvent m) {
 // Az egérmutató pozíciója
 x = m.getX();
 y = m.getY();
 // Az 1. egér gombbal a nagyítandó terület kijelölését
 // végezzük:
 if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
 // A nagyítandó kijelölt területet bal felső sarka: (x, ↵
 y)
 // és szélessége (majd a vonszolás növeli)
 mx = 0;
 my = 0;
 repaint();
 } else {
 // Nem az 1. egér gombbal az egérmutató mutatta c
```

```
 // komplex számból indított iterációkat vizsgálhatjuk
 MandelbrotIterációk iterációk =
 new MandelbrotIterációk(
 MandelbrotHalmazNagyító.this, 50);
 new Thread(iterációk).start();
 }
}

// Vonszolva kijelölünk egy területet...
// Ha felengedjük, akkor a kijelölt terület
// újraszámítása indul:
public void mouseReleased(java.awt.event.MouseEvent m) {
 if(m.getButton() == java.awt.event.MouseEvent.BUTTON1) {
 double dx = (MandelbrotHalmazNagyító.this.b
 - MandelbrotHalmazNagyító.this.a)
 /MandelbrotHalmazNagyító.this.szélesség;
 double dy = (MandelbrotHalmazNagyító.this.d
 - MandelbrotHalmazNagyító.this.c)
 /MandelbrotHalmazNagyító.this.magasság;
 // Az új Mandelbrot nagyító objektum elkészítése:
 new MandelbrotHalmazNagyító(
 MandelbrotHalmazNagyító.this.a+x*dx,
 MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
 MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
 MandelbrotHalmazNagyító.this.d-y*dy,
 600,
 MandelbrotHalmazNagyító.this.iterációsHatár);
 }
}

});

// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
 // Vonszolással jelöljük ki a négyzetet:
 public void mouseDragged(java.awt.event.MouseEvent m) {
 // A nagyítandó kijelölt terület szélessége és magassága:
 mx = m.getX() - x;
 my = m.getY() - y;
 repaint();
 }
});

}

/**
 * Pillanatfelvételek készítése.
 */
public void pillanatfelvétel() {
 // Az elementendő kép elkészítése:
 java.awt.image.BufferedImage mentKép =
 new java.awt.image.BufferedImage(szélesség, magasság,
 java.awt.image.BufferedImage.TYPE_INT_RGB);
 java.awt.Graphics g = mentKép.getGraphics();
 g.drawImage(kép, 0, 0, this);
}
```

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

```

 g.setColor(java.awt.Color.GREEN);
 g.drawRect(x, y, mx, my);
 }
 /**
 * Hol áll az egérmutató?
 * @return int a kijelölt pont oszlop pozíciója.
 */
 public int getX() {
 return x;
 }
 /**
 * Hol áll az egérmutató?
 * @return int a kijelölt pont sor pozíciója.
 */
 public int getY() {
 return y;
 }
 /**
 * Példányosít egy Mandelbrot halmazt nagyító obektumot.
 */
 public static void main(String[] args) {
 // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
 // tartományában keressük egy 600x600-as hálózattal és az
 // aktuális nagyítási pontossággal:
 new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
 }
}

```

```

//MandelbrotIterációk.java
public class MandelbrotIterációk implements Runnable{
 /** Mennyi időt várakozunk két iteráció bemutatása között? */
 private int várakozás;
 // Kissé igaz redundánsan, s nem szépen, de kényelmesen:
 private MandelbrotHalmazNagyító nagyító;
 private int j, k;
 private double a, b, c, d;
 private int szélesség, magasság;
 private java.awt.image.BufferedImage kép;
 /**
 * Létrehoz egy iterációkat vizsgáló <code>MandelbrotIterációk</code>
 * szál objektumot egy adott <code>MandelbrotHalmaznagyító</code>
 * objektumhoz.
 *
 * @param nagyító egy <code>MandelbrotHalmazNagyító</code> ↵
 * objektum
 * @param várakozás várakozási idő
 */
 public MandelbrotIterációk(MandelbrotHalmazNagyító nagyító, int ↵

```

```
várákozás) {
 this.nagyító = nagyító;
 this.várákozás = várákozás;
 j = nagyító.getY();
 k = nagyító.getX();
 a = nagyító.getA();
 b = nagyító.getB();
 c = nagyító.getC();
 d = nagyító.getD();
 kép = nagyító.kép();
 szélesség = nagyító.getSz();
 magasság = nagyító.getM();
}

/** Az vizsgált pontból induló iterációk bemutatása. */
public void run() {
 /* Az alábbi kód javarészt a MandelbrotHalmaz.java számolást
 végző run() módszeréből származik, hiszen ugyanazt csináljuk,
 csak most nem a hálón megyünk végig, hanem a háló adott a
 példányosításunkkor az egérmutató mutatta csomópontjában (ennek
 felel meg a c komplex szám) számolunk, tehát a két külső for
 ciklus nem kell. */
 // A [a,b]x[c,d] tartományon milyen sűrű a
 // megadott szélesség, magasság háló:
 double dx = (b-a)/szélesség;
 double dy = (d-c)/magasság;
 double reC, imC, reZ, imZ, ujreZ, ujimZ;
 // Hány iterációt csináltunk?
 int iteráció = 0;
 // c = (reC, imC) a háló rácspontjainak
 // megfelelő komplex szám
 reC = a+k*dx;
 imC = d-j*dy;
 // z_0 = 0 = (reZ, imZ)
 reZ = 0;
 imZ = 0;
 iteráció = 0;
 // z_{n+1} = z_n * z_n + c iterációk
 // számítása, amíg |z_n| < 2 vagy még
 // nem értük el a 255 iterációt, ha
 // viszont elértük, akkor úgy vesszük,
 // hogy a kiindulási c komplex számra
 // az iteráció konvergens, azaz a c a
 // Mandelbrot halmaz eleme
 while(reZ*reZ + imZ*imZ < 4 && iteráció < 255) {
 // z_{n+1} = z_n * z_n + c
 ujreZ = reZ*reZ - imZ*imZ + reC;
 ujimZ = 2*reZ*imZ + imC;

 // az iteráció (reZ, imZ) -> (ujreZ, ujimZ)
 // ezt az egyenest kell kirajzolnunk, de most
```

```
// a komplex számokat vissza kell transzformálnunk
// a rács oszlop, sor koordinátájává:
java.awt.Graphics g = kép.getGraphics();
g.setColor(java.awt.Color.WHITE);
g.drawLine(
 (int)((reZ - a)/dx),
 (int)((d - imZ)/dy),
 (int)((ujreZ - a)/dx),
 (int)((d - ujimZ)/dy)
);
g.dispose();
nagyító.repaint();

reZ = ujreZ;
imZ = ujimZ;

++iteráció;
// Várakozunk, hogy közben csodálhassuk az iteráció
// látványát:
try {
 Thread.sleep(várakozás);
} catch (InterruptedException e) {}
}
}
```

A MandelbrotHalmaz publikus osztály így minden más osztály számára elérhető mindenütt, az `extends java.awt.Frame` azt jelenti hogy a `java.awt.Frame` osztály elemei öröklődnek a MandelbrotHalmaz számára, az `implements Runnable` pedig azt jelenti hogy a `Runnable` függvényeihez hozzáferhetünk a MandelbrotHalmaz osztályban. A `Runnable` egy interfész, az interfészek olyan osztályok amelyek függvényeinek nincs parametere. A jelen esetben használt `Runnable` interfész segítségével hozunk létre egy szálát és definiáljuk benne a `run` függvényt. A védett típusú változókat az adott osztályhoz tartozó osztályok és az azonos csomagban lévő osztályok számára elérhető, a magán típusú pedig csak az adott osztály, ha pedig nincs megadva akkor az osztállyal azonos csomagban lévő osztályok érhetik el. Ezek segítségével deklaráljuk illetve inicializáljuk a szükséges változókat és a `java.awt.image.BufferedImage` kép; kóddal deklarálunk egy kép nevű objektumot. Javában minden osztályban van így van legalább egy konstruktor ami inicializálja majd az új objektumokat és a nevenek ugyanakkor kell lennie mint az osztálynak, jelen esetben létrehozuk a MandelbrotHalmaz konstruktorát a megfelelő paraméterekkel. A konstruktorok a `new` operátor hatására hívódnak meg amikor objektumot hozunk létre az osztályhoz. Az objektumok eléréséhez tarolnunk kell a memóriacímüket egy változóban, jelen esetben a `kep` változóban, amelyre mutató objektummal létrehozunk egy szélesség széles, magasság magas és egész típusú RGB szinkodolható képet. A `this.nev` kódot használva az adott függvény és változó nevek értékeit elérhetik az objektumok, ezeket a konstruktor elso sorától kell definiálni. Az `addWindowListener(new java.awt.event.WindowAdapter() {public void windowClosing(java.awt.event.WindowEvent e) {setVisible(false);System.exit(0);}})` kóddal beállítjuk hogy ha bezárjuk az ablakot álljon le a program. Az `addKeyListener` segítségével definiáljuk hogyha a S billentyűt lenyomva meghívódik a pillanatfelvétel függvény, ha pedig a N billentyűt lenyomva akkor az `iteraciosHatar`-t növeljük meg 256-tal és új szálát hozunk létre és a `start` függvénnyel hívjuk rá a `run`



fuggvenyt azaz kiszámoljuk rá a Mandelbrot halmazt. Az `M` billentyű lenyomasakor is először ellenőrizzük hogy a `futSzamitas` hamis-e, ha nem akkor nem csinál semmit a program a billentyű lenyomasakor, itt azonban az `iteraciosHatar`-t  $10 \cdot 256$ -tal növeljük és így számoljuk újra a Mandelbrot halmazt. A `new Thread(this).start();` kóddal létrehozunk egy szálát és az aktuális objektumra hívjuk a `run` függvényt. A `set`-tel kezdődő konstruktorokkal beállítjuk az ablak címet, méretét, lathatóságát, meretezhetőseget. A `this` szót használjuk meg megkelevesztályok változainak elérésére is, jelen esetben a `MandelbrotHalmaz` `iteraciosHatar` változójánál. A `this` az aktuális osztály objektumára hivatkozik, de megadva a `.` előtt az osztálynevet akkor a `.` előtti osztály objektumára hivatkozik. Létrehozzuk a `paint` függvényt amelyben létrehozzuk a `Graphics` osztály egy `g` objektumát, melynek a `drawImage` függvényével kirajzoljuk a kép objektumot, a kép bal felső sarkának  $(x,y)$  koordinátái  $(0,0)$ , az aktuális objektummal pedig frissítjük a képet. Ha pedig a `szamitasFut` logikai típusú változó értéke igaz akkor beállítjuk az objektum színét pirosra és a `drawLine` függvény segítségével kirajzolunk a sor névű változó sorába úgy  $0$ -tól a sor végéig amit a `getWidth()` függvény határoz meg. Létrehozzuk az `update` függvényt melynek paramtere a `Graphics` osztály `g` objektuma és meghívjuk a létrehozott `g` objektumra a `paint` függvényt. A pillanatfelvétel függvénnyel létrehozunk egy egész RGB szinkodolással készült képet ami szélesség széles és magasság magas és a mentkepi címet a mentkepi változóba mentjük. A `getGraphics` konstruktorral létrehozunk egy grafikai környezetet a mentkepi részére és mentjük a `g` objektumba. A `drawImage` függvénnyel megrajzoljuk a kép nevű képet a  $(0,0)$  bal felső koordinátától, az aktuális objektummal frissítve, majd beállítjuk a színt a `setColor` függvény segítségével. Az adott színre pedig kirajolja az első paraméterként adott szívet a második és harmadik paraméter szerint amik az  $x$  és  $y$  koordinátái a grafikai ablaknak. A `dispose` függvénnyel pedig bezárjuk az aktuális ablakot azaz a mentkepi képet. A `StringBuffer` konstruktorral létrehozunk egy `sb` nevű objektumot, ezt akkor használjuk ha megkeveszt szöveget változtatunk. Először a `delete` függvénnyel kitoroljuk a `sztring-buffer` tartalmát  $0$ -tól a `sztring` hosszáig. Az `append` függvénnyel a paraméterül adott `sztringet` hozzafűzzük a `buffer` tartalmához. A `javax.imageio.ImageIO.write` függvény segítségével a mentkepi fájlt kiírjuk egy `png` típusú fájlba a készített `sztring` néven, ami a `java.io.File` konstruktor által létrehozott objektum. A fájl nevét az `sb.String()` függvény határozza meg mely kiírja `sztring`ként az `sb` `buffer` tartalmát. A fájl létrehozásához hibakezelést használtunk hogy tudjuk ha ez a hiba. A `run` függvényben kiszámítjuk a Mandelbrot halmazt, az iteráció alapján megadjuk az `rgb` változó értékeit, a számításához használjuk a bal shift operátort, ami a balra ketteszeresítéssel  $n$  esetben  $n$ -el shiftel bittel shiftel balra binárisan. A `iteráció \% 256`; miatt úgy  $0$  és  $255$  közé esik. A `setRGB` függvénnyel beállítjuk a kép objektum pixelszínét pixelenként, magasságonként pedig újrafestjük, majd mielőtt kilep a függvényből a `szamitasFut` változót `false`-ra állítjuk. Létrehozzuk függvényeket melyek visszadják a megfelelő változókat értékek. A `public static void main(String[] args)` a `java` main programja amely lefut először ha futtatjuk a programot ebben létrehozunk egy objektumot a `MandelbrotHalmaz`-hoz a konstruktorát használva, ami úgy  $600 \times 600$ -as képet hoz létre és a számítás a  $[-2.0, .7] \times [-1.35, 1.35]$  halmazon végzi  $255$ -ös `iteraciosHatar`al. A `MandelbrotHalmazNagyito.java` fájlban létrehozzuk a `MandelbrotHalmazNagyito` osztályt melynek osztoztálya lesz a `MandelbrotHalmaz`, úgy  $0$  a fájlnevek egyeznie kell az osztálynevel. Deklaráljuk a nagyításhoz szükséges változókat és a `super` szóval meghívjuk az osztoztály azaz a `MandelbrotHalmaz` konstruktorát a megfelelő paraméterekkel, amely lenyegeben lefutatta a `MandelbrotHalmaz.java` program a `MandelbrotHalmaz` osztályból áll. Létrehozzuk egy `m` objektumot a `MouseEvent` osztályból ebbe mentjük az  $x$  és  $y$  koordinátáit a kattintásunknak ha ez balegergőgombbal történt, ha jobb eger gombbal kattintunk akkor létrehozunk egy `MandelbrotIteracio` konstruktort, és kiszámítjuk az adott pontból indított iterációkat. A `mouseDragged` osztályt felhasználva kiszámítjuk a kijelölt területre az  $m_x$  és  $m_y$  értékeit, amik jelen esetben téglalap vagy négyzet oldalai lesznek. A `MouseReleased` osztályt felhasználva a megfelelő változókat módosításával kiszámoljuk a kijelölt terület Mandelbrot halmazát teljes kép objektum teljes méretére. A pillanatkepi függvényt kiegészítjük azzal hogy a kijelölt alakzat oldalvonalait zoldra festve megjelöljük, ugyanígy kirajzoljuk a `paint` függvényben is. A `MandelbrotIteraciok.java` fájlban létrehozzuk a `MandelbrotIteraciok` osztályt amelynek

interfesze a Runnable így használhatjuk majd az általunk definált run függvenyt. A MandelbrotIteraciok osztályban defináljuk a megfelelő változókat és két objektum mutató változót is. A MandelbrotIteraciok konstruktorát létrehozzuk melynek parametere a MandelbrotHalmazNagyito objektumának mutató változóját nagyito névvel és egy egész típusú változót varakozas néven, defináljuk benne a változók értékadását. A run függvényben nem használunk for ciklust mivel csak ahhoz a ponthoz számoljuk ki a Mandelbrot halmazt ahova jobb eger gombbal kattintottunk. Mivel ennek a pontnak az x y koordinátáit értékel adtuk a k j változóknak így a  $reC$   $imC$ -t ezek alapján számítjuk ki, ami az a komplex szám lesz ahonnan az iterációt kezdjük. Az iterációkat addig számoljuk amíg a while ciklus mindeket feltétele igaz. Minden újabb kiszámolt komplex számot a valós és képzetes részeinek koordinátáit összekötő fehér vonallal ábrázolunk. Majd az egyes komplex számok kirajzoltatása után a szálak végrehajtását szüneteltetjük adott milliszekundumig a sleep függvény segítségével. A sleep függvényt a InterruptedException hibakezeléssel használjuk.

## 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 elspájjolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltérő kiszámolt szám.

```
//polargen.cpp
#include <cstdlib>
#include <cmath>
#include <ctime>
#include <iostream>
class PolarGen
{
public:
 PolarGen ()
 {
 nincsTarolt = true;
 std::srand (std::time (NULL));
 }
 ~PolarGen ()
 {
 }
 double 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;
}
}

private:
 bool nincsTarolt;
 double tarolt;
};

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

 for (int i = 0; i < 10; ++i)
 std::cout << pg.kovetkezo () << std::endl;

 return 0;
}
```

```
//PolarGenerator.java
public class PolarGenerator {

 boolean nincsTarolt = true;
 double tarolt;

 public PolarGenerator() {

 nincsTarolt = true;

 }

 public double kovetkezo() {

 if(nincsTarolt) {
```

```
double u1, u2, v1, v2, w;
do {
 u1 = Math.random();
 u2 = Math.random();

 v1 = 2*u1 - 1;
 v2 = 2*u2 - 1;

 w = v1*v1 + v2*v2;

} while(w > 1);

double r = Math.sqrt((-2*Math.log(w))/w);

tarolt = r*v2;
nincsTarolt = !nincsTarolt;

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

public static void main(String[] args) {

 PolarGenerator g = new PolarGenerator();

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

A polargen.cpp fájlban a program elején definíaltuk a PolarGen osztályt, melynek először a private része definiálódik ahol deklaráltuk a nincsTarolt változót bool típusúval tehát logikai értéket vehet fel úgy igazat vagy hamisat és a tarolt double típusú változót úgyé lebegőpontos szám lesz az értéke majd. A public részben definiáltuk az osztály konstruktorát aminek a nevének meg kell egyeznie az osztály nevével, inicializáljuk benne a nincsTarolt változót true értékre és az std::srand(std::time(NULL)); kóddal beállítjuk hogy a rand függvény minden program futtatáskor más eredményeket adjon, mivel a time(NULL) az aktuális időt adja eredményül. Ezek használatához szükséges include-olni a ctime.h és cstdlib.h fájlokat. Letrehozzuk az osztály destruktort is amely törli az objektumot és az osztályt, a destruktort nevének meg kell egyeznie az osztály nevével és ~ jelöljük hogy destruktort a név előtt. Egy osztálynak több konstruktora lehet, de csak egy destruktora. A következő függvényt definiáljuk double visszatérési értékkel melyben definiálunk egy if feltételt melyben ha a nincsTarolt változó értéke igaz akkor akkor a polartranszformációs algoritmus hajtodik végre, a rand függvény által generált a RAND\_MAX értéke ami minimum 32767. A rand

fuggvény random számokat general, az sqrt a gyokvonast vegzi, a log pedig a logaritmus fuggvény. Az v1 es v2 ertekeket addig számoljuk a while ciklus do utasitas reszeben amig a negyzetosszeguk nagyobb mint 1, a v1 es v2 ertekeket az u1 es u2 valtozok ketszeresebol kivonva egyet számítjuk ki. Az r valtozoba kiszámoljuk a w valtozo segitsegevel aminek az erteke 0 es 1 kozotti szám aminek a logaritmusa negativ így negativ számmal kell megszoroznunk gyokvonas előtt mivel valos számoknál csak pozitiv gyokvonast értelmezzük. A v1 valtozo es az r szorzatat adjuk visszateresi ertekul a fuggvénynek, a másik valtozot mentjük a tarolt valtozoba hasonloan es a nincsTarolt valtozo erteke ellenkezojere allitjuk a ! jellel azaz igazrol hamisra, mivel ha az if feltetelben a nincsTarolt nem igaz erteke azaz hamis erteke eseten lep az else agba ahol a visszateresi erteke a tarolt valtozo es atallitjuk igazra a nincsTarolt valtozo erteke. Az int main fuggvényben a PolarGen osztalyhoz létrehozunk egy objektum tipusu valtozot pg neven. Ekkor a konstruktor meghivodik es létrejön a qp objektum. A for ciklussal 10-szer kiíratjuk a szabvanyos kimenetre a pq objektumra a kovetkezo fuggvény visszateresi erteke. Mivel a tarolt valtozoba mentjük minden masodikat így feleannyiszor kell használni a generalasos részt. A cmath.h fajlt a log fuggvényhez használjuk es az iostream.h header pedig a szabvanyos bemeneti es kimeneti fuggvények hasznalatahoz szukseges, jelen esetben a cout-hoz. A PolgarGenerator.java programban a random fuggvény a Maths osztaly fuggvényeit használjuk a számításokhoz az sqrt es log fuggvényeket es a random fuggvényt. A random fuggvény visszateresi erteke egy 0 es 1 kozotti számot az 1 kivetelevel. Itt nem biztosítjuk hogy ujrafuttataskor nem jön e ki majd ugyanaz a random szám. Itt nem használunk private részt, es a nincsTarolt valtozo erteke deklarációjakor megadjuk. A konstruktor hasznalatahoz a new operátorral létre kell hoznunk egyet melynek erteke a g objektum tipusu valtozoba mentjük. A System.out.println(g.kovetkezo()) fuggvénnyel kiíratjuk az ugye OlarGenerator osztaly g objektumához tartozo fuggvény ertekeit 10-szer.

Megoldás videó:

Megoldás forrása:

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

## 6.2. LZW

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

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

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

BINFA_PTR
uj_elem ()
{
```

```
BINF_PTR p;

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

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

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

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

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

```

```
 fa = fa->jobb_egy;
 }
}

printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d", max_melyseg);
szabadit (gyoker);
}

static int melyseg = 0;
int max_melyseg = 0;

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

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

A strucut binfa koddal létrehozunk egy binfa nevű struktúrát, a typedef szóval nem lesz szükséges a struct szó használata a későbbiekben az adott osztály típusú változók létrehozásakor. A binfa struktúrában deklarálunk egy egész típusú változót érték néven, a struktúrán belül szükséges a struct szót használni mutatók létrehozásakor, jelen esetben létrehozunk két binfa struktúrára mutató mutatót bal\_nulla és jobb\_egy néven.



A struktúra definícióját végén deklarálhatunk struktúra típusú változókat, jelen esetben a BINFA változót és a \*BINFA\_PTR struktúrára mutató változót deklaráljuk a binfa végén. Mivel typedef típussal hoztuk létre a struktúrát így nem lesz látható a későbbiekben tehát a BINFA\_PTR változót használjuk binfa típus definíciójához. Létrehozzuk a BINFA\_PTR struktúra típussal az uj\_elem függvényt melyben létrehozzuk a p binfa típusú változót és if feltételben malloc segítségével foglalunk memóriát sizeof(BINFA) azaz binfa struktúra típusú BINFA változó méreténnyit, ami 24 bajt. A malloc jelen esetben típuskenyszerítés nélkül is 24 bajt memóriát foglal. Ha sikeres a memóriafoglalás akkor a p változóba mentjük a memóriacímét a foglalt memóriaterületnek, mivel ekkor a memóriacímét adjuk értékel a malloc. Ha nem tudunk memóriát foglalni a malloc visszatérési értéke NULL tehát teljesül az if feltétel akkor a perror ("memoria"); kód kiírja a szabványos kimenetre a memória sztringgel a hibauzenetet és exit (EXIT\_FAILURE); kóddal pedig kilep a program. Az uj\_elem függvény visszatérési értéke pedig a p változó értéke ami ugye a lefoglalt memória címe. Az extern típusú függvények lathatók és újradefiniálhatóak, két ilyen függvényt deklarálunk így kiír és a szabadít neven void típussal tehát nincs visszatérési értékük, paramétereik egy binfa típusú változó lesz, mivel a BINFA\_PTR a binfa struktúra mutatója és az elem pedig BINFA\_PTR típusú változó. Az int main függvényben deklaráljuk a b változót char típussal, ami 1 bajt memóriát használ. Deklaráljuk a gyoker változót ami mutató típusú változó lesz mivel a binfa pointeret adtuk típusának és inicializáljuk az uj\_elem függvénnyel aminek ugye a visszatérési értéke a lesz a gyoker értéke. A struktúra értéket ha struktúra típusú mutatóval akarjuk elérni a -> operátort kell használnunk. Így a gyoker->érték kóddal tudjuk értékel adni az érték változónak '/' értéket amire ugye a gyoker változó mutat. A fa változót hasonlóan deklaráljuk a gyoker változóhoz és inicializáljuk a gyoker változó értékével. A while feltételben a read függvény van amivel a szabványos bemenetről beolvassunk 1 bajtot és mentjük a b változó void típusú memóriacímével ugye a b változóba. Mivel a read függvény visszatérési értéke a beolvasott bajtok száma vagy ha nincs több beolvasandó bajt akkor 0, így a while ciklus addig fut amíg van beolvasandó bajt ugye a while(1) és más nagyobb természetes számok végtelen ciklust hoznak létre, a while(0) pedig kilep. A write függvény segítségével a szabványos kimenetre írunk 1 bajtot, a b memóriacímét felhasználva a b változó értéket. Ha a beolvasott érték 0 akkor fa->bal\_nulla == NULL feltétel teljesül tehát nem létezik még a fa->bal\_nulla mutató típusú változó akkor foglalunk neki helyet és értékel adjuk a 0-t majd létrehozzuk a fa->bal\_nulla mutató típusú változóba bal\_nulla és jobb\_nulla ágat NULL értékekkel és a fa mutató típusú változót a gyoker mutatóra állítjuk. Ha a fa->bal\_nulla változó értéke ugye már 0 akkor a fa mutatót reálítjuk. A jobboldali ágaknál ugyanígy van csak mutatókat fordítva használjuk és az értéket is. A printf ("\n"); kóddal kihagyunk egy sort, majd a kiír függvényt hívjuk a gyoker változóra és újradeklaráljuk a main függvényen belül is a max\_elyseg változót egész típusúra az extern kifejezést használva így az utóljára adott értéket kapjuk a változó értéket, nem a 0-t. Majd a printf függvény segítségével kiírjuk az értéket a melyseg=sztring után. Majd a szabadít függvényt hívjuk a gyoker változóra. Deklaráljuk és inicializáljuk a melyseg és max\_melyseg globális változókat 0 értékekkel és a melyseg változó statikus típusú tehát a memóriába van a program futása alatt. A kiír függvény void típusú így nincs visszatérési értéke, ha a paraméterként adott BINFA\_PTR típusú mutató változó amit elemmel neveztünk el NULL értékű akkor veget ér a program. Ha nem NULL értékű akkor növeljük a melyseg számát eggyel és egy if feltétel segítségével keressük a max\_melyseg értéket, majd hívjuk a kiír függvényt a bal\_nulla változóra. A for ciklus segítségével kiírunk annyi --- vonalat amennyi a melyseg a printf függvény segítségével. A másik printf függvény tartalmaz egy kondíciós feltételt melynek ha a ? előtti rész igaz akkor a : előtti rész, ha hamis akkor a : utáni rész íródik ki a melyseg értéke előtt. Majd a kiír függvényt hívjuk a elem->jobb\_ely változóra majd a melyseg értéket csökkentjük eggyel. Az első if feltétel nyilván a érték változó miatt szükséges mert előbb utóbb NULL lesz a jobb\_ely és a bal\_nulla esetében is. Hasonlóképpen van a szabadít void típusú globális függvény feltétele is, itt felszabadítjuk a paraméterként adott változó által foglalt helyet a free függvénnyel majd minden jobb\_ely és bal\_nulla eseten is. A program 0 és nem 0 bemeneti értékeket kezel. Mivel egymásba ágyazottan vannak a kiír függvények így az if feltételnek az ágak kialakításához szükséges.

Megoldás videó:

Megoldás forrása:

## 6.3. Fabejárás

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

```
//preorder.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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

BINFA_PTR
uj_elem ()
{
 BINFA_PTR p;

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

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

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

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

 while (read (0, (void *) &b, 1))
 {
```

```
 write (1, &b, 1);
 if (b == '0')
 {
 if (fa->bal_nulla == NULL)
 {
 fa->bal_nulla = uj_elem ();
 fa->bal_nulla->ertek = 0;
 fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
 fa = gyoker;
 }
 else
 {
 fa = fa->bal_nulla;
 }
 }
 else
 {
 if (fa->jobb_egy == NULL)
 {
 fa->jobb_egy = uj_elem ();
 fa->jobb_egy->ertek = 1;
 fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
 fa = gyoker;
 }
 else
 {
 fa = fa->jobb_egy;
 }
 }
}

printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d", max_melyseg);
szabadit (gyoker);
}

static int melyseg = 0;
int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
{
 if (elem != NULL)
 {
 ++melyseg;
 if (melyseg > max_melyseg)
 max_melyseg = melyseg;
 for (int i = 0; i < melyseg; ++i)
```

```
 printf ("---");
 printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
 ,
 melyseg);
 kiir (elem->bal_nulla);
 kiir (elem->jobb_egy);
 --melyseg;
}
}

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

```
//posztorder.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

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

BINFA_PTR
uj_elem ()
{
 BINFA_PTR p;

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

extern void kiir (BINFA_PTR elem);
```

```
extern void szabadit (BINFA_PTR elem);

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

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

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

 printf ("\n");
 kiir (gyoker);
 extern int max_melyseg;
 printf ("melyseg=%d", max_melyseg);
 szabadit (gyoker);
}
```

```
static int melyseg = 0;
int max_melyseg = 0;

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

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

Az inorder bejarasnal eloszor a binaris fa legbaloldalibb reszfajanak gyokerenek baloldali erteke majd a jobboldalit vesszük. Ezt rekurziván hajtjuk végre tehát minden csomópont esetén így járunk el, eloszor a baloldali elemet majd a csomópontot majd a jobboldali elemet dolgozzuk fel. Beolvasáskor csomópontokat hozunk létre amiknek van értéke bal és jobboldali mutatójuk, létrehozunk egy csomópontot gyöker néven ekkor úgy / lesz az értéke és a két mutató értéke 0. Ha a beolvasott érték 0 akkor megnezzük van-e az aktuális csomópontnak baloldali mutatójának értéke, ha van akkor beállítjuk a mutató érték által mutatott csomópontot aktuális csomópontnak, ezzel beolvastunk egy 0-t így tomorítódik a fájl. Ha pedig nincs baloldali csomópontja akkor létrehozunk egyet melynek értéke 0 és beállítjuk a létrehozott csomópont mutatóját az aktuális csomópont baloldali mutatójának majd az aktuális csomópont a gyöker lesz. 1 beolvasás esetén hasonlóan hajtódik végre. Az hogy milyen bejárás a fa a kiír függvény határozza meg hogy milyen sorrendben használjuk a következőket: a baloldali részre a kiír függvényt, a jobboldali részre a kiír függvényt, az aktuális csomópont értékének kiírását. Ha már az aktuális csomópont értéke null azaz nincs bal vagy úgy jobboldali mutatója nem hívódik újra a kiír függvény hanem az azelőtt meghívott kiír függvények folytatodnak egyenként míg le nem fut egy. A beagyazódás addig folytatódik így amíg úgy baloldalra

hívva baloldali újabb csomópont letezik, jobboldali eseten jobboldali csomópont. Az posztorder rednezesnel addig megyünk a baloldali agakon lefele amig van újabb baloldali ag, ha nincs akkor az utolsora megnezzuk van e jobboldali aga, ha nincs kiiratjuk a nullat majd visszafele haladva ugye marcsak a jobboldali agakat kell ellenoriznunk. A jobboldalinak lehet azonban újabb bal es jobboldali aga. Tehat vagy gyokerbol legbaloldalibb agig ahol ellenorizzuk a jobboldali agat ha van kiiratjuk a csomópont erteket es ralepunk az ellenorzott agra ekkor ismet ahogy a gyokertol, vagy ha nincs kiirjuk a csomópont erteket es visszalepunk egyet ekkor pedig ismet a legbaloldalibb eleresetol. Preorder eseten kiiratjuk a gyoker erteket majd a baloldali elemeket ameddig vannak, majd ellenorizzuk van e jobboldali elem ha van ismet a baloldali elemek kiiratas ha nincs akkor visszalepunk az elozi csomópontira ahol ismet ellenorzunk. Mindharom bejarasnal a jobboldali elem elott van a baloldali elem az algoritmusaban így a kiiratas lehet elotte kozotte mogotte mivel 3 bejarasi forma van a preorder inorder postorder.

Megoldás videó:

Megoldás forrása:

## 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!

```
#include <iostream>
#include <fstream>

class LZWBinFa
{
public:

 LZWBinFa ():fa (&gyoker)
 {
 }
 ~LZWBinFa ()
 {
 szabadit (gyoker.egyGyermek ());
 szabadit (gyoker.nullasGyermek ());
 }

 void operator<< (char b)
 {

 if (b == '0')
 {

 if (!fa->nullasGyermek ())
 {
 Csomopont *uj = new Csomopont ('0');
 fa->ujNullasGyermek (uj);
 fa = &gyoker;
 }
 }
 }
};
```

```
 }
 else
 {

 fa = fa->nullasGyermekek ();

 }
 }

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

}

void kiir (void)
{
 melyseg = 0;
 kiir (&gyoker, std::cout);
}

int getMelyseg (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);
}

private:
class Csomopont
{
```



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

 Csomopont *nullasGyermekek () const
 {
 return balNulla;
 }

 Csomopont *egyenesGyermekek () const
 {
 return jobbEgy;
 }

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

 void ujEgyenesGyermekek (Csomopont * gy)
 {
 jobbEgy = gy;
 }
 char getBetu () const
 {
 return betu;
 }

private:
 char betu;
 Csomopont *balNulla;
 Csomopont *jobbEgy;
 Csomopont (const Csomopont &); //másoló konstruktor
 Csomopont & operator= (const Csomopont &);
};

Csomopont *fa;

int melyseg;

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

void kiir (Csomopont * elem, std::ostream & os)
```

```
{

 if (elem != NULL)
 {
 ++melyseg;
 kiir (elem->egyenesGyermek (), os);
 for (int i = 0; i < melyseg; ++i)
 os << "---";
 os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
 kiir (elem->nullasGyermek (), os);
 --melyseg;
 }
}

void szabadit (Csomopont * elem)
{

 if (elem != NULL)
 {
 szabadit (elem->egyenesGyermek ());
 szabadit (elem->nullasGyermek ());
 delete elem;
 }
}

protected:
 Csomopont gyoker;
 int maxMelyseg;

 void rmelyseg (Csomopont * elem);
};

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

void
LZWBInFa::rmelyseg (Csomopont * elem)
{
 if (elem != NULL)
 {
 ++melyseg;
 if (melyseg > maxMelyseg)
 maxMelyseg = melyseg;
 rmelyseg (elem->egyenesGyermek ());
 // ez a postorder bejáráshoz képest
 // 1-el nagyobb mélység, ezért -1
 }
}
```

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

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

int
main (int argc, char *argv[])
{
 if (argc != 2)
 {
 usage ();
 return -1;
 }

 char *inFile = *++argv;

 std::fstream beFile (inFile, std::ios_base::in);
 if (!beFile)
 {
 std::cout << inFile << " nem letezik..." << std::endl;
 usage ();
 return -2;
 }

 unsigned char b;
 LZWBinFa binFa;

 while (beFile.read ((char *) &b, sizeof (unsigned char)))
 {
 if (b=='0')
 {
 binFa << b;
 }
 else if (b=='1')
 {
 binFa << b;
 }
 }

 std::cout<<binFa;
 std::cout<< "depth = " << binFa.getMelyseg () << std::endl;
```

```
beFile.close ();

return 0;
}
```

Tutoraltam Nagy Laszlo Mihalyt.

Letrehozzuk az LZWBInfa osztályt, mivel az osztályon kívül is használni fogjuk a konstruktort objektum létrehozására így az osztály public részébe definiáljuk az osztály konstruktort és destruktort. Az alapértelmezett osztályrész a private, ami nem elérhető csak osztályon belül vagy barát osztályok számára. Az osztály referenciaváltozóit a konstruktor inicializációs listájában inicializálhatjuk, jelen esetben a fa változó értéke a gyöker változó memóriacímre lesz. Letrehozzuk a destruktort is melyben a szabadt fuggvenyt hívjuk a gyöker objektum egyesGyerek és nullasGyerek fuggvényeire. Az operator fuggvény segítségével tulterhelhetjük az operatorokat jelen esetben a duplakisebbkacsacsor operátort. Void típusként használjuk az operator fuggvenyt így visszatérési érték nélkül és char b paraméterrel, azaz egy bajtot olvas be a szabványos kimenetből. Ha a beolvasott bajt 0 akkor ha a fa változó nem mutat a nullasGyerek fuggvényre akkor az Csomopont konstruktor segítségével és a new paranccsal létrehozunk egy új objektumot melynek értéke 0 lesz és memóriacímét az új változóba mentjük. A fa változóval által mutatott újNullasGyermek fuggvenyt hívjuk új változó paraméterrel, majd a fa változó értéket visszaállítjuk a gyöker változó memóriacímére. A feltétel különben agaban pedig a fa változó értéket a fa változó által mutatott egyesGyermek fuggvény visszatérési értéke. Az eredeti if feltétel else ágat ehhez hasonlóképpen definiáljuk csak a jobboldali ághoz tartozó fuggvény és változó nevekkal. Definiáljuk a kiír fuggvenyt visszatérési érték és void paraméterrel, tehát nem adhatunk paramétert neki, a melység változó értéket 0-ra állítja és meghívja a kiír fuggvenyt két paraméterrel a gyöker változó memóriacímével és a cout fuggvénnyel. Definiáljuk a geMelység fuggvenyt egész típusú visszatérési értékkel a fuggvénynek nem adhatunk paramétert mivel void paraméter típusú. Egy adott osztályban definiálunk friend típussal fuggvényeket másik osztályokból vagy másik osztályokat akkor a másik osztály fuggvényei és a friend típussal definiált fuggvények hozzáférhetnek az adott osztály private részének elemeihez, jelen esetben az std kimeneti fuggvények és az duplakisebbkacsacsor operátor fuggvény hozzáférhet az LZWBInfa privat részéhez. Az std::ostream end és egy ostream osztálybeli objektumot os-t jelöl, az LZWBInfa end és bf pedig egy LZWBInfa belső objektumot jelöl a jelen esetben a bf-et. Ezek voltak a tulterhelt duplakisebbkacsacsor operátor fuggvény paraméterei melyben egy bf objektumban definiált kiír fuggvenyt hívunk os objektum paraméterrel és a visszatérési értéke az os objektum. A kiír fuggvenyt definiáljuk paraméteresen a paraméter egy ostream-beli objektum lesz jelen esetben az os és visszatérési értéke nincs, a melység változó értéket 0-ra állítjuk és meghívjuk a két paraméteres kiír fuggvenyt a gyöker változó memóriacímére és a megadott os objektumra. A privat részben definiáljuk a Csomopont osztályt melynek publikus részében definiáljuk a parametizált konstruktort melynek paramétere egy char típusú változó / kezdőértékkel amelyet jelen esetben b-nek nevezünk. A konstruktorhoz tartozó inicializált lista segítségével tudunk kezdőértéket adni az osztály privat részben deklarált változóinak, a betű kezdőértéke a b változó értéke a balNulla és a jobbEgy változók pedig 0 azaz NULL értéket kapnak. Letrehozzuk a destruktort is az osztálynak. Definiáljuk a nullasGyermek fuggvenyt Csomopont osztály mutató típusú fuggvenyt mivel const tagu fuggvény így nem változtathat a változók értékein, értéket adja a 0-t egy mutatóval ha van az aktuális csomópontnak. A visszatérési érték nélküli újNullasGyermek fuggvény a paraméterül kapott Csomopont osztály mutató típusú gy változó értéket értéket adja a balNulla változónak. Az egyesGyermek és az újEgyesGyermek fuggvények hasonlóan működnek csak más változónevekkel. A getBetű fuggvény char visszatérési értékű paramétermentes const típussal és visszatér a betű változó értéket. A privat részben deklaráljuk a betű változót char típussal és a balNulla és jobbEgy Csomopont mutató típusú változókat. Majd létrehozunk egy masoló konstruktort és egy egy segéd konstruktort. Letrehozzuk

a fa változat Csomopont mutató típusú, deklaráljuk int típusú a mélység, átlagösszeg, átlagdb változatokat és double típusú a szorzóösszeg változatot. Letrehozunk az LZWBinfa osztályhoz is egy másoló és egy segédkonstruktor. Defináljuk a kiír függvényt két paraméterrel az első paraméter egy Csomopont mutató típusú elem névű változó a második az ostream osztály os objektuma. A függvény segítségével kiírjuk a binfát in-order módon. A szabadít függvénnyel töröljük a paraméterül adott mutató típusú változó egyes majd nullas gyermekeinek memóriacímét majd a változó által foglalt memóriát is felszabadítjuk a delete függvény segítségével. A védett részben letrehozunk egy Csomopont osztálybeli objektumot gyöker néven és deklaráljuk int típusú a maxMélyseg változatot és double típusú a szorzó és átlag változatokat. Defináljuk void típusú a rmélyseg függvényt Csomopont mutató típusú elem névű változó paraméterrel. A usage függvény segítségével hibajelzést írunk ki a szabványos kimenetre, pontosabban használati utasítást. Az int main függvényben vizsgáljuk hogyha a parancssori argumentumok száma nem kettő akkor hívjuk a usage függvényt és kilépünk a main-ból. A bemeneti fájl mutatóját inicializáljuk az első parancssori argumentum értékével. Az ifstream osztály segítségével a beFile objektumot beállítjuk bemenetnek az inFile változó segítségével. Ha nincs beFile akkor hibajelzést írunk ki hogy nem létezik... és kilépünk a programból. Deklaráljuk a b változat unsigned char típusú így a mérete egy bajt és az értékei csak pozitívak lehetnek 0 és 255 közt. Letrehozzuk az LZWBinfa osztály binFa névű objektumát és a kommentben bool típusú változat false értékkel inicializáljuk. A read függvény segítségével while ciklusban bajtonként olvasunk a bemeneti fájlból a b változóba amíg van beolvasandó bajt. A beolvasott bajtokat if feltétellel ellenőrizve eldöntjük 0 vagy 1 és ez alapján írjuk a binFa objektumba a b változatot. A depth= sztringet kiírjuk a cout függvénnyel és az értéke a binfa legnagyobb mélysége amit a getMélyseg függvény határoz meg az rmélyseg függvény segítségével.

Megoldás videó:

Megoldás forrása:

## 6.5. Mutató a gyökér

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

```
#include <iostream>
#include <fstream>

class LZWBinFa
{
public:

 LZWBinFa ()
 {
 fa = gyoker;
 }
 ~LZWBinFa ()
 {
 free(gyoker);
 }
}
```

```
void operator<< (char b)
{
 if (b == '0')
 {
 if (!fa->nullasGyermek ())
 {
 Csomopont *uj = new Csomopont ('0');
 fa->ujNullasGyermek (uj);
 fa = gyoker;
 }
 else
 {
 fa = fa->nullasGyermek ();
 }
 }
 else
 {
 if (!fa->egyenesGyermek ())
 {
 Csomopont *uj = new Csomopont ('1');
 fa->ujEgyenesGyermek (uj);
 fa = gyoker;
 }
 else
 {
 fa = fa->egyenesGyermek ();
 }
 }
}

void kiir (void)
{
 melyseg = 0;
 kiir (gyoker, std::cout);
}

int getMelyseg (void);
double getAtlag (void);
double getSzoras (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);
}

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

 Csomopont *nullasGyermekek () const
 {
 return balNulla;
 }

 Csomopont *egyenesGyermekek () const
 {
 return jobbEgy;
 }

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

 void ujEgyenesGyermekek (Csomopont * gy)
 {
 jobbEgy = gy;
 }
 char getBetu () const
 {
 return betu;
 }

private:
 char betu;
 Csomopont *balNulla;
 Csomopont *jobbEgy;
 Csomopont (const Csomopont &); //másoló konstruktor
 Csomopont & operator= (const Csomopont &);
};
```

```
Csomopont *fa;

int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;

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

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

void szabadit (Csomopont * elem)
{
 if (elem != NULL)
 {
 szabadit (elem->egyenesGyermeke ());
 szabadit (elem->nullasGyermeke ());
 delete elem;
 }
}

protected:
 Csomopont * gyoker = new Csomopont ();

 int maxMelyseg;

 void rmelyseg (Csomopont * elem);
};

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



```
void
LZWBinFa::rmelyseg (Csomopont * elem)
{
 if (elem != NULL)
 {
 ++melyseg;
 if (melyseg > maxMelyseg)
 maxMelyseg = melyseg;
 rmelyseg (elem->egyGyermek ());
 // ez a postorder bejáráshoz képest
 // 1-el nagyobb mélység, ezért -1
 rmelyseg (elem->nullasGyermek ());
 --melyseg;
 }
}

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

int
main (int argc, char *argv[])
{
 if (argc != 2)
 {
 usage ();
 return -1;
 }

 char *inFile = *++argv;

 std::fstream beFile (inFile, std::ios_base::in);
 if (!beFile)
 {
 std::cout << inFile << " nem letezik..." << std::endl;
 usage ();
 return -2;
 }

 unsigned char b;
 LZWBinFa binFa;

 while (beFile.read ((char *) &b, sizeof (unsigned char)))
 {
 if (b=='0' || b=='1')
 {

```

```
binFa << b;
}
}
std::cout<<binFa;
std::cout<< "depth = " << binFa.getMelyseg () << std::endl;

beFile.close ();

return 0;
}
```

A gyoker objektum nem lesz kompozicioban a fa mutato tipsusu valtozoval ami foglal egy Csomopont osztalybeli objektum meretu memoriat mivel most kulon hoztunk létre gyoker neven mutato tipusu valtozot így a gyoker objektum a fa valtozo mellett a gyoker valtozo ertekekent is a memoriaba van, így a gyoker objektum aggregacioban áll a fa valtozoval mivel nemcsak a fa valtozo határozza meg hogy meddig van a memoriaban a gyoker objektum. A Csomopont gyoker; helyett most Csomopont \* gyoker = new Csomopont(); a gyoker nem egy Csomopont tipusu objektum neve lesz hanem egy Csomopont meretnyi memoriara mutato valtozo mely megadja a memoriacimet a lefoglalt helynek és kezdőértéke egy Csomopont tipusu objektum. A fa mutato tipusu valtozonak értéket adjuk a gyoker valtozo értéket így a fa és a gyoker valtozo is ugyanarra az objektumra mutat ami ugye a memoriaba van. A new paranccsal foglalunk memoriát melynek a címet tarolnunk kell a használatához különben elvesz. A new által foglalt memoriát torolnunk kell a delete paranccsal melynek parametere a torolni kívánt memória címe ami jelen esetben a gyoker vagy a fa is lehet. Mivel memóriacímekkel dolgozunk a függvényekben így eddig az gyoker objektum memóriacímét az end paranccsal értük el, de mivel a most a gyoker a memóriacím az objektumnak így torolnunk kell az end jeleket. A gyoker objektumot a stack memoriába hoztuk létre így az osztály destruktora azt automatikusan törli, de a jelen esetben a gyoker valtozot torolne csak a destruktorki a heap-en foglalt memória címe így ezt külön torolnunk kell különben elvesz. A destruktorki tartalmát a szabadit(gyoker.egyGyermek) és szabadit(gyoker.nullasGyermek)-rol modositjuk szabadit(gyoker)-re mivel torolnunk kell a gyoker által foglaltat memoriát is így nem a szabadit(gyoker->egyGyermek) és szabadit(gyoker->>nullasGyermek)-re írjuk át mivel ez csak a gyoker által mutatott memóriacímeket torolne hanem a szabadit(gyoker)-re ami ugye törli a gyoker által foglalt memoriát is.

Megoldás videó:

Megoldás forrása:

## 6.6. Mozgató szemantika

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

```
#include <iostream>
#include <fstream>

class LZWBinFa
{
```

```
public:

 LZWBinFa ()
 {
 fa = gyoker;
 }
 ~LZWBinFa ()
 {
 szabadit(gyoker);
 }

 LZWBinFa (LZWBinFa&& masik)
 {
 gyoker = nullptr;
 *this = std::move(masik);
 }

 LZWBinFa& operator=(LZWBinFa&& masik)
 {
 std::swap(gyoker, masik.gyoker);
 return *this;
 }

 void operator<< (char b)
 {
 if (b == '0')
 {
 if (!fa->nullasGyermek ())
 {
 Csomopont *uj = new Csomopont ('0');
 fa->ujNullasGyermek (uj);
 fa = gyoker;
 }
 else
 {
 fa = fa->nullasGyermek ();
 }
 }
 else
 {
 if (!fa->egyenesGyermek ())
 {
 Csomopont *uj = new Csomopont ('1');
 fa->ujEgyenesGyermek (uj);
 fa = gyoker;
 }
 }
 }
}
```

```
 else
 {
 fa = fa->egyenesGyermek ();
 }
 }
}

void kiir (void)
{
 melyseg = 0;
 kiir (gyoker, std::cout);
}

int getMelyseg (void);
double getAtlag (void);
double getSzoras (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);
}

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

 Csomopont *nullasGyermek () const
 {
 return balNulla;
 }

 Csomopont *egyenesGyermek () const
 {
 return jobbEgy;
 }
}
```

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

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

char getBetu () const
{
 return betu;
}

private:
 char betu;
 Csomopont *balNulla;
 Csomopont *jobbEgy;
 Csomopont (const Csomopont &); //másoló konstruktor
 Csomopont & operator= (const Csomopont &);
};

Csomopont *fa;

int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;

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

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

void szabadit (Csomopont * elem)
{

```

```
 if (elem != NULL)
 {
 szabadit (elem->egyenesGyermek ());
 szabadit (elem->nullasGyermek ());
 delete elem;
 }
 }

protected:
 Csomopont * gyoker = new Csomopont();

 int maxMelyseg;

 void rmelyseg (Csomopont * elem);
};

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

void
LZWBInFa::rmelyseg (Csomopont * elem)
{
 if (elem != NULL)
 {
 ++melyseg;
 if (melyseg > maxMelyseg)
 maxMelyseg = melyseg;
 rmelyseg (elem->egyenesGyermek ());
 // ez a postorder bejáráshoz képest
 // 1-el nagyobb mélység, ezért -1
 rmelyseg (elem->nullasGyermek ());
 --melyseg;
 }
}

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

int
main (int argc, char *argv[])
{
 if (argc != 2)
 {
```

```
 usage ();
 return -1;
 }

 char *inFile = *++argv;

 std::fstream beFile (inFile, std::ios_base::in);
 if (!beFile)
 {
 std::cout << inFile << " nem letezik..." << std::endl;
 usage ();
 return -2;
 }

 unsigned char b;
 LZWBinFa binFa;

 while (beFile.read ((char *) &b, sizeof (unsigned char)))
 {
 if (b=='0' || b=='1')
 {
 binFa << b;
 }
 }
 std::cout<<binFa;
 std::cout<< "depth = " << binFa.getMelyseg () << std::endl;
 LZWBinFa binFa2 = std::move(binFa);

 std::cout << "\nEredeti fa mozgatás után:\n";
 std::cout << binFa;

 std::cout << "depth = " << binFa.getMelyseg () << std::endl;

 std::cout << "\nMozgatással létrejött fa:\n\n";
 std::cout << binFa2;

 std::cout << "depth = " << binFa2.getMelyseg () << std::endl;

 beFile.close ();

 return 0;
}
```

A `LZWBinFa binFa2 = std::move(binFa);` paranccsal létrehozunk egy `LZWBinfa` objektumot `binFa2` néven melynek értéket egy másik `LZWBinFa` osztálybeli objektummal tesszük egyenlővé így másoló konstruktor hívódna, de jelen esetben a `binFa` objektumot paraméterül adtuk az `std::move` függvénynek így jobbertekke alakítottuk át így a mozgató konstruktort hívja meg. A mozgató konstruktor definíciójakor a paraméterként

end end operatort használunk end helyett. A másik objektum jelen esetben a binFa objektum lesz, a this az aktuális objektum azaz a BinFa2. A gyoker változó az aktuális objektum azaz a binFa2 változója amit nullptr értékre állítunk majd a this\*-nak azaz az aktuális objektum memóriacímének érték adására a binFa objektumot parameterként felhasználva meghívjuk az értékado konstruktort melyben az std::swap függvény segítségével megcseréljük a két objektum gyoker változóinak memóriacímét, majd visszateresi értéként megadjuk az aktuális objektum memóriacímét. Az értékado konstruktor a mozgato konstruktoron belül hívódik meg. Így a BinFa2 objektum kiírásakor a gyoker változó hivasakor ugye már a BinFa objektumot használjuk így arról felel a fa, de a BinFa objektum kiírásakor a gyoker már nem mutat sehova így kilep a kiírás függvényből és nem rajzol ki semmit. Mivel a melyseget is a gyoker alapján számoljuk ott is hasnolnánk kilep mivel ellenőrzi hogy null-e a mutató típusú változó értéke.

Megoldás videó:

Megoldás forrása:



## 7. fejezet

# Helló, Conway!

### 7.1. Hangyaszimulációk

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

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

```
//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
```

```
//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

//ant.h
#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
```

```
//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];
 }
}
```



```
 1)
);

 QPainter painter (j*cellWidth, i*cellHeight,
 cellWidth, cellHeight);
}

QPainter painter (
 QPen (
 QColor (0,0,0),
 1)
);

painter.drawRect (j*cellWidth, i*cellHeight,
 cellWidth, cellHeight);

}
}

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

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

}

painter.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();
}
```

```
//antThread.cpp
```

```
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;
}
```

A main.cpp fájlban létrehozunk egy QApplication osztálybeli objektumot a neven, mellyel a parancssori objektumokat kezeljük. A QCommandLineOption x konstruktor segítségével létrehozunk egy x nevű kapcsolót, az első paraméterei a kapcsoló nevei a parancssorban, a második a leírás a kapcsolóhoz tartozó leírás ami egy mondat, a harmadik az érték neve, a negyedik az alapérték. A QCommandLineParser parser; paranccsal létrehozunk egy parser nevű objektumot a QCommandLineParser osztályhoz. A parser.addHelpOption(); paranccsal definiálunk a parser objektumhoz egy -h segítség kapcsolót, a parser.addVersionOption(); függvény pedig a verzióját adja meg az applikációnak, kapcsolója a -v. A parser.addOption függvénynek paraméterül adunk egy kapcsolót nevet amit az elemzés alatt keres. A parser.process ( a ); paranccsal beállítjuk hogy a parancssor a QApplication a objektumából származzon. A QString x paranccsal létrehozunk egy x nevű sztringet melynek értéket adjuk a parser.value függvény visszatérési értéket ami a parame-



terül adott kapcsoló neve, ehhez a kapcsolót hozzákell adnunk a `parser.addOption` függvénnyel. A `qrand` függvénnyel random számokat generálunk a parameterül adott mennyiségben ami a `QDateTime::currentMSecsSinceEpoch()` függvény visszatérési értéke azaz hogy hány miliszekundum telt el 1970-01-01T00:00:00.000 óta. Majd létrehozunk egy `w` nevű objektumot az `AntWin` osztályhoz parametizált konstruktor segítségével, a paraméterekben használt `toInt` függvény egész típusra konvertálja a `.` előtt levő sztringet. A `show` függvénnyel tesszük láthatóvá az objektumot és az `exec` függvénnyel beállítjuk hogy csak akkor záródjon be az ablak ha a felhasználó zárja be. Az `antwin.h` fájlban definiáljuk az `AntWin` alosztályt a `QMainWindow` foosztályból melyben a `Q_OBJECT` markot használjuk a privat részben mivel saját szignállal használjuk az osztályt. A publikus részben definiáljuk a parametizált konstruktort a megfelelő kezdőértékekkel és a `QWidget *parent = 0` paraméterrel pedig beállítjuk hogy a `QWidget` foosztálya legyen az objektumnak kiveve ha nem adunk meg értéket. Az `AntThread` osztálynak deklarálunk egy mutató változót `antThread` néven. Létrehozunk egy `closeEvent` függvényt melyben mutatókkal hivatkozunk függvényekre. Definiáljuk a `keyPressEvent` függvényt melyben definiáljuk hogy ha a `P` billentyűt lenyomáskor meghívódik az `antThread` változó által mutatott `pause` függvényt, ha pedig a `Q` vagy az `Escape` billentyű kerül lenyomásra akkor a `close` függvény hívódik meg. Mivel alosztályt használunk így a destruktort a virtual szóval definiáljuk. Definiáljuk a `PaintEvent` függvényt, majd a privat részben egy `Ants` osztály mutató típusú változót `ants` néven és egész típusú változókat egy és két és három dimenziós tömböt. Definiáljuk a slot-ként a `step` függvényt ami azt jelenti hogy a szignálok tudnak rá csatlakozni. Az `antthread.h` fájlban a `QThread` alosztályt definiáljuk melynek foosztálya a `QThread` osztály. A `Q_OBJECT` markot definiáljuk a szignálok és slotok használata miatt, definiáljuk a konstruktort az `AntThread` osztálynak melynek paramétereit egész típusú változók és az `Ants` osztálynak egy mutató változója. Majd definiáljuk az osztály destruktort, a `run` függvényt, a `finish` függvényt ami `false`-ra állítja a `running` változó értékét, a `pause` függvényt ami ellenkezőjére állítja a `paused` változó értékét és az `isRunning` függvényt ami visszaadja a `running` változó értékét. A privat részben definiáljuk `bool` típusú a `running` és a `paused` változót `true` és `false` kezdőértékekkel, egész típusú változókat, az `Ants` típusú mutató változót, egy három és egy két dimenziós tömböt, a `timeDevel` függvényt, a `newDir` függvényt, a `detDirs` függvényt, a `moveAnts` függvényt, a `s sumNbhs` függvényt és a `setPheromone` függvényt és a `step` függvényt szignálként. Az `ant.h` fájlban létrehozzuk az `Ant` osztályt amiben `x`, `y`, `dir` változókat egész típusúval deklaráljuk és az `Ant` konstruktor `x` és `y` paramétereinek segítségével definiáljuk őket és a konstruktorban a `dir` változó értékét a `grand` függvény által generált szám és a nyolccal való maradékos osztás határozza meg, így egy 0 és 8 közötti számot kapunk mivel 8 irány van a negyzetes halon. Létrehozunk egy `Ants` nevű vektort `Ant` típusú elemekkel és `typedef` típusúval így a későbbi használatkor nem szükséges a típusát elelni csak a nevet a vektornak. Definiáljuk az `AntWin` osztály konstruktorát melynek a `QWidget` a foosztály aminek a `QMainWindow` a foosztálya. A konstruktorban beállítjuk az ablak címet `Ant Simulation-re` a `SetWindowTitle` függvénnyel. A `this->x=x` paranccsal beállítjuk az aktuális objektum `x` változóinak értékét a konstruktori `x` változó értékeire, a `cellWidths` és a `cellHights` változók értékeit 6-ra állítjuk ezek adják meg majd a cellák pixel méretét. A `setFixedSize ( QSize ( width*cellWidth, height*cellHeight ) );` paranccsal beállítjuk az ablak méretét fixre amit a cellák száma és a cellák mérete ad meg. Létrehozunk `grids` néven 2 mátrixot `widths*heights` méretben amivel foglalunk heap memóriát a celláknak. Létrehozjuk a `gridIdx` változót inicializáljuk 0 értékkel majd a `grid` változót inicializáljuk a 0-ik indexű változóban majd for ciklust használva a mátrix értékeit a `cellDef` változó értékével tesszük egyenlővé. Létrehozunk egy vektort `ants` néven az `Ant` osztály segítségével, majd egy `AntThread` objektumot `antThread` néven. A `connect` függvénnyel kapcsoljuk össze az `antThread` objektum szignálját az aktuális objektum slotjával. Az `antThread` objektumra meghívjuk a `start` slot függvényt ami majd meghívja a `run` függvényt. Definiálunk egy `paintEvent`-et melyben létrehozunk egy `QPainter` osztályhoz esz egy `qpainter` nevű objektumot aktuális objektumként. A `grid` változó értékét a `grids` vektor `gridIdx` indexű elemére állítjuk. Majd az egymásbaagyazott for ciklusokkal végigmegyünk minden cellán és melyekben inicializáljuk a `rel` változót `double` típusúval `255.0/max` értékkel. A `qpainter` objektum `fillRect` függvényének segítségével a

QColor RGB szinkodu szine alapján kiszinezi az első két parameterből alkotott (x,y) koordinától kiinduló harmadik\*negyedik koordinata meretű pixel tómbot. Egysegesszínű lesz mivel az összes grid matrix értéke celDef és a rel változó értéket is max változóból számoljuk ki aminek az értéke nem változik. A grid matrix értékeit ellenőrizzük egy if feltétellel hogy nem egyenlőek-e a min változó értékével, ha a feltétel teljesül a setPen függvény segítségével beállítjuk a toll színt és vastagságot 1 pixelnyire, majd a drawRect függvény segítségével rajzolunk az első két parameterből alkotott (x,y) koordinátára egy harmadik\*negyedik parameter meretű teglalapot. Majd minden cellának a körvonalát hasonlóképp atszínezzük feketére. Letrehozunk egy for ciklust amely a h objektum értékeitől az ants vektorig megy, az auto azt jelenti hogy ciklus végéig van a változó a stack memoriában. A for ciklusban qt fekete színre és 1 pixel vastagságúra állítjuk a tollat és rajzolunk a h objektum x és y értékei alapján egy két pixellel kisebb oldalú négyzetet hogy közepén legyen a kirajzolásai (x,y) koordinátákat növeltük 1 pixellel, majd az end függvény jelzi a kirajzolás végét, a kezdetet a konstruktor hívása azaz a QPainter objektum létrehozása jelentette. Majd definiáljuk az AntWin függvény destruktort melyben a delete paranccsal töröljük a heap memorian foglalt helyeket. A step függvényt definiáljuk melyben a parameterül kapott gridIdx változó értéket értékeljük az aktuális objektum gridIdx változóinak. Az antthread.cpp fájlban definiáljuk az AntThread konstruktort melyben a lokális változók inicializáljuk a hozzátartozó konstruktorbeli értékekkel, majd létrehozunk egy két dimenziós tómbot melyeknek elemei 0-k. A qrand függvénnyel generálunk 1970 óta eltelt milliszekundumnyi számra állítja a pszeudosorozatát ahonnan visszatérési értéket ad egy számot ez a szám több szálon is ugyanaz lesz. Letrehozunk egy Ant osztálybeli objektumot h néven 0, 0 kezdőértékekkel amik az x és y változók értékei, majd ezek értékeit állítjuk a height és width változók fele +20 és -20 közötti random értékre a qrand függvény által generált random számok segítségével a for ciklus minden ciklusában ami annyiszor fut le amennyi az numAnts változó értéke. Majd a numAntsinCells matrix megfelelő értéket növeljük eggyel és mentjük a h objektumot az Ants nevű vektorba, majd a gridIdx változó értéket 0-ra állítjuk. Letrehozzuk a sumNbhs double típusú függvényt melynek paraméterei egy három dimenziós tómba memóriacímek és három int típusú változó. A függvényben inicializáljuk a sum változót 0.0 kezdőértékkel, majd az ifrom, ito, jfrom, fto változókat deklaráljuk, majd meghívjuk a detDirs függvényt a megfelelő paraméterekre majd a kapott i j értékekre minden i minden j értékére kiveve ha i is és j is 0 meghatározzuk az s és o változók értéket s=row+i és c=col+j alapján ahol az i és j változók a for ciklusban, mivel ezek értéke adja a grids matrix sor és oszlop koordinátáját így a megfelelő tartományban fellenniük azaz 0 és width-1 és height-1 között. Ha a megfelelő tartománytól kisebb az os vagy s értéke akkor a maximumra állítjuk, ha nagyobb akkor a minimumra. Az értékek úgyé azert nem lehet mindekettnél 0 mert akkor visszakapnánk a col és a row értékeit. Majd minden ciklusban hozzáadjuk a sum változó értékét az előző értékéhez ami a grids matrix s sorának o oszlophoz tartozó értéke +1 a harmadikon, majd a függvény visszatérési értéke ez a sum változó értéke lesz. A newDir függvény a megfelelő feltételnek megfelelő 0 és 7 közötti számot adja visszatérési értékel. A detDirs függvényben használjuk a switch beépített függvényt amely a parameterül kapott értékre használja az azonos case értékekhez tartozó kódot, ami jelen esetben beállítja az ifrom, ito, jfrom, jto változók értékeit és a break paranccsal kilep a függvényből. A moveAnts függvényben az x és y változókat inicializáljuk a sor és oszlop változók értékeivel amik a függvény paramétereiben vannak, majd deklaráljuk az ito, ifrom, jto, jfrom változókat melyeknek értékeit a detDirs függvény határozza meg a dir változó alapján ami függvény parameter. Inicializáljuk az összes változót a sumNbhs függvény visszatérési értékével, a random változót egy 0 és 1 közötti random generált számmal, majd a gvalue változót 0 értékkel. Majd a numNbs függvényhez hasonlóan meghatározzuk a kedvezo változó értéket, a value változóba meghatározzuk a kedvezo/összes értéket, majd minden ciklusban növeljük a gvalue változó értéket a value változó értékével. Ha a gvalue nagyobb egyenlő a random változó értékenél akkor a vsor=s és voszlop=o változók értékadása után a newDir függvényt meghívjuk aminek értéke a moveAnts függvény visszatérési értéke lesz vagy ha nem teljesül az if feltétel egyik ciklusban sem akkor a dir lesz a moveAnts függvény visszatérési értéke és a vsor és voszlop változó értékeit y és x változó értékeire amik a sor és oszlop változó értékei. A timeDevel

fuggvényben a `racsElotte` és `racsUtanna` két dimenziós tömbök memóriacímének értékei adjuk a `grids[0]` és `grids[1]` tömbök memóriacímét így változnak a `grids` tömbök értékei. A `grids` tömb indexének meghatározásakor a  $(gridIdx+1)\%2$  kifejezést azért használjuk hogy a `grids` tömbök értékeit egy `gridIdx` változó értékeiből meghatározzuk és sima +1 az lehetne kettő is. A for ciklusokkal végig megyünk minden `i` és `j` változéhoz tartozó tömb értékeken és a `racsUtanna` tömb értékeit egyenlővé tesszük a `racsElotte` tömb megfelelő értékeivel, majd ellenőrizzük hogy levonjuk a `racsUtanna` értékeiből az `evaporation` változó értéket ha az eredmény nagyobb egyenlő e mint 0 akkor az lesz a `racsUtanna` tömb értéke ha nem akkor 0. Majd for ciklussal az `ants` vektor elemein azaz `Ant` osztály objektumaiban inicializáljuk a sor és oszlop változókat -1 értékekkel és az `ujirany` változót a `moveAnts` függvény visszatérési értékeivel, majd meghívjuk a `setPheromone` függvényét megfelelő paraméterekkel, majd ellenőrizzük hogy a `numAntsinCells` két dimenziós tömb értéke kisebb-e mint a `cellAntMax` változó értéke akkor csökkentjük egyel a `numAntsinCells` `h.y` és `h.x`-hez tartozó értéket majd növeljük egyel `numAntsinCells` változó értéket és a `h.x` és `y` értéket -1-re állítjuk, a `h.dir` változót pedig az `ujterek` változó értékeire, a `gridIdx` változó értéket pedig az ellenkezőjére állítjuk. A `setPheromone` függvényben a `i` és `j` változók -1 és 2 közötti értékeivel meghatározzuk az `s` és `o` változók értékeit ha az `o` és `s` változókhoz tartozó `racs` tömb értéke kisebb egyenlő a `max` változónál növeljük a `pheromone` változó értékeivel ha nem akkor az értéke a `max` változó értéke lesz, majd a `racs` tömb sor és oszlop változóihoz tartozó értékeket hasonlóan határozzuk meg. A `run` függvényben a `running` változó értéket `true`-ra állítjuk és a `while` ciklus paramétere lesz így végtelen ciklus lesz amiben az `msleep` függvénnyel kezeljük a ciklus végrehajtását a `delay` változó értéke által meghatározott milliszekundummal, majd ha a `paused` változó értéke hamis akkor meghívjuk a `timeDevel` függvényt, majd a `step` szignált `gridIdx` paraméterrel. Majd definiáljuk az `AntThread` osztály destruktort hogy törölje a `numAntsinCells` matrix által foglalt memóriát.

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.2. Java életjáték

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

```
public class Sejtautomata extends java.awt.Frame implements Runnable {

 public static final boolean ELO = true;

 public static final boolean HALOTT = false;

 protected boolean [][][] racsok = new boolean [2][][];

 protected boolean [][] racs;

 protected int racsIndex = 0;

 protected int cellaSzelesseg = 20;
 protected int cellaMagassag = 20;

 protected int szelesseg = 20;
```

```
protected int magasság = 10;

protected int varakozas = 1000;

private java.awt.Robot robot;

public Sejtautomata(int szelesseg, int magassag)
{
 this.szelesseg = szelesseg;
 this.magassag = magassag;

 racsok[0] = new boolean[magassag][szelesseg];
 racsok[1] = new boolean[magassag][szelesseg];
 racsIndex = 0;
 racs = racsok[racsIndex];

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

 sikloKilovo(racs, 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)
 {
 cellaSzelesseg /= 2;
 cellaMagassag /= 2;
 setSize(Sejtautomata.this.szelesseg*cellaSzelesseg,
 Sejtautomata.this.magassag*cellaMagassag);
 validate();
 }
 else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N)
 {
 cellaSzelesseg *= 2;
```

```
 cellaMagassag *= 2;
 setSize(Sejtautomata.this.szelesseg*cellaSzelesseg,
 Sejtautomata.this.magassag*cellaMagassag);
 validate();
 }

 else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
 varakozas /= 2;
 else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
 varakozas *= 2;
 repaint();
}
});

addMouseListener(new java.awt.event.MouseAdapter()
{

 public void mousePressed(java.awt.event.MouseEvent m)
 {

 int x = m.getX()/cellaSzelesseg;
 int y = m.getY()/cellaMagassag;
 racsok[racsIndex][y][x] = !racsok[racsIndex][y][x];
 repaint();
 }
});

addMouseMotionListener(new java.awt.event.MouseMotionAdapter()
{

 public void mouseDragged(java.awt.event.MouseEvent m)
 {

 int x = m.getX()/cellaSzelesseg;
 int y = m.getY()/cellaMagassag;
 racsok[racsIndex][y][x] = ELO;
 repaint();
 }
});

cellaSzelesseg = 10;
cellaMagassag = 10;

try
{
 robot = new java.awt.Robot(
 java.awt.GraphicsEnvironment.
 getLocalGraphicsEnvironment().
 getDefaultScreenDevice());
}
catch(java.awt.AWTException e)
```

```
{
 e.printStackTrace();
}

setTitle("Sejtautomata");
setResizable(false);
setSize(szelesseg*cellaSzelesseg,
 magassag*cellaMagassag);
setVisible(true);

new Thread(this).start();
}

public void paint(java.awt.Graphics g)
{
 boolean [][] racs = racsok[racsIndex];

 for(int i=0; i<racs.length; ++i)
 {
 for(int j=0; j<racs[0].length; ++j)
 {
 if(racs[i][j] == ELO)
 g.setColor(java.awt.Color.BLACK);
 else
 g.setColor(java.awt.Color.WHITE);
 g.fillRect(j*cellaSzelesseg, i*cellaMagassag,
 cellaSzelesseg, cellaMagassag);

 g.setColor(java.awt.Color.LIGHT_GRAY);
 g.drawRect(j*cellaSzelesseg, i*cellaMagassag,
 cellaSzelesseg, cellaMagassag);
 }
 }
}

public int szomszedokSzama(boolean [][] racs,
 int sor, int oszlop, boolean allapot)
{
 int allapotuSzomszed = 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 = szelesseg-1;
 else if(o >= szelesseg)
 o = 0;

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

 if(racs[s][o] == allapot)
 ++allapotuSzomszed;
 }

 return allapotuSzomszed;
}

public void idoFejlodes()
{

 boolean [][] racsElotte = racsok[racsIndex];
 boolean [][] racsUtana = racsok[(racsIndex+1)%2];

 for(int i=0; i<racsElotte.length; ++i)
 {
 for(int j=0; j<racsElotte[0].length; ++j)
 {

 int elok = szomszedokSzama(racsElotte, i, j, ELO);

 if(racsElotte[i][j] == ELO)
 {

 if(elok==2 || elok==3)
 racsUtana[i][j] = ELO;
 else
 racsUtana[i][j] = HALOTT;
 }
 else
 {

 if(elok==3)
 racsUtana[i][j] = ELO;
 else
 racsUtana[i][j] = HALOTT;
 }
 }
 }

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

```
public void run()
{
 while(true)
 {
 try
 {
 Thread.sleep(varakozas);
 }

 catch (InterruptedException e) {}

 idoFejlodes();
 repaint();
 }
}

public void siklo(boolean [][] racs, int x, int y)
{
 racs[y+ 0][x+ 2] = ELO;
 racs[y+ 1][x+ 1] = ELO;
 racs[y+ 2][x+ 1] = ELO;
 racs[y+ 2][x+ 2] = ELO;
 racs[y+ 2][x+ 3] = ELO;

}

public void sikloKilovo(boolean [][] racs, int x, int y)
{
 racs[y+ 6][x+ 0] = ELO;
 racs[y+ 6][x+ 1] = ELO;
 racs[y+ 7][x+ 0] = ELO;
 racs[y+ 7][x+ 1] = ELO;

 racs[y+ 3][x+ 13] = ELO;

 racs[y+ 4][x+ 12] = ELO;
 racs[y+ 4][x+ 14] = ELO;

 racs[y+ 5][x+ 11] = ELO;
 racs[y+ 5][x+ 15] = ELO;
 racs[y+ 5][x+ 16] = ELO;
 racs[y+ 5][x+ 25] = ELO;

 racs[y+ 6][x+ 11] = ELO;
 racs[y+ 6][x+ 15] = ELO;
 racs[y+ 6][x+ 16] = ELO;
```



```
 racs[y+ 6][x+ 22] = ELO;
 racs[y+ 6][x+ 23] = ELO;
 racs[y+ 6][x+ 24] = ELO;
 racs[y+ 6][x+ 25] = ELO;

 racs[y+ 7][x+ 11] = ELO;
 racs[y+ 7][x+ 15] = ELO;
 racs[y+ 7][x+ 16] = ELO;
 racs[y+ 7][x+ 21] = ELO;
 racs[y+ 7][x+ 22] = ELO;
 racs[y+ 7][x+ 23] = ELO;
 racs[y+ 7][x+ 24] = ELO;

 racs[y+ 8][x+ 12] = ELO;
 racs[y+ 8][x+ 14] = ELO;
 racs[y+ 8][x+ 21] = ELO;
 racs[y+ 8][x+ 24] = ELO;
 racs[y+ 8][x+ 34] = ELO;
 racs[y+ 8][x+ 35] = ELO;

 racs[y+ 9][x+ 13] = ELO;
 racs[y+ 9][x+ 21] = ELO;
 racs[y+ 9][x+ 22] = ELO;
 racs[y+ 9][x+ 23] = ELO;
 racs[y+ 9][x+ 24] = ELO;
 racs[y+ 9][x+ 34] = ELO;
 racs[y+ 9][x+ 35] = ELO;

 racs[y+ 10][x+ 22] = ELO;
 racs[y+ 10][x+ 23] = ELO;
 racs[y+ 10][x+ 24] = ELO;
 racs[y+ 10][x+ 25] = ELO;

 racs[y+ 11][x+ 25] = ELO;

 }

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

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

Letrehozzuk a Sejtautomata osztályt ami öröklődik az `java.awt.Frame` osztályból és interfésze a `Runnable` osztály így ezen osztályok függvényeit használhatjuk a program során. Letrehozunk `public static final` típusú változókat azaz az osztályhoz tartozó konstansokat ami az jelenti hogy értékek nem változik azaz az `ELO` értéke `true` és a `HALOTT` értéke `false` marad. Letrehozunk `racsok` néven egy három dimenziós tömböt melynek két sora lesz és típusa `boolean` tehát igaz hamis értékek változókat tartalmaz majd. Letrehozunk egy két dimenziós tömbhöz mutató változót `racs` néven `boolean` típusú és inicializáljuk a szükséges egész típusú változókat és letrehozunk egy robot névű objektumot a `java.awt.Robot` osztályhoz. Letrehozzuk a Sejtautomata osztály parametizált konstruktorát szélesség és magasság egész típusú változókkal, melyben az aktuális objektum változóinak értékeit adjuk a konstruktorbeli változó értékek, így változókat használhatjuk a konstruktorban. A `racsok` tömb nulladik elemének értékeit adunk egy két dimenziós tömböt amivel lefoglalunk `magassag*szelesség` méretű memóriát `boolean` típusú, hasonlóan lefoglalunk memóriát a `racsok[1]` tömb memóriacíméhez. A `racsIndex` változó értéket 0-ra állítjuk és a `racs` két dimenziós tömb mutatójának értékeit adjuk a `racsok` tömb `racsIndex` elemű mutatójait. Két egymásba ágyazott for ciklus segítségével végigmegyünk a `racs` matrix minden elemén és értékeit adjuk a `HALOTT` változót, a matrix sorainak számát a `matrix.neve.length` paranccsal határoztuk meg az oszlopainak számát pedig a nulladik sorát megszámlálva a `neve[0].length` paranccsal, tehát a `length` függvény megadja egy egydimenziós tömb elemeinek számát. Majd meghívjuk a `siklokilovo` függvényt a `racs`, `ot` és hatvan paraméterekkel, majd letrehozunk egy `WindowEvent` objektumot hogy az ablak bezáraskor megszűnjön a lathatósága és kilepjen a program is. Letrehozunk egy `KeyEvent` objektumot is melyben megadjuk hogy a `K` billentyű lenyomása esetén a `CellaSzelesség` és `CellaMagasság` változók értékeit a felére állítjuk és az ablakméretet is atallítjuk az aktuális objektumhoz tartozó szélesség és magasság változók értékeit megszorozva a `CellaSzelesség` és `CellaMagasság` változókkal. A `validate` függvénnyel érvényesítjük az újraméretezést. Az `N` billentyű lenyomásakor hasonlóan számoljuk csak felezes helyett most kétszeresére nagyítunk azaz kétszeresére növeljük a cellák oldalait így négyszeresére a méretet. A `G` billentyű lenyomásával a varakozás változó értékeit a felére csökkentjük majd `repaint` függvénnyel meghívjuk a `paint` függvényt az újraszínezéshez. A `L` billentyű lenyomásakor hasonlóan kétszeresére növeljük a varakozás névű változó értékeit és meghívjuk a `repaint` függvényt az újrarajzoláshoz. Letrehozunk egy `MouseEvent` objektumot is melyben az `x` és `y` változó értékeiben kiszámoljuk hanyadik cellába kattintottunk sor és oszlop szerint, ha a cellákba kattintunk az értéke a cella sorszáma lesz, mivel egész típust használunk a tört értékeknek a meghatározásához ami lefele kerekíti a legközelebbi egész számmá azaz levágja a tizedesrészt. Majd a cella megfelelő koordinátákkal a hozzátartozó matrix értékeit az ellenkezőjére állítjuk és meghívjuk a `repaint` függvényt hogy frissüljön a kép. Letrehozunk egy másik `MouseEvent` eseményt a kijelölések kezelésére amiben a kattintásához hasonlóan meghatározzuk a kijelölt pixelekhez tartozó cellák sor és oszlopszámát majd az ezekhez tartozó matrix értékeket módosítjuk az `ELO` változó értékeire azaz `true`-ra majd meghívjuk a `repaint` függvényt. A koordinátáknál az `x` koordináta az a vízszintes elhelyezkedés, de a matrix koordinátánál a vízszintes elhelyezkedés az oszlopszáma ami a második [] részbe irando, az `y` koordináta is hasonlóan a koordinátáknál az oszlopszám matrixnal viszont a sorszám. A `cellaSzelesség` és `cellaMagasság` változók értékeit 10-re állítjuk. A `try` hibakezelő függvényben a robot mutató értékeit egy `java.awt.Robot` objektumra állítjuk amivel meghívjuk az aktuális környezetre az alapértelmezett képernyő eszközt a hozzátartozó hibakezéssel. A `setTitle` függvénnyel beállítjuk az ablak címet Sejtautomata-ra, majd beállítjuk a `setResizable` függvénnyel hogy újraméretezhető legyen, majd megadjuk az ablak méretét a `setSize` függvény paramétereként ami a cellák számából és egy cella méretének szorzatából számítunk ki. Majd `setVisible` függvénnyel beállítjuk a megjelenítést majd letrehozunk egy új szálát az aktuális objektumból és meghívjuk a `start` függvényt. Defináljuk a `paint` függvényt melynek paraméterében letrehozunk egy `g` objektumot a `java.awt.Graphics` osztályhoz amiben letrehozunk egy `racs` névű két dimenziós tömbhöz mutatót aminek értékeit adjuk a `racsok` matrix `racsIndex` elemének memóriacímét így a megfelelő `racsok` matrix értékeivel dolgozunk. Majd a matrix minden elemén végigmegyünk két egymásbaágyazott for ciklussal minden cellán és egy `if` feltellel

ellenorizzuk hogy az erteke megegyezik e az ELO valtozo ertekevel azaz igaz vagy nem, ha igaz akkor a szint feketere allitjuk, ha hamis akkor fehérre majd kiszinezuk az aktualis cella pixeleit az aktualis szinure. Majd a szint szurkere allitjuk es keretet rajzolunk az aktualis cella szelo pixeleire. Majd definialjuk a szomszedokSzama fuggvenyt esegz tipusu visszateresi ertekkel es a megfelelo parameterekkel, melyben inicializaljuk az allapotuSzomszed esegz tipusu valtozot nulla ertekkel. Majd ket egymasba agyazott for ciklus segitsegevel az adott cella minden szomszedos cellajanak ellenorizzuk az erteket es ha megfelelo akkor az allapotuSzomszed valtozo szamat noveljuk eggyel, melynek az erteke nulla es nyolc koze eso szam mivel nulla kezdertekekre allitottuk es nyolcszor novelhetjuk eggyel mivel minden cellanak nyolc szomszedja van. Az s es o valtozok hasznalata szukseges mivel a kepernyo veges igy ha elerjuk valamelyik oldal legszelo cellajat akkor a szemkozti oldal oldalhoz legkozelebbbi cellajara allitjuk az erteket, mindket valtozo erteke nem lehet egyszerre nulla mivel azaz eredeti cella lenne. Majd a fuggveny visszateresi erteke az allapotuSzomszed valtozo erte lesz. Majd definialjuk az idoFejlodes fuggvenyt melyben létrehozuk a racsElotte es racsUtana ket dimenzios tomb tipsus mutokat melyeknek ertekul adjuk a racsok harm dimenzios tomb megfelelo ket dimenzios tombjet melyet a racsIndex valtozo határoz meg, ha az eloszor deklalt racsIndex valtozo erteke egy es a lehetséges ertek null es egy akkor a masodszor deklalt valtozo ertekenek meghatarozasahoz kettovel valo maradékos osztást használunk mivel ha kettovel osztunk a maradék nulla vagy egy es ugye elotte noveljuk a racsIndex valtozo erteket. Majd a racsElotte mutathoz tartozo matrix minden erteken vegig megyunk egyesevel majd minden ciklusban kiszamitjuk hogy az adott cellanak hany igaz igazsagerteku szomszedos cellaja van. Majd ellenorizzuk hogy az aktualis cella igazsagerteku igaz e, ha igen akkor ellenorizzuk hogy a szomszedos cellaiban az igaz igazsagerteku cellak szama ketto vagy harm e ha ketto vagy harm akkor a visgalt cella koordinataraval azonos koordinataja racsUtanna mutatoju matrixhoz tartozo cella erteket igazra allitjuk, ha nem ketto vagy harm akkor a hozzatarozo racsUtanna matrix cellajanak igazsagerteku hamis lesz. Ha a racsElotte igazsagerteku nem igaz akkor hasonloan allitjuk az erteket de jelen esetben csak harm igaz igazsagerteku szomszed eseten lesz igaz egyebekent hamis lesz a cella igazsagerteku. Majd a racsIndex valtozo erteket az ellenkezojere allitjuk. A run fuggvenyben definialunk egy while vegtelen ciklust melyben a létrehozott szal objektumot altatjuk a varakozas valtozo ertekevel egyenlo miliszekundumig es definialunk hozza hibakezelest, majd meghivjuk az idoFejlodes fuggvenyt es a repaint fuggvenyt. Definialjuk a siklo es a sikloKilovo fuggvenyeket mellyel létrehozuk a siklot es a siklokilovot az adott racson az adott x es y cellakhoz. Definialjuk az update fuggvenyt egy java.awt.Graphics osztalybeli g objektum parameterrel melyre meghivjuk a paint fuggvenyt. Majd definialjuk a main fuggvenyt melyben létrehozunk egy Sejtautomat objektumot.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

```
//main.cpp
#include <QApplication>
#include "sejtablak.h"

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

```
{
 QApplication a(argc, argv);
 SejtAblak w(100, 75);
 w.show();

 return a.exec();
}
```

```
//sejtablak.h
#ifndef SEJTABLAK_H
#define SEJTABLAK_H

#include <QMainWindow>
#include <QPainter>
#include "sejtszal.h"

class SejtSzal;

class SejtAblak : public QMainWindow
{
 Q_OBJECT

public:
 SejtAblak(int szelesseg = 100, int magassag = 75, QWidget *parent = 0);
 ~SejtAblak();

 static const bool ELO = true;

 static const bool HALOTT = false;
 void vissza(int racsIndex);

protected:

 bool ***racsok;
 bool **racs;
 int racsIndex;
 int cellaSzelesseg;
 int cellaMagassag;
 int szelesseg;
 int magassag;
 void paintEvent(QPaintEvent*);
 void siklo(bool **racs, int x, int y);
 void sikloKilovo(bool **racs, int x, int y);

private:
 SejtSzal* eletjatek;
};
```

```
#endif
```

```
//sejtszal.h
#ifndef SEJTSZAL_H
#define SEJTSZAL_H

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

class SejtAblak;

class SejtSzal : public QThread
{
 Q_OBJECT

public:
 SejtSzal(bool ***racso, int szelesseg, int magassag,
 int varakozas, SejtAblak *sejtAblak);
 ~SejtSzal();
 void run();

protected:
 bool ***racso;
 int szelesseg, magassag;
 int racsIndex;
 int varakozas;
 void idoFejlodes();
 int szomszedokSzama(bool **racso,
 int sor, int oszlop, bool allapot);
 SejtAblak* sejtAblak;
};

#endif
```

```
//sejtablak.cpp
#include "sejtablak.h"

SejtAblak::SejtAblak(int szelesseg, int magassag, QWidget *parent)
 : QMainWindow(parent)
{
 setWindowTitle("A John Horton Conway-féle életjáték");

 this->magassag = magassag;
 this->szelesseg = szelesseg;
```

```
cellaSzelesseg = 6;
cellaMagassag = 6;

setFixedSize(QSize(szelesseg*cellaSzelesseg, magassag*cellaMagassag));

racsok = new bool**[2];
racsok[0] = new bool*[magassag];
for(int i=0; i<magassag; ++i)
 racsok[0][i] = new bool [szelesseg];
racsok[1] = new bool*[magassag];
for(int i=0; i<magassag; ++i)
 racsok[1][i] = new bool [szelesseg];

racsIndex = 0;
racs = racsok[racsIndex];

for(int i=0; i<magassag; ++i)
 for(int j=0; j<szelesseg; ++j)
 racs[i][j] = HALOTT;

sikloKilovo(racs, 5, 60);

eletjatek = new SejtSzal(racsok, szelesseg, magassag, 120, this);
eletjatek->start();
}

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

 bool **racs = racsok[racsIndex];

 for(int i=0; i<magassag; ++i)
 {
 for(int j=0; j<szelesseg; ++j)
 {
 if(racs[i][j] == ELO)
 qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
 cellaSzelesseg, cellaMagassag, Qt::black) ←
 ;
 else
 qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
 cellaSzelesseg, cellaMagassag, Qt::white) ←
 ;
 qpainter.setPen(QPen(Qt::gray, 1));

 qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
```

```
 cellaSzelesseg, cellaMagassag);
 }
}

qpainter.end();
}

SejtAblak::~SejtAblak()
{
 delete eletjatek;

 for(int i=0; i<magassag; ++i)
 {
 delete[] racsok[0][i];
 delete[] racsok[1][i];
 }

 delete[] racsok[0];
 delete[] racsok[1];
 delete[] racsok;
}

void SejtAblak::vissza(int racsIndex)
{
 this->racsIndex = racsIndex;
 update();
}

void SejtAblak::siklo(bool **racs, int x, int y)
{
 racs[y+ 0][x+ 2] = ELO;
 racs[y+ 1][x+ 1] = ELO;
 racs[y+ 2][x+ 1] = ELO;
 racs[y+ 2][x+ 2] = ELO;
 racs[y+ 2][x+ 3] = ELO;
}

void SejtAblak::sikloKilovo(bool **racs, int x, int y)
{
 racs[y+ 6][x+ 0] = ELO;
 racs[y+ 6][x+ 1] = ELO;
 racs[y+ 7][x+ 0] = ELO;
 racs[y+ 7][x+ 1] = ELO;
```

```
 racs[y+ 3][x+ 13] = ELO;

 racs[y+ 4][x+ 12] = ELO;
 racs[y+ 4][x+ 14] = ELO;

 racs[y+ 5][x+ 11] = ELO;
 racs[y+ 5][x+ 15] = ELO;
 racs[y+ 5][x+ 16] = ELO;
 racs[y+ 5][x+ 25] = ELO;

 racs[y+ 6][x+ 11] = ELO;
 racs[y+ 6][x+ 15] = ELO;
 racs[y+ 6][x+ 16] = ELO;
 racs[y+ 6][x+ 22] = ELO;
 racs[y+ 6][x+ 23] = ELO;
 racs[y+ 6][x+ 24] = ELO;
 racs[y+ 6][x+ 25] = ELO;

 racs[y+ 7][x+ 11] = ELO;
 racs[y+ 7][x+ 15] = ELO;
 racs[y+ 7][x+ 16] = ELO;
 racs[y+ 7][x+ 21] = ELO;
 racs[y+ 7][x+ 22] = ELO;
 racs[y+ 7][x+ 23] = ELO;
 racs[y+ 7][x+ 24] = ELO;

 racs[y+ 8][x+ 12] = ELO;
 racs[y+ 8][x+ 14] = ELO;
 racs[y+ 8][x+ 21] = ELO;
 racs[y+ 8][x+ 24] = ELO;
 racs[y+ 8][x+ 34] = ELO;
 racs[y+ 8][x+ 35] = ELO;

 racs[y+ 9][x+ 13] = ELO;
 racs[y+ 9][x+ 21] = ELO;
 racs[y+ 9][x+ 22] = ELO;
 racs[y+ 9][x+ 23] = ELO;
 racs[y+ 9][x+ 24] = ELO;
 racs[y+ 9][x+ 34] = ELO;
 racs[y+ 9][x+ 35] = ELO;

 racs[y+ 10][x+ 22] = ELO;
 racs[y+ 10][x+ 23] = ELO;
 racs[y+ 10][x+ 24] = ELO;
 racs[y+ 10][x+ 25] = ELO;

 racs[y+ 11][x+ 25] = ELO;

}
```



```
//sejtszal.cpp
#include "sejtszal.h"

SejtSzal::SejtSzal(bool ***racsok, int szelesseg, int magassag, int ←
 varakozas, SejtAblak *sejtAblak)
{
 this->racsok = racsok;
 this->szelesseg = szelesseg;
 this->magassag = magassag;
 this->varakozas = varakozas;
 this->sejtAblak = sejtAblak;

 racsIndex = 0;
}

int SejtSzal::szomszedokSzama(bool **racs,
 int sor, int oszlop, bool allapot)
{
 int allapotuSzomszed = 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 = szelesseg-1;
 else if(o >= szelesseg)
 o = 0;

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

 if(racs[s][o] == allapot)
 ++allapotuSzomszed;
 }

 return allapotuSzomszed;
}

void SejtSzal::idoFejlodes()
{
}
```

```
bool **racsElotte = racsok[racsIndex];
bool **racsUtana = racsok[(racsIndex+1)%2];

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

 int elok = szomszedokSzama(racsElotte, i, j, SejtAblak::ELO);

 if(racsElotte[i][j] == SejtAblak::ELO)
 {

 if(elok==2 || elok==3)
 racsUtana[i][j] = SejtAblak::ELO;
 else
 racsUtana[i][j] = SejtAblak::HALOTT;
 }
 else
 {

 if(elok==3)
 racsUtana[i][j] = SejtAblak::ELO;
 else
 racsUtana[i][j] = SejtAblak::HALOTT;
 }
 }
 racsIndex = (racsIndex+1)%2;
}

void SejtSzal::run()
{
 while(true) {
 QThread::msleep(varakozas);
 idoFejlodes();
 sejtAblak->vissza(racsIndex);
 }
}

SejtSzal::~SejtSzal()
{
}
```

A main.cpp fájlban létrehozunk egy objektumot a QApplication osztályhoz, majd egy w nevű objektumot a SejtAblak osztályhoz a parametizált konstruktora segítségével, majd beállítjuk hogy látható legyen az ob-

jektum és a main függvény visszatérési értékeiben beállítjuk hogy akkor lépjen ki a program ha bezárjuk az ablakot azaz az exit függvény lefut. Az exec függvény szükséges az eventek kezeléséhez is. A sejtablak.h fájlban létrehozunk a Sejtszal osztályt, majd a Sejtابلak osztályt a QMainWindow osztály alosztályaként és definiáljuk benne a Q\_OBJECT macrot mivel eventeket használunk szignallokkal. Majd definiáljuk az osztály konstruktorát melynek paramétere két egész típusú változó és megadjuk foosztálynak a QWidget osztályt kiveve ha nem adunk meg harmadik paramétert. Majd definiáljuk az osztály destruktort. Letrehozunk két static const típusú változót azaz két konstansot amik a memóriába maradnak a program futása alatt és értékük nem változik, ezek boolean típusúak az ELO értéke igaz, a HALOTT változója hamis. Majd definiáljuk a vissza függvényt racsIndex egész típusú változó paraméterrel és void visszatérési értékkel. A védett részben deklarálunk egész típusú változókat egy két dimenziós és egy három dimenziós tömb mutatókat és két void típusú függvényt és egy PaintEvent eseményt. Majd a privat részben létrehozunk egy életjatek nevű mutatót a Sejtابلak osztályhoz. A sejtszal.h fájlban létrehozunk a Sejtابلak osztályt majd a Sejtszal osztályt a QThread osztály alosztályaként, melyben definiáljuk a Q\_OBBJECT macrot és a Sejtszal konstruktorát a használatos paraméterekkel majd definiáljuk az osztály destruktort, majd definiáljuk a run függvényt. A védett részben egy három dimenziós tömböt és egész típusú változókat és az idoFejlo-des függvényt és a szomszedokSzama parametizált függvényt majd egy sejtابلak nevű mutatót a Sejtابلak osztályhoz. A header fájlunknál használtuk az ifndef macrot hogy ne definiáljuk kétszer az egyes header fájlokat. A sejtابلak.cpp fájlban definiáljuk a Sejtابلak osztály konstruktorát melynek paraméterei két egész típusú változó a szélesség és magasság és a QWidget osztályhoz egy mutató típusú változót parent néven, majd a QMainWindow foosztály konstruktorának paraméterül adjuk a parent objektum értéket. Majd aa setWindowTitle segítségével beállítjuk az ablak nevét A John Horton Conway-féle életjáték névre majd a szélesség és magasság konstruktori változók értéket kezdőértékül adjuk az adott objektum szélesség és magasság lokális változóinak majd inicializáljuk a cellaSzelesseg és cellaMagassag változók értékeit hálál. Majd a setFixedSize függvény segítségével beállítjuk a QSize függvényben megadott abalkmeretet fixre. A racsok három dimenziós tömb mutatónak értékek adunk két kétdimenziós tömb memóriacímét, majd a mindket két dimenziós tömbnek lefoglalunk magasság\*szelesseg méretnyi memóriát a heapen. Majd a racsIndex változó értéket nullára állítjuk és a racs mutató értéket a racsok három deimenziós tömb megfelelő két dimenziós altömbjére állítjuk jelen esetben a racsIndex változó értékűjére ami nulla. Ennek a matrixnak minden elemének az értéket a HALOTT változó értékére azaz hamisra állítjuk. Majd meghívjuk a racs tömbre és ot és hatvan paraméterre a sikloKilovo függvényt. Majd életjatek Sejtosztály típusú mutatónak értékek adunk egy új objektumot amit a Sejtosztály konstruktorával hozunk létre a racsok három dimenziós tömb mutatójával, szélesség, magasság változókkal és százhusz értékkel és az aktuális objektum objektum memóriacímé paraméterekkel. Majd az életjatek mutató objektumára meghívjuk a start függvényt. Majd definiálunk egy paintEvent-et melyben létrehozunk egy QPainter objektumot aktuális objektum paraméterrel QPainter osztályból majd létrehozunk egy két dimenziós tömbhöz memóriacímét racs néven melynek értékek adunk a racsok tömb racsIndex indexű tömbjének memóriacímét. Majd a racs matrix minden elemének értéket ellenőrizzuk hogy egyenlő-e az ELO változó értékével azaz igaz igazságértékű-e. Ha teljesül a feltétel akkor feketére festjük az adott cella pixeleit ha nem teljesül akkor fehérre festjük, majd minden cellának a szélső pixeleit szürkére festjük. Majd meghívjuk az end függvényt jelezve a festés végét. Majd definiáljuk a Sejtابلak destruktort melyben toroljuk a heapen foglalt mutatókat az objektumnak foglaltat és a 3 dimenziós tömbnek foglaltakat. Definiáljuk a vissza függvényt egész típusú racsIndex paraméterekkel melyben az aktuális objektum racsIndex változójának értékek adunk a függvény paraméterként adott értéket, majd meghívjuk az update függvényt. Definiáljuk a siklo és sikloKilovo függvényeket amelyek adott kétdimenziós tömbnek az x és y elemére megadják a siklo és a siklokilóvót. A sejtszal.cpp fájlban a létrehozunk a Sejtszal osztály konstruktorát melyben inicializáljuk az aktuális objektum lokális változóit a konstruktori értékükkel majd a racsIndex változó értéket nullára állítjuk. Majd definiáljuk a szomszedokSzama függvényt melyben az állapotuSzomszed változóba meghatározzuk hány adott igazságértékű szomszedos

cellaja van az adott cellanak, mivel egy cella egy matrix koordinata így noveljük a matrix koordinatait a megfelelő -1 és 1 közötti értékekkel soronként minden oszlopra, kivéve 0 és 0 értékkel mivel azzal az eredeti cellan maradjunk és az ablak szelet átlépve az ellentétes oldal legelső celláinak értékeivel dolgozunk. Az `idoFejloDes` függvényben létrehozunk két két dimenziós tömböt `racElotte` és `racUtanna` néven melyeknek értékei a `racok` matrix megfelelő indexű elemeinek mutatója. A megfelelő index meghatározásához a `racIndex` változó értéket noveljük eggyel majd maradékos osztást végzünk rajta kettővel így ha az először megadott `racIndex` érték egy akkor a második nulla lesz. Majd két egymásba ágyazott for ciklus segítségével a `racElotte` mutatóhoz tartozó matrix minden elemére meghatározzuk az előző egész típusú változó értéket a `szomszedokSzama` függvény segítségével. Majd egy if feltétel segítségével ellenőrizzük hogy a cella értéke igaz-e azaz egyenlő-e az `ELO` változó értékével, ami ugye a `SejtAblak` osztályhoz tartozik mivel statikus kulcsszóval lett definiálva így használatához az osztálynévvel kell rá hivatkozni :: operátorral, ahogy a `HALOTT` változó esetén is. Tehát ha megfelel az if feltételnek akkor ha a hozzátartozó előző változó értéke 2 vagy 3 akkor a hozzátartozó `racUtanna` cella értéket az `ELO` változó igazságértékére állítjuk, ha nem teljeseul ez a feltétel akkor `HALOTT`-ra. Ha ugye az eredeti feltétel nem teljesül tehát az aktuális cella értéke nem egyenlő az `ELO` változó értékével akkor ellenőrizzük hogy az előző változó értéke három-e ha teljesül a feltétel akkor az aktuális cellához tartozó `racUtanna` matrixbeli cella értéket `ELO` változó értékére állítjuk, ha nem teljeseul a feltétel akkor a cella értéket a `HALOTT` változó értékére állítjuk. Majd a `racIndex` változó értéket az ellenkezőjére állítjuk. Defináljuk a `run` függvényt egy while végtelen ciklussal melyben a `QThread` osztálybeli függvényt használva állítjuk a szálát azaz késleltetjük az `idoFejloDes` függvény meghívását a várakozás változó értéknyi miliszekundumig az `msleep` függvény segítségével. A várakozás változó értéke határozza meg a ciklusok közötti sebességet és a ciklus mozgási sebességet is. Majd meghívjuk az `idoFejloDes` függvényt majd a `sejtAblak` mutató által mutatott vissza függvényt meghívjuk a `racIndex` paraméterrel. Majd meghívjuk az osztály destruktort.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.4. BrainB Benchmark

```
//main.cpp
#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 Bótfai") << endl;
```

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

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

qout << QString::fromUtf8 ("Ez a program szabad szoftver; ↵
 terjeszthető illetve módosítható a Free Software") << endl;
qout << QString::fromUtf8 ("Foundation által kiadott GNU General ↵
 Public License dokumentumában leírtak;") << endl;
qout << QString::fromUtf8 ("akár a licenc 3-as, akár (tetszőleges) ↵
 későbbi változata szerint.\n") << endl;

qout << QString::fromUtf8 ("Ez a program abban a reményben kerül ↵
 közreadásra, hogy hasznos lesz, de minden") << endl;
qout << QString::fromUtf8 ("egyéb GARANCIA NÉLKÜL, az ↵
 ELADHATÓSÁGRA vagy VALAMELY CÉLRA VALÓ") << endl;
qout << QString::fromUtf8 ("ALKALMAZHATÓSÁGRA való származtatott ↵
 garanciát is beleértve. További") << endl;
qout << QString::fromUtf8 ("részleteket a GNU General Public ↵
 License tartalmaz.\n") << endl;

qout << "http://gnu.hu/gplv3.html" << endl;

QRect rect = QApplication::desktop()->availableGeometry();
BrainBWin brainBWin (rect.width(), rect.height());
brainBWin.setWindowState (brainBWin.windowState() ^ Qt::↵
 WindowFullScreen);
brainBWin.show();
return app.exec();
}
```

```
//BrainBWin.h
#ifndef BrainBWin_H
#define 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);

double mean (std::vector<int> vect) {

 if (vect.size() > 0) {
 double sum = std::accumulate (vect.begin (), vect.end (), 0.0 ←
);
 return sum / vect.size();
 } else {
 return 0.0;
 }
}

double var (std::vector<int> vect, double mean) {

 if (vect.size() > 1) {

 double accum = 0.0;

 std::for_each (vect.begin (), vect.end (), [&] (const double ←
 d) {
 accum += (d - mean) * (d - mean);
 });

 return sqrt (accum / (vect.size()-1));
 } else {
 return 0.0;
 }
}

}

void millis2minsec (int millis, int &min, int &sec) {

 sec = (millis * 100) / 1000;
 min = sec / 60;
 sec = sec - min * 60;

}

bool save (int t) {
```

```
bool ret = false;

if (!QDir (statDir).exists())
 if (!QDir().mkdir (statDir)) {
 return false;
 }

QString name = statDir + "/Test-" + QString::number (t);
QFile file (name + "-screenimage.png");
if (file.open (QIODevice::WriteOnly)) {
 ret = pixmap.save (&file, "PNG");
}

QFile tfile (name + "-stats.txt");
ret = tfile.open (QIODevice::WriteOnly | QIODevice::Text);
if (ret) {
 QTextStream textStremam (&tfile);

 textStremam << appName + " " + appVersion << "\n";
 textStremam << "time : " << brainBThread->getT() << "\n";
 textStremam << "bps : " << brainBThread->get_bps() << "\n";
 textStremam << "noc : " << brainBThread->nofHeroes() << "\n";
 textStremam << "nop : " << brainBThread->get_nofPaused() << "\n";

 textStremam << "lost : " << "\n";
 std::vector<int> l = brainBThread->lostV();
 for (int n : l) {
 textStremam << n << ' ';
 }
 textStremam << "\n";
 int m = mean (l);
 textStremam << "mean : " << m << "\n";
 textStremam << "var : " << var (l, m) << "\n";

 textStremam << "found : " ;
 std::vector<int> f = brainBThread->foundV();
 for (int n : f) {
 textStremam << n << ' ';
 }
 textStremam << "\n";
 m = mean (f);
 textStremam << "mean : " << m << "\n";
 textStremam << "var : " << var (f, m) << "\n";

 textStremam << "lost2found: " ;
 for (int n : lost2found) {
```



```

 textStremam << n << ' ';
 }
 textStremam << "\n";
 int m1 = m = mean (lost2found);
 textStremam << "mean : " << m << "\n";
 textStremam << "var : " << var (lost2found, m) << "\n" ←
 ;

 textStremam << "found2lost: " ;
 for (int n : found2lost) {
 textStremam << n << ' ';
 }
 textStremam << "\n";
 int m2 = m = mean (found2lost);
 textStremam << "mean : " << m << "\n";
 textStremam << "var : " << var (found2lost, m) << "\n" ←
 ;

 if (m1 < m2) {
 textStremam << "mean(lost2found) < mean(found2lost)" << "\n" ←
 ";
 }

 int min, sec;
 millis2minsec (t, min, sec);
 textStremam << "time : " << min << ":" << sec << "\n";

 double res = (((double) m1+ (double) m2) /2.0) /8.0) ←
 /1024.0;
 textStremam << "U R about " << res << " Kilobytes\n";

 tfile.close();
}
return ret;
}

```

```
public slots :
```

```

 void updateHeroes (const QImage &image, const int &x, const int &y);
 //void stats (const int &t);
 void endAndStats (const int &t);
};

```

```
#endif // BrainBWin
```

```

//BrainBThread.h
#ifndef BrainBThread_H

```

```
#define BrainBThread_H

#include <QThread>
#include <QSize>
#include <QImage>
#include <QDebug>
#include <sstream>
#include <QPainter>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <opencv2/opencv.hpp>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>

class Hero;
typedef std::vector<Hero> Heroes;

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;
 }
 }
}
```

```
};

class BrainBThread : public QThread
{
 Q_OBJECT

 //Norbi
 cv::Scalar cBg { 247, 223, 208 };
 cv::Scalar cBorderAndText { 47, 8, 4 };
 cv::Scalar cCenter { 170, 18, 1 };
 cv::Scalar cBoxes { 10, 235, 252 };

 /*
 //Matyi
 cv::Scalar cBg { 86, 26, 228 };
 cv::Scalar cBorderAndText { 14, 177, 232 };
 cv::Scalar cCenter { 232, 14, 103 };
 cv::Scalar cBoxes { 14, 232, 195 };
 */

 Heroes heroes;
 int heroRectSize {40};

 cv::Mat prev {3*heroRectSize, 3*heroRectSize, CV_8UC3, cBg };
 int bps;
 long time {0};
 long endTime {10*60*10};
 int delay {100};

 bool paused {true};
 int nofPaused {0};

 std::vector<int> lostBPS;
 std::vector<int> foundBPS;

 int w;
 int h;
 int dispShift {40};

public:
 BrainBThread (int w = 256, int h = 256);
 ~BrainBThread();

 void run();
 void pause();
 void set_paused (bool p);
 int getDelay() const {
```

```
 return delay;
 }
 void setDelay (int delay) {
 if (delay > 0) {
 delay = delay;
 }
 }

 void devel() {
 for (Hero & hero : heroes) {
 hero.move (w, h, (h<w) ?h/10:w/10);
 }
 }

 int nofHeroes () {
 return heroes.size();
 }

 std::vector<int> &lostV () {
 return lostBPS;
 }

 std::vector<int> &foundV () {
 return foundBPS;
 }

 double meanLost () {
 return mean (lostBPS);
 }

 double varLost (double mean) {
 return var (lostBPS, mean);
 }
}
```

```
}

double meanFound () {

 return mean (foundBPS);

}

double varFound (double mean) {

 return var (foundBPS, mean);

}

double mean (std::vector<int> vect) {

 double sum = std::accumulate (vect.begin (), vect.end (), 0.0);
 return sum / vect.size();

}

double var (std::vector<int> vect, double mean) {

 double accum = 0.0;
 std::for_each (vect.begin (), vect.end (), [&] (const double d) ↔
 {
 accum += (d - mean) * (d - mean);
 });

 return sqrt (accum / (vect.size()-1));

}

int get_bps() const {

 return bps;

}

int get_w() const {

 return w;

}

bool get_paused() const {

 return paused;

}
```

```
int get_nofPaused() const {
 return nofPaused;
}

void decComp() {
 lostBPS.push_back (bps);

 if (heroes.size() > 1) {
 heroes.pop_back();
 }

 for (Hero & hero : heroes) {
 if (hero.agility >= 5) {
 hero.agility -= 2;
 }
 }
}

void incComp() {
 foundBPS.push_back (bps);

 if (heroes.size() > 300) {
 return;
 }

 /*
 Hero other (w/2 + 200.0*std::rand() / (RAND_MAX+1.0)-100,
 h/2 + 200.0*std::rand() / (RAND_MAX+1.0)-100,
 255.0*std::rand() / (RAND_MAX+1.0), 11, "New Entropy ←
 ");
 */

 double rx = 200.0;
 if(heroes[0].x - 200 < 0)
 rx = heroes[0].x;
 else if(heroes[0].x + 200 > w)
 rx = w - heroes[0].x;

 double ry = 200.0;
 if(heroes[0].y - 200 < 0)
 ry = heroes[0].y;
```

```
else if(heroes[0].y + 200 > h)
 ry = h - heroes[0].y;

 Hero other (heroes[0].x + rx*std::rand() / (RAND_MAX+1.0)-rx/2,
 heroes[0].y + ry*std::rand() / (RAND_MAX+1.0)-ry/2,
 255.0*std::rand() / (RAND_MAX+1.0), 11, "New Entropy ↵
 ");

 heroes.push_back (other);

 for (Hero & hero : heroes) {

 ++hero.conds;
 if (hero.conds == 3) {
 hero.conds = 0;
 hero.agility += 2;
 }

 }

}

void draw () {

 cv::Mat src (h+3*heroRectSize, w+3*heroRectSize, CV_8UC3, cBg);

 for (Hero & hero : heroes) {

 cv::Point x (hero.x-heroRectSize+dispShift, hero.y- ↵
 heroRectSize+dispShift);
 cv::Point y (hero.x+heroRectSize+dispShift, hero.y+ ↵
 heroRectSize+dispShift);

 cv::rectangle (src, x, y, cBorderAndText);

 cv::putText (src, hero.name, x, cv::FONT_HERSHEY_SIMPLEX, .35, ↵
 cBorderAndText, 1);

 cv::Point xc (hero.x+dispShift , hero.y+dispShift);

 cv::circle (src, xc, 11, cCenter, CV_FILLED, 8, 0);

 cv::Mat box = src (cv::Rect (x, y));

 cv::Mat cbox (2*heroRectSize, 2*heroRectSize, CV_8UC3, cBoxes ↵
);
 box = cbox*.3 + box*.7;

 }

}
```

```
cv::Mat comp;

cv::Point focusx (heroes[0].x- (3*heroRectSize) /2+dispShift, ↵
 heroes[0].y- (3*heroRectSize) /2+dispShift);
cv::Point focusy (heroes[0].x+ (3*heroRectSize) /2+dispShift, ↵
 heroes[0].y+ (3*heroRectSize) /2+dispShift);
cv::Mat focus = src (cv::Rect (focusx, focusy));

cv::compare (prev, focus, comp, cv::CMP_NE);

cv::Mat aRgb;
cv::extractChannel (comp, aRgb, 0);

bps = cv::countNonZero (aRgb) * 10;

//qDebug() << bps << " bits/sec";

prev = focus;

QImage dest (src.data, src.cols, src.rows, src.step, QImage:: ↵
 Format_RGB888);
dest=dest.rgbSwapped();
dest.bits();

emit heroesChanged (dest, heroes[0].x, heroes[0].y);

}

long getT() const {

 return time;

}

void finish () {

 time = endTime;

}

signals:

 void heroesChanged (const QImage &image, const int &x, const int &y);
 void endAndStats (const int &t);

};

#endif // BrainBThread_H
```



```
//BrainBWin.cpp
#include "BrainBWin.h"

const QString BrainBWin::appName = "NEMESPOR BrainB Test";
const QString BrainBWin::appVersion = "6.0.3";

BrainBWin::BrainBWin (int w, int h, QWidget *parent) : QMainWindow (←
 parent)
{

 // setWindowTitle(appName + " " + appVersion);
 // setFixedSize(QSize(w, h));

 statDir = appName + " " + appVersion + " - " + QDate::currentDate() ←
 .toString() + QString::number (QDateTime:: ←
 currentMsecsSinceEpoch());

 brainBThread = new BrainBThread (w, h - yshift);
 brainBThread->start();

 connect (brainBThread, SIGNAL (heroesChanged (QImage, int, int) ←
),
 this, SLOT (updateHeroes (QImage, int, int)));

 connect (brainBThread, SIGNAL (endAndStats (int)),
 this, SLOT (endAndStats (int)));

}

void BrainBWin::endAndStats (const int &t)
{

 qDebug() << "\n\n\n";
 qDebug() << "Thank you for using " + appName;
 qDebug() << "The result can be found in the directory " + statDir;
 qDebug() << "\n\n\n";

 save (t);
 close();

}

void BrainBWin::updateHeroes (const QImage &image, const int &x, const int ←
 &y)
{

 if (start && !brainBThread->get_paused()) {
```

```
int dist = (this->mouse_x - x) * (this->mouse_x - x) + ←
 (this->mouse_y - y) * (this->mouse_y - y);

if (dist > 121) {
 ++nofLost;
 nofFound = 0;
 if (nofLost > 12) {

 if (state == found && firstLost) {
 found2lost.push_back (brainBThread ←
 ->get_bps());
 }

 firstLost = true;

 state = lost;
 nofLost = 0;
 //qDebug() << "LOST";
 //double mean = brainBThread->meanLost();
 //qDebug() << mean;

 brainBThread->decComp();
 }
} else {
 ++nofFound;
 nofLost = 0;
 if (nofFound > 12) {

 if (state == lost && firstLost) {
 lost2found.push_back (brainBThread ←
 ->get_bps());
 }

 state = found;
 nofFound = 0;
 //qDebug() << "FOUND";
 //double mean = brainBThread->meanFound();
 //qDebug() << mean;

 brainBThread->incComp();
 }
}

}

pixmap = QPixmap::fromImage (image);
update();
}
```

```
void BrainBWin::paintEvent (QPaintEvent *)
{
 if (pixmap.isNull()) {
 return;
 }

 QPainter qpainter (this);

 xs = (qpainter.device()->width() - pixmap.width()) /2;
 ys = (qpainter.device()->height() - pixmap.height() +yshift) /2;

 qpainter.drawPixmap (xs, ys, pixmap);

 qpainter.drawText (10, 20, "Press and hold the mouse button on the ↵
 center of Samu Entropy");

 int time = brainBThread->getT();
 int min, sec;
 millis2minsec (time, min, sec);
 QString timestr = QString::number (min) + ":" + QString::number (↵
 sec) + "/10:0";
 qpainter.drawText (10, 40, timestr);

 int bps = brainBThread->get_bps();
 QString bpsstr = QString::number (bps) + " bps";
 qpainter.drawText (110, 40, bpsstr);

 if (brainBThread->get_paused()) {
 QString pausedstr = "PAUSED (" + QString::number (↵
 brainBThread->get_nofPaused()) + ")";

 qpainter.drawText (210, 40, pausedstr);
 }

 qpainter.end();
}

void BrainBWin::mousePressEvent (QMouseEvent *event)
{
 brainBThread->set_paused (false);
}

void BrainBWin::mouseReleaseEvent (QMouseEvent *event)
{
 //brainBThread->set_paused(true);
}
```

```
void BrainBWin::mouseMoveEvent (QMouseEvent *event)
{
 start = true;

 mouse_x = event->pos().x() -xs - 60;
 //mouse_y = event->pos().y() - yshift - 60;
 mouse_y = event->pos().y() - ys - 60;
}

void BrainBWin::keyPressEvent (QKeyEvent *event)
{
 if (event->key() == Qt::Key_S) {
 save (brainBThread->getT());
 } else if (event->key() == Qt::Key_P) {
 brainBThread->pause();
 } else if (event->key() == Qt::Key_Q || event->key() == Qt::Key_Escape) {
 close();
 }
}

BrainBWin::~BrainBWin()
{
}
```

```
//BrainBThread.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()
{

}

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;

}
```

A main.cpp fájlban létrehozunk egy app nevű objektumot a QApplication osztályhoz, majd létrehozunk egy objektumot a QTextStream osztályhoz qout néven melynek konstruktorába az stdout kerül ami a szabványos kimenetre utal tehát a szabványos kimenetet használja kiírásra. Beállítjuk a setCodec függvény segítségével a szöveg kódolását UTF-8-ra, majd a kiírandó szöveget a cout-hoz hasonlóan kiírjuk mivel ezzel is egy csatornát hoztunk létre a kimenetre, de jelen esetben qout-ot használunk. Kiírjuk a BrainBWin osztálynak az appName változójának értékét, a magyar szövegrészeknél ahol van benne ékezetes betű is a hatékony kódolás érdekében jelezzük hogy UTF-8 kódolása a szöveg úgy hogy a QString osztály fromUtf8 függvényébe írjuk a szöveget, a \n és az endl is az újsort jelenti és többször is használhatóak egymás után, de jelen esetben a kód formalitása miatt egyet egyet használtunk belőle. Létrehozzuk a QRect osztály rect nevű objektumát melynek értéke a QApplication osztály desktop függvényébe meghívja az availableGeometry függvényt, ami meghatározza hogy mekkora területű hely van az applikáció számára az ablakban. Majd létrehozzuk a BrainBWin osztálynak egy objektumát brainBWin néven a rect.width és rect.height adatokkal azaz a rect objektumban lévő width függvényre hivatkozunk ami megadja ugye az asztalon az applikációk számára rendelkezésre álló hely szélességét, a height függvény pedig a magasságát. A setWindowState függvény segítségével beállítjuk a képernyő merettípusát, jelen esetben teljes képernyőben azaz nincs keret az ablak körül hanem kitölti a teljes ablakot. Mivel xor-osan kezeli a paraméterül kapott értéket a windowState így megadhatjuk xor művelettel is. Majd láthatóvá tesszük a show függvény segítségével a brainBWin objektumot, majd a main függvény visszatérési értékét az exec függvényre állítjuk aminek nulla ha meghívódik az exit függvény azaz bezárjuk az ablakot és az események használatához is szükséges. Létrehozunk az enum segítségével egy típust playerstate néven melyben definiáljuk a hozzátartozó változókat lost és found néven. Majd definiáljuk a BrainBWin osztályt a QMainWindow osztály alosztályaként

melyben használjuk a `Q_OBJECT` marcot mivel szignálokat használunk, majd létrehozunk egy mutatót a `BrainBThread` osztályhoz `brainBThread` néven, majd létrehozunk egy objektumot a `QPixmap` osztályhoz `pixmap` néven, majd létrehozunk egy mutatót a `Heroes` osztályhoz `heroes` néven. Majd dekraláljuk és inicializáljuk a szükséges változókat a megfelelő kezdőértékekkel és létrehozunk két egész típusú vektort `lost2found` és `found2lost` néven, majd létrehozunk egy `QString` objektumot `statDir` néven. Majd létrehozunk két `static const` típusú `QString` objektumot `appName` és `appVersion` néven mivel nem parametizált konstruktort használunk így két üres sztringet melyek az osztályhoz tartoznak és értéket egyszer lehet megadni objektumonként. Majd definíáljuk a `BrainBWin` osztály konstruktorát melynek paramétereit inicializáljuk a `w` és `h` egész típusú változókat ketszázötvenhat kezdőértékekkel és a `QWidget *parent` mutatót nulla kezdőértékekkel. Majd létrehozunk egy `closeEvent`-et melyben egy `if` feltetellel ellenőrizzük a `save` függvény visszatérési értéke melynek paramétere hogy a `brainThread` mutató objektumára meghívjuk a `getT` függvényt, ha igaz a `save` függvény visszatérési értéke akkor a `brainThread` mutató objektumára meghívjuk a `finish` függvényt ami leállítja a `run` függvényt, majd az `event` objektumára meghívjuk az `accept` függvényt ami engedélyezi az ablak bezárását, ha nem teljesül a feltétel akkor az `event` objektumára az `ignore` függvény hívódik meg a nem engedélyezi az ablak bezárását így nem tudjuk bezárni az ablakot. Majd definíáljuk a `brainBWin` virtuális destruktort a virtuális kifejezéssel, melyet akkor használunk ha az adott osztály foosztályként használjuk és nincs virtuális destruktort tartalmazó foosztálya, mivel a foosztály virtuális destruktora meghívja az alosztályok destruktort. Dekraláljuk a használatos eseményeket, majd definíáljuk a `mean` függvényt `double` visszatérési értékkel, melynek paramétere egy egész típusú vektor `vect` névvel. Majd a függvényben ellenőrizzük egy `if` feltetellel hogy a paraméterül adott vektor mérete nagyobb-e mint egy, ehhez a vektorra meghívjuk a `size` függvényt mely megadja a vektor elemeinek a számát. Ha teljesül az `if` feltétel akkor inicializáljuk a `sum` `double` típusú változót az `accumulate` függvény visszatérési értéke lesz. Az `accumulate` összeadja az első paraméterül a második paraméteréig az elemeket az első és második paraméter elemével együtt, a harmadik paraméterben pedig megadhatjuk a kezdőértéket, ami jelen esetben `0.0`. Mivel a vektor elemeit adjuk össze így az első elem meghatározására a vektorra használjuk a `begin` függvényt, az utolsó elem meghatározására pedig az `end` függvényt. Majd a függvény visszatérési értéke a `sum` változó és a vektorra meghívott `size` függvény visszatérési értékenek a hányadosa lesz. Ha nem teljesül az `if` feltétel akkor a függvény visszatérési értéke `0.0` lesz. Majd definíáljuk a `var` függvényt `double` típusúval, melynek első paramétere egy egész típusú vektor neve ami jelen esetben `vect`, a második paraméter `double` típusú változó neve ami jelen esetben `mean`. A függvényben ellenőrizzük egy `if` feltetellel hogy a vektor mérete nagyobb-e mint egy, a vektor méretét a vektorra meghívott `size` függvény visszatérési értéke lesz ami a vektor elemeinek a száma. Ha teljesül az `if` feltétel akkor inicializáljuk a `double` típusú `accum` változót `0.0` kezdőértékekkel. A vektor elemeivel való számoláshoz a `for_each` függvényt használjuk mely az első paraméter értéketől a második paraméter értékeig átadja az adott értéket memóriacímét a harmadik paraméterként megadott függvénynek, ami egy `lambda` kifejezés mely a kapott paraméternek veszi a referenciáját így memóriacím helyett az értéket mentjük a `d` változóba amivel számolunk minden érték esetén. Mivel a második paraméterként adott érték memóriacímét nem kezeli a függvény így az elemek számából ki kell vonni egyet mivel az utolsó vektor elemmel nem számoltunk. A paraméterül adott vektor minden elemének értékeiből kivéve az utolsóból kivonjuk a paraméterül adott `mean` változó értékét majd megszorozzuk a kapott értéket önmagával így negyzetre emelünk, ezek a számítások az `accum` változó értékeit változtatják ami `double` típusú így mivel kivonunk egész számból és nem a `d` változóba mentjük vissza így használhattuk volna `int` azaz egész típusúval is. Majd a függvény visszatérési értéke az `sqrt` függvény visszatérési értéke lesz melynek paramétere a vektor elemeinek az értékenek a `mean` változóval vett különbségeinek a negyzetösszegének és a vektor elemszámának a hányadosa, de a az utolsó vektor elemmel nem számoltunk a `for_each` függvényben így a `size` függvény visszatérési értékeiből kikell vonni egyet. Ha nem teljesül az eredeti feltétel azaz a vektor elemeinek a száma nulla vagy egy akkor a függvény visszatérési értéke `0.0` lesz. Majd definíáljuk a `millis2minsec` függvényt `void` típusúval és egy egész típusú értékkel és két egész ti-

pusu memoriacimmet. A függvényben meghatározzuk a parameterul adott miliszekundum százszorosának az értéket percben és a maradék másodpercben mivel egész számokkal dolgozunk, ezeket az értékeket a második harmadik parameterkent megadott memoriacimhez tartozó változó értékekben számoljuk. A számításhoz használtuk hogy egy másodperc ezer miliszekundum és egy perc hatvan másodperc. Az átváltás így sem pontos mivel százszorosával számolva is tízdel osztunk de egész típusú változó értéket számolva az osztás a tízes helyiértéket levágja. Majd definiáljuk a save függvényt bool típusú mellynek parametere egész típusú változó jelen esetben t névvel. A függvényben inicializáljuk a rev változót bool típusúval és hamis igazságértékkel. Majd egy if feltetellel ellenőrizzük hogy létezik-e StatDir névű mappa az aktuális könyvtárban, ehhez a QString konstruktort használjuk ami a parameterkent adott sztring névű mappához ad vissza egy mutatót, az exist függvény pedig ellenőrzi hogy létezik-e a mappa ha létezik akkor a visszatérési értéke igaz, ha nem akkor hamis illetve ha azonos fájl névű fájl található akkor is hamis. Majd mivel egy ! van az if feltétel előtt így ha az exist függvény igazságértéke hamis akkor lépünk a beagyazott if feltételre ahol az aktuális QT mappába létrehozunk egy statDir névű mappát ha már létezik a parameterkent adott névű mappa akkor a visszatérési értéke hamis, ha nem akkor létrehoz egyet és a visszatérési értéke igaz lesz. Mivel ! jelet használunk a feltételben az mkdir függvény előtt így a feltétel akkor teljesül ha az mkdir függvény visszatérési értéke hamis, ekkor az eredeti függvény azaz a save visszatérési értéke hamis lesz. Tehát ha létezik statDir névű mappa akkor kilepünk a feltételből, ha nem létezik akkor létrehozunk egyet, ha valami hiba van létrehozáskor például nincs elég tárhely akkor a második feltétel is igaz és kilep a függvényből hamis igazságértékkel. Lenyegeben az mkdir visszatérési értéke is meghatározná a feltétel igazságértékét az exec függvény használatával. Majd létrehozunk a QString osztályhoz egy objektumot name néven tehát egy sztringet mellynek értéke statDir/Test-1 ha a save függvény parametere egy volt. A StatDir ugye sztring típusú mivel a QDir parametere QString osztálybeli sztring, a számot pedig a QString osztály number függvényével határozzuk meg mivel változóba van mentve. Az összeadás jel pedig az egybeírást jelenti, az időjelek közötti szöveg pedig sztring típusú. Létrehozunk egy QFile osztálybeli objektumot file néven mellynek konstruktorában megadhatjuk a fájl nevet ami jelen esetben statDir/Test-1-screenimage.png lenne ha a save függvény parametere egy. A file.open függvénnyel megnyitjuk a file objektumhoz a fájlkezelést csak írás módban, ha sikerül akkor a visszatérési érték igaz, ha nem akkor hamis. Majd ha teljesül a feltétel akkor a ret változó igazságértéke amit a pixmap objektumra meghívott save függvény visszatérési értéke határoz meg. A save függvény parameterben megadjuk a fájl memoriacimmet amit mentünk majd a kiterjesztést ami PNG lesz. Sikeres mentés esetén a ret változó igazságértéke igaz lesz, sikertelen mentés esetén hamis. Létrehozunk egy tfile névű objektumot a QFile osztályhoz aminek a konstruktorával megadjuk a fájlnevet ami statDir/Test-1-stats.txt ha a save függvény parametere egy. A ret változó értéke a tfile objektumra meghívott open függvény visszatérési értéke lesz ami igaz ha sikeresen létrejön az interfész csak írás és a sorvegi kifejezések is értelmeződnek például a \n újsort írás. Ha a ret igazságértéke igaz akkor teljesül a következő if feltétel mellyben létrehozunk a QTextStream osztályhoz egy objektumot textStream néven az osztály parametizált konstruktorával mellyben megadjuk a memoriacimmet hogy hova írunk majd ez jelen esetben a tfile objektum memoriacime. Majd a szöveget a textStream objektumba irányítva a fájlba írjuk a szükséges vektorok, változók értékeit és függvények visszatérési értékeit. Majd a tfile objektumra meghívva a close függvényt bezárjuk a csatornát illetve ha nem teljesül az if feltétel. Majd a save függvény visszatérési értéke a ret változó értéke azaz hibamentesség esetén true. Majd definiáljuk a slot függvényeket az updateHeroes függvényt mellynek paraméterei egy QImage konstans objektum memoriacime és két egész típusú konstans változó memoriacime. Majd az endAndStats függvényt egy egész típusú konstans változó memoriacimevel. A BrainBThread.h fájlban létrehozunk a Hero osztályt, majd típusdefiniálunk egy Hero osztály típusú elemeket azaz objektumokat tartalmazó vektort Heroes névvel így ilyen típusú vektorokat Heroes vektor név parancssal hozhatunk létre. Majd definiáljuk a Hero osztályt mellyben deklaráljuk a használatos egész típusú változókat és inicializálunk is egyet majd definiálunk egy sztringet name néven. Majd a Hero osztály parametizált konstruktorát definiáljuk a hozzátartozó inicializált listával, majd defini-



aljuk a Hero osztály destruktort. Majd definíáljuk a move függvenyt melyben a parameterul adott változók feltételeben módosítjuk az x és y változók értékeit random számok generálása segítségével, melyet a rand függvény tesz lehetővé. Majd létrehozunk a BrainBThread osztályt a QThread osztály alosztályaként, majd definíáljuk a Q\_OBJECT macrot a szignálok használatához. A programban cv osztálybeli objektumokat készítünk, a Mat osztályhoz tartozó parametizált konstruktorral létrehozunk egy két dimenziós tömböt, a konstruktornak az első értéke a matrix szélessége, a második paramétere a matrix magassága, a harmadik paramétere a matrix típusát jelen esetben CV\_8UC3 azaz három csatornás nyolc bites egész típusú elemekből álló matrix amit az RGB szinkódokhoz használunk, a negyedik paraméter a matrix nevet jelenti. A long típusú változók is egész típusúak, de nagyobb biten tárolódnak. A for ciklusban hogy végigmegyünk a vektor elemein deklarálnak egy vektor elemet a vektor típusal majd : vektor neve kódot használjuk, ugye jelen esetben a típusa Hero és létrehozunk egy elemet így objektumot a Hero osztályhoz hero néven. Az end jelet azért használjuk a vektor elem előtt mert az objektum memóriacímre szükséges. A ? : operator használatkor ha a ? előtti kifejezés igazságértéke igaz akkor a : előtti utasítás hajtodik végre, ha hamis akkor a : utáni utasítás. Definíáljuk a lostV függvenyt mely a lostBPS névű vektort egy új egész vektor típusú memóriacímrel látja el mivel a függvény visszatérési értéke referencia egy egész típusú vektorra ami automatikusan merterzi magát. A függvény használata azért szükséges mert egy másik osztályból hívjuk az adott osztály privat részében definíált vektort, melynek értékeihez a hozzáférést a publikus részben definíált függvény teszi lehetővé új memóriacímrel. Definíáljuk a decComp függvenyt melyben a push\_back függvény segítségével a parameterul adott változó értéket betesszük a vektorba melyre meghívtuk a függvenyt egy új létrehozott utolsó elemként. A popback függvény segítségével a vektornak melyre meghívtuk kiveszük az utolsó elemet. A decComp függvényben ellenőrizzük egy if feltétel segítségével hogy a heroes vektor mérete nagyobb-e mint egy, ha nagyobb akkor kiveszünk egy elemet a vektorból. Majd a hero vektor minden elemére ellenőrizzük egy if feltétel segítségével hogy a Hero osztály hero objektumának agility változóba tárolt értéke nagyobb-e egyenlő-e mint öt, ha teljesül a feltétel csökkentjük kétfelé az agility változó értékét, ugye a Hero osztály objektumai a Heroes vektor elemei. A void típusú függvényekben mivel a függvényeknek nincs visszatérési értéke, így a return szót a függvényből való kilépésre használhatjuk if feltétellel, ha nem a program végén akarunk kilépni, úgy-e a program végéről elhagyható. A cv osztály Point alosztályának konstruktorával létrehozhatunk két dimenziós pontokat objektumként, az első paraméter a pont x koordinátája, a második a pont y koordinátája. A cv osztály circle függvényével köröt rajzolunk az adott paraméterek alapján, az első paraméterben megadjuk a köröt ahova rajzoljuk a köröt ami egy InputOutputArray osztálybeli objektum, jelen esetben Mat osztálybeli ami InputOutputArray osztályhoz tartozik, a második paraméterként megadjuk egy Point osztálybeli objektumot ami a kör középpontja lesz, a harmadik paramétere egész típusú melyben megadhatjuk mennyi pixel legyen a kör sugara, a negyedik paraméterben megadhatjuk konstans típusú Scalar osztálybeli objektum mely megadja a kör színét, az ötödik paraméterben megadhatjuk a kör vonalának a pixel vastagságát egész típusú értékkel, mely ha negatív a kör kitöltött lesz, jelen esetben a CV\_FILLED változó értéke -1 így a kör kitöltött lesz, a hatodik paraméterként a kör rajzolásához használt vonal típust adhatjuk meg egész típusú értékkel, hetedik paraméterként pedig megadhatjuk a frakcióbitek számát a középpontban és a sugarban egész típusú értékkel. A cv osztály Scalar alosztályának konstruktorával létrehozhatunk a paraméterként adott elemekből egy objektumot az adott néven. A cv osztály rectangle függvényével egy téglalapot rajzolhatunk, melynek első paraméterben megadhatjuk a circle függvényhez hasonlóan hova rajzoljon, a második paraméterben megadhatjuk a téglalapnak egy csúcsát egy Point osztálybeli objektummal, a harmadik paraméterben a második paraméterként megadott csúccsal szemközti csúcsot adjuk meg egy Point osztálybeli objektummal, a negyedik paraméterben megadhatjuk a téglalap színét egy Scalar típusú objektummal, a többi adatnak van alapértéke így nem szükséges megadni a circle függvényhez hasonlóan. A cv osztály putText függvényével megadhatjuk szöveget írhatunk alakzatokra, melynek első paramétere a circle és rectangle függvényekhez hasonlóan a kör ahova írunk, a második paraméter egy String típusú objektum ami megadja a rajzolando szöveget, a harmadik paraméterben meg-

adjuk hogy egy Point típusu objektum segítségével hogy a kepen a rajzolando szoveg balalso sarka hol van, a negyedik parameterben megadjuk a betutipust ami jelen esetben normal simplex, otodik parameterkent megadjuk double típusu szam ami a betumerethez kell, hatodik parameterkent megadjuk a szoveg szinet Scalar osztalybeli objektummal, a hetedik parameterkent megadjuk a vonal vastagsagot a rajzolashoz ami jelen esetben egy pixel, a tobbi parametert nem adtuk meg mivel az alapeterkeik megfeleloek. A cv osztaly compare fuggvenye osszehasonlitja ket tomb elemeit vagy egy tomb elemeit egy skalarral, melynek elso parametere az elso osszehasonlitando sor vagy skalar mely InputArray osztalybeli objektum melyhez tartozik a Mat osztaly melynek objektumat jelen esetben használjuk, a masodik parameterkent megadjuk az elso parameterhez hasonloan a masodik osszehasonlitando sort vagy skalar, harmadik parameterkent megadhatjuk a kimeneti sort amiben ment OutputArray objektumkent melynek resze a Mat osztaly, a negyedik parameter egesz típusu jelen esetben cv osztaly CMP\_NE mely ellenorzi hogy az elso sor eleme nem egyenlo e a masodik sor elemével elemenkent, ha nem egyenlo akkor 255-ot ir a kimeneti sorba. A cv osztaly extartctChannel fuggvényének segítségével létrehozunk egy egy dimenzios sort az elso parameterkent adott tombbol es mentjuk a masodik parameterkent megadott sorba a harmadik parameterkent megadott sor index alapján. A cv osztaly countNonZero fuggvénye visszateresi ertekekent megadja a nulla erteke elemek szamat a parameterul adott tombben. A BrainBWin.cpp fajlban a const QString BrainBWin::appName = "NEMESPOR BrainB Test"; paranccsal kezdoerteket adunk a BrainBWin osztaly appName valtozojanak ami ugye egy QString osztalybeli objektum ami konstans típusu es mivel static is ily a memoriaba van a program futasa alatt az adott osztalyhoz tartozoan es a konstruktor nem peldanyositja objektum létrehozaskor. A QDate osztaly currentDate fuggvényének visszateresi erteke az aktualis datum a rendszer ideje alapján, melyet a toString fuggvénnyel sztringge konvertalunk mivel a valtozo amibe mentjuk sztring típusu. A QDate osztaly currentMSecsSinceEpoch fuggvényének visszateresi erteke az 1970.01.01 00:00:00 ota eltel miliszekundumok szama, melyet a QStrin osztaly number fuggvényével alakitunk sztringge. A qDebug fuggvény kiirja a szabvanyos kimenetre a beleiranyított sztringet. A QPixmap fromImage fuggvényével egy QImage objektumot atkonvertalunk QPixmap objektumma, mivel most kesz kepet jelenitunk meg vele, nem a pixeleket alljuk kozvetlenul. A device fuggvény visszateresi erteke az objektum amit festunk vagy 0 ha nem aktiv a paintevent, mivel jelen esetben a fuggvenyt a paintevent-ben definialtuk ily mindig aktiv lesz, akkor aktiv mikor létrejön a QPainter objektum es mig meg nem hivodik az létrejott objektumra az end fuggvény. A QPainter osztaly drawText fuggvénye az elso es masodik parameterkent adott x es y koordinatakra irja ki a hamradik parameterkent adott QString objektumot. A QPainter osztaly drawPixmap fuggvénye kirajzolja az elso es masodik parameterkent adott x es y koordinatakra a harmadik parameterkent adott QPixmap objektumot. A mouseevent-ben hasznalt pos fuggvény az eger poziciojat adja meg a kepen amire használjuk az eventet, az x es y fuggvények pedig az x es y koordinatait a pozicionak. A key fuggvény a keyevent-ben hasznalt fuggvény melynek visszateresi erteke egesz típusu szam ami a lenyomott billentyu azonositoja, ezeket a QT osztaly KEY strukturaja tartalmazza. A BrainThread.cpp fajlban az std::time(0) fuggvény megadja hany miliszekundum telt el a program futtatasa ota, ily minden meghivaskor mast kapunk. Az std::srand fuggvény a parameterkent megadott ertekekkel inicializalja az std::rand fuggvenyt, ami a RAND\_MAX generalt maximum erteke es az srand erteke kozt general egy random szamot. Ha nulla es egy kozt generalunk random szamot hatekonyan akkor ezutan leosztjuk a kapott szamot a RAND\_MAX+1.0 kifejezessel mivel a rand fuggvény által generalt fuggvény kisebb mint a RAND\_MAX ily a szam kisebb mint egy es nagyobb mint nulla mivel pozitivakat generalunk. A +1.0 azert szukseges hogy double típusu osztassal szamoljunk, mert eredetileg egesz szamokkal szamol ily minden esetben az eredmény nulla lenne, ily viszont egy nulla es egy kozotti lebegopontos szam. Az QThread osztalybeli msleep fuggvény altatja az aktualis QThread objektumot a parameterul adott miliszekundumig.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 9. fejezet

# Helló, Chaitin!

### 9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

### 9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

## 9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9.6. Omega

Megoldás videó:

Megoldás forrása:

DRAFT

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Juhász István - Magas szintű programozási nyelvek 1; olvasó-napló

A számítógépes nyelveknek megkülönböztetjük a szintjeit, leteznek gépi nyelvek és magas szintű nyelvek. Minden processzornak van egy gépi nyelve, melyen végrehajtja az utasításokat, azonban az általunk írt magasszintű nyelveken írt kódon írt utasításokat a processzor nem tudja végrehajtani mivel nem gépi nyelven van írva, így át kell alakítanunk gépi nyelvvé. A magasszintű nyelvek például a C, melyek saját szintaktikai és szemantikai tulajdonságokkal rendelkeznek. A magasszintű nyelvek gépi nyelvekké alakítására két módszer van az egyik a fordítóprogramos a másik az interpretés. A fordítóprogramos átalakítást használjuk mivel ellenőrzi a kódot szintaktikailag és szemantikailag mielőtt lefordítana, ilyen például a C nyelvhez a gcc, melynek több változata ismert c90, c11, melyeknek neveiben az eljáratra utalnak a számjegyek. A fordítóprogram készíti tárgyprogramot, ha szintaktikailag helyes a kódunk, melyet a gcc fordítónál az -o kapcsolóval nevezhetünk el. Az interpretés fordítással a kód rögtön lefut az ellenőrzések után, mivel nem készült tárgyprogram amit később is futtathatunk. A programozási nyelveket két főbb osztályba sorolhatjuk az egyik az imperatív nyelvek, melyekbe algoritmusok segítségével írjuk meg az utasításokat különböző változókat felhasználva és van lehetőségünk memóriafoglalásra is, ezeket gyakrabban használjuk ilyen például a C nyelv. Vannak a deklaratív nyelvek amelyek nagyjából az imperatív nyelvekkel ellenkező tulajdonságokkal rendelkeznek, presze ezeken kívül leteznek még más nyelvű nyelvek amelyek nincs egyseges jellemzőjük. Minden programnyelvnek saját karakterkészlete van mely betűkből, számokból és egyéb karakterekből épül fel. A számok általában egyseges decimális számok. A betűkkel eltérő lehet hogy külön kezeli a kis és nagybetűket mint például a C nyelv. A speciális karakterek közül a szóköz, tabulatort, enter nem különbözteti meg a fordító, ezeket egysegesen white space-knek nevezzük, tehát mindegy melyiket használjuk szó alkotásra szintaktikai szempontból, azonban a jobb átlathatóság miatt használjuk mindegyiket. A forrásprogramban előforduló lexikális egysegeket a lexikális elemzés során a fordító felismeri és tokenizálja azokat, ilyenek a többkarakteres szimbólumok például C nyelvben a ++, a szimbolikus nevek amiket a programozó hoz létre az egyes elemek azonosítására, ezeknek a neveinek betű karakterekkel kell kezdődniük és nem tartalmazhatnak speciális karaktereket például műveleti jeleket és a kulcsszók például az if melynek a nyelv tulajdonít jelentést, megjegyzések melyek segítik a program értelmezést az olvasó számára, a címek melyekkel a címezett utasításra a program egy másik részéből hivatkozni tudunk, literálok melyek megadják a változó típusát például a double. Vannak kötött és szabad formátumú nyelvek, a kötött formátumú nyelvekben egy sor egy utasítás ha nem fejt ki egy sorba külön jelezni kellett a sorvége torlással és a programelemeknek is meghatározott helye van a sorban. A szabad formátumú elemekkel egy



sorban akárhany utasítás kerülhet és tetszőleges pozíciókban használhatjuk a programelemeket, az utasítások véget pedig pontosvesszővel jelezzük. Az eljárásorientált nyelvekben a lexikális egységeket a megfelelő módon el kell választanunk egymástól whitespace-szel vagy elhárító jelekkel például zárójellel. A kifejezések segítségével egy adott értékből, operandusból új értéket határozunk meg operátorok segítségével, ehhez használhatunk zárójelzést is hogy meghatározzuk a műveleti sorrendet. A ketoperandusú kifejezések alakjait az operátor elhelyezkedése határozza meg, ha elöl van akkor prefix alakú, ha közepén infix, ha az operandusok mögött akkor postfix. A műveletek megfelelő sorrendbeli végrehajtása után a kifejezésnek megkapjuk az új értéket és hozzárendelődik a típusa, ezt a kifejezés kiértékelésének nevezzük. A műveletek végrehajtási sorrendje lehet balról-jobbra, jobbról-balra, balról-jobbra a precedenciatablázat segítségével. Mivel az infix alakú operátorai nem azonos erősségűek így az infix alak nem egyértelmű, ehhez használjuk a precedenciatablázatot, melynek műveleti sorrendjét felülírhatjuk zárójelek használatával, a teljesen bezároljelezett alaknak egy műveletvégrehajtási sorrendje van. A kifejezés típusának meghatározására kétféle módszer van, az egyik a típuségyenlőség ekkor a ketoperandusú kifejezés mindeket értéke azonos típusú vagy az operátor határozza meg a típust. Két programozási eszköz típusa akkor azonos ha egy utasításban deklaráltuk és azonos típussal és a két eszköz összetett típusú, szerkezetük megegyezik. A típuskenyszerítés módszerével a ketoperandusú kifejezéseknek különböző típusú operandusai lehetnek, a műveletek viszont csak azonos típusú operandusokkal végezhetőek el így annak megfelelően átkonvertálódnak. A típuskenyszerítésnek van bővítéses fajtája amikor a konvertálando típus elemei elemei a céltípusnak is, szükséges esetén viszont nem elemei és ekkor értékcsontítás vagy kerekítés történik. Az utasítások alkotják az algoritmusok egyes részeit, illetve a fordítóprogram ennek segítségével generalja le a tárgyprogramot, ezek alapján két csoportjuk van deklarációs és végrehajtó utasítások. A deklarációs utasítások információt szolgáltat a tárgykód generalásához, a végrehajtó utasításokat pedig lefordítja tárgykóddra. A végrehajtó utasításokat kilenc csoportba sorolhatjuk. Az értékadó utasítás segítségével értéket adhatunk változóknak a program futása alatt. Az üres utasítás segítségével a program egy üres gépi utasítást hajt végre, ez szintaktikailag hasznos. Az ügros utasítás segítségével egy adott címkevel ellátott utasításnak adhatjuk át a vezérlést a GOTO parancs segítségével. Az elágaztató utasítás segítségével a program több tevékenység közül használja a megfelelőt. A ciklusvezérlő utasítás segítségével a program egy adott tevékenységet többször is végrehajthat, akár végtelenszer is. C nyelvben a CONTINUE utasítás az aktuális ciklusszalból kilep nem hajtja végre a további utasítást, hanem újabb szálba kezd.

## 10.2. Kernighan és Richie; olvasónapló

A végrehajtando műveletek sorrendjét a vezérlésadó utasítások adják meg. A kifejezések végére pontosvesszőt teszünk hogy utasításként működjenek, egyébként fordításkor szintaktikai hibat kapnánk. Összetett utasításokat létrehozhatunk, ha kapcsos zárójelek közé adunk meg több utasítást, ezek végére nem kell pontosvessző. Az if-else utasítás a paraméterként kapott kifejezés értéke alapján dönti el hogy végrehajtja-e a hozzárendelt utasítást, ha egy az értéke akkor végrehajtja, nulla érték esetén viszont nem hajtja végre, hanem ha adunk meg else ágat annak az utasításat hajtja végre. Az if illetve az else végrehajtja az utána következő utasítást vagy utasításokat amit úgy kapcsoszárójelben kell megadnunk. Az if utasításokat egymásba is ágyazhatjuk egymásból következő feltetelek esetén, kapcsoszárójel használata nélkül az else mindig a hozzá legközelebbi else-if-hez tartozik. A kapcsos zárójel használatával a beágyazott if utasítások esetén az else ág melyik if utasításhoz tartozását is meghatározhatjuk. Egy if utasításhoz egy else ág tartozhat ami hatékonyabban működik mint két if utasítás használata. Az else-if utasításokban megadunk egy kezdő if utasítást majd akárhany elseif utasítást megadhatunk melyek egymás után kiértékelődnek és ha az egyik értéke egy akkor végrehajtja a hozzárendelt utasítást és kilep az else-if utasításból. Ha az if és minden else-if ág értéke nulla akkor ha van else ág akkor az ahhoz rendelt utasítás hajtódik végre, ezt

hasznaljuk hibakezeleskent is. A switch utasitast parameterekent megadunk egy kifejezest majd egy osszetett utasitasban megadjunk akarhany case `` : agat majd egy default: agat. A case `` reszeben megadunk egy allando erteku kifejezest melynek erteke ha megegyezik a switch parameterekent adott ertekkel akkor vegrehajtja a case : utan megadott utasitast. Ha egyetelen case eseteben megadott kifejezes erteke sem egyezik meg a switch parameterkent adott ertekével akkor a default aghoz rendelt utasitas hajtodik vegre. A while utasitas a parameterekent megadott kifejezes erteket ellenorzi hogy egy e, ha igen akkor vegrehatja a hozzarendelt utasitast, majd ismet ellenorzi es egy ertek eseten ujra vegrehatja az utasitast, majd ezt addig hajtja vegre rekurzivan amig a kifejezes erteke nem lesz nulla. Ha a parameterkent adott kifejezes erteke egy akkor vegtelenszer hajtja vegre a while a hozzarendelt utasitast. Ha a parameterkent adott kifejezes tartalmaz valtozot, annak erteket modositjak a while utasitashoz rendelt utasitasok. A for utasitasnak harom kifejezest adhatunk meg parameterul, melyek kozul az elso kezdorteket ad a masodik kifejezesben vizsgalt operandusnak, a masodik kifejezes operator segitsegevel eldonti nem nulla vagy nulla az aktualis erteke a kifejezenek, a harmadik kifejezzel pedig modsitjuk az elso parameterkent megadott valtozo erteket. A kifejezesek elhagyasaval es a harom pontosvesszo hasznalataval vegtelenszer fogja vegreahajtani a for utasitas a hozzarendelt utasitast. Minden for utasitast atirhatunk while utasitassal ugy hogy a for ciklus parameterekent megadott elso kifejezest a while ciklus elott definialjuk, a masodik kifejezes a while utasitas parametere lesz, a harmadik kifejezest pedig a while utasitashoz rendelt utasitasban definialjuk. A do-while utasitas a do utasitashoz rendelt utasitast hajtja vegre majd ellenorzi a while utasitas parametereul adott kifejezes erteket hogy nulla e ,ha nem ujra vegreahajtja a do utasitashoz rendelt utasitast majd ismet ellenorzi a while utasitas parametereul adott kifejezes erteket hogy nulla e, ezt hajtja vegre rekurzivan amig a while utasitas parametereul adott kifejezes erteke nulla nem lesz. A for es a while utasitassal ellentetben nem az utasitas vegreahajtasa elott ellenorzi hogy vegreahajtja e vagy nem, hanem eloszor vegreahajtja az utasitast majd ellenorzi hogy vegreahajtja e ujbol. A while ciklushoz hasonloan itt is az utasitas reszben valtozhat a valtozo erteke, amit operanduskent hasznal a while utasitas parameterekent megadott kifejezes. A do utasitashoz rendelt utasitast celszeru kapcsoszarojelben hasznalni hogy biztosan a while utasitashoz tartozzon. A break utasitas segitsegevel kilep a for, while, do-while, switch utasitasokbol a lefordulasakor, tehat nem addig hajtodik vegre amig a kifejezes erteke nulla nem lesz hanem addig amig a break utasitas le nem fordul. A countinuu utasitas a for, while, do-while utasitasokhoz tartozo utasitasbol kilep es ismet a parameterkent adott kifejezes vagy kifejezesek erteke alapjan donti el hogy ujra lefut e az ugye for vagy while utasitashoz rendelt utasitas. Tehat a for es a while utasitasokhoz rendelt utasitasoknak az utasitasai a countinuu utasitas lefutasaig hajtodnak vegre, nem a teljes utasitas utolso utasitasaig. A goto utasitast a tobbszorosen egymasba agyazott utasitasokbol valo kilepesre hasznaljuk ugy hogy az adott utasitasban megadott goto cimke nevu utasitas vegreahajtodasakor a cimke nevu cimkehez tartozo utasitasra ugrik a program. Cimket a cimkeneve : utasitas parancsal hozhatunk létre. A goto utasitas hasznalataval csak az adott fuggvenyben cimkezett utasitas erhető el. A goto hasznalata kikerulhető megfelelő segéd utasítások hasznalatával.

### 10.3. BME: Szoftverfejlesztés C++ nyelven / Benedek Zoltán, Levendovszky Tihamér ; olvasónapló

A könyvtárakban van definialva a hiba, melyeket megfelelően kezelhetünk, az ilyen definialt hibákat kiveteleknek azaz exception-nak nevezzük. A kivételkezesles ugy tortenik hogyha egy fuggveny nem tud kezelni egy hibát kivetelt dob ra a throw fuggvennyel melyet a catch fuggvennyel elkaphatunk ha megfelelő kivetelt definialtunk. Hibakezelesnel nelkul a program hiba eseten leall vagy hibasan mukodik. A C++ programoknak az alapveto hibakezelese hogy hiba eseten a program leall, kiveve ha a el nem kapjuk

az összes hibát és megfelelően kezeljük hogy ne alljon le. A C programoknál viszont tovább fut hibásan is, és így csak később derülhet ki a hiba esetleges leállás esetén. A kivételkezelés a szinkron kivételek kezelésére szolgál például a ki és bemeneti hibák, az aszinkron eseményeket például a billentyűzet felől érkező hibát szignálokkal kezeljük. A kivétel elfordulást egy osztállyal írjuk le melynek egy objektuma a kivétel. Van úgynevezett végrehajtási verem mely a throw függvény meghívásával tekeredik vissza a catch függvényig. A kivételek osztályait hierarikusan kezeljük hogy egyszerűbben tudjuk módosítani azokat. A kivételt csak akkor tudjuk elkapni ha a catch parameternek típusa megegyezik a throw-val meghívottnak, vagy ha a catch parametere főosztálya a throw-val meghívottnak, vagy ha a catch parametere és a throw-ra meghívott is mutatótípus melyeknek vagy megegyezik a típusuk vagy a catch parametere bazisosztálya a throw-val meghívottnak, vagy ha a catch parametere referencia és teljesül rá hogy megegyezik a típusa a throw-val meghívottnak vagy bazisosztálya a throw-val meghívottnak. A const szó használatával a kivételek értéket nem változtathatjuk és minden kivétel másolható. Ha nem tudjuk kezelni teljes egészében az elkapott kivételt akkor továbbdobhatjuk a throw utasítás lefuttatásával, ekkor nem az elkapott kivételt dobjuk tovább hanem az eredeti throw által dobottat. A try függvényben definiáljuk a catch függvényeket, melyeknek a sorrendje osztály hierarchia szerint növekvő sorrendben kell legyen mivel ekkor az alosztályok nem kapodnának el. Fájlkezelésnél a megnyitott fájlt nem mindig zárul automatikusan, mivel ha a hiba az fclose meghívódása előtt van akkor a hibát kezelhetjük a use\_file() függvény visszatérési értéket ad a hibat felismerő függvénynek, de az fclose függvény már nem hívódik meg így a lefoglalt memóriafolyik. Ezt a hibát javíthatnánk úgyis hogy a catch függvénnyel elkapunk minden kivételt és meghívjuk benne az fclose függvényt, de ez nagyobb kódoknál bonyolultabb így osztályban definiáljuk a fájlkezelést melynek a destruktora elvégzi a megfelelő memóriafelszabadítást a kivételkezeléstől függetlenül. A használatos objektumokat lokális objektumként hozzuk létre a konstruktor inicializációs listájával, mivel a konstruktor segítségével létrehozott objektumot törli a destruktork és az így létrehozott objektum teljesen jön létre megfelelően megírt konstruktor esetén. Az inicializációs lista használatával a fordító ellenőrzi hogy a létrehozott objektumok létrehozása közben kivétel adódik akkor csak a létrehozott objektum torlódik, így nem szükséges a konstruktor irashoz kivételkezelő kód. Ha csak egy objektum rész jön létre mivel kivétel van a konstruktor parameterben akkor nem jön létre teljesen az objektum így a destruktork nem törli, ehelyett inicializációs lista segítségével vektort használunk az objektum memóriakezeléséhez. Az std könyvtárban van auto\_ptr kisebbkacsacsor nagyobbkacsacsor sablon mely a kisebbkacsacsor nagyobbkacsacsor között megadott típussal hoz létre mutatót az adott objektumra, hasonlóan működik a hagyományos mutatókhoz, de torlésekor megsemmisül az általa mutatót objektum is. Az auto\_ptr egy másik mutatóba másolással a mutató nem mutat semmire és az auto\_ptr-eket a másolás megváltoztatja, de használhatjuk rá a const kulcsszót melynek segítségével nem lesz másolható. Ezt az auto\_ptr\_ref akadályozza meg és megvalósítja a másolását. Nem std típusokkal való másolása például vektorba másoláskor sérülés veszély áll fenn. Az auto\_ptr által mutatót objektum torlódik a kivételkezeléstől függetlenül. Elhelyező utasítás használatával kivételkezeléskor is felszabadul a new által lefoglalt hely a megfelelő delete parancsokkal automatikusan, más módon lefoglalt memóriát nem szabadít fel automatikusan. Ha elfogy a memória a memóriaszivargások miatt akkor kivételkezeléssel és egy végtelen ciklussal használt malloc függvény segítségével kereshetünk memóriacímeket, melyeket a túlterhelt new függvény ad vissza, ha nem talál hibakezelést használunk. A bad\_alloc kivételt melynek működéséhez marad memória, és a new\_handler-t hívjuk meg előtte ha nem talál memóriát a malloc. A konstruktoroknak alapvetően nincs visszatérési értékük így hibás állapotú objektumot hozhatunk létre vagy globális változóval ellenőrizzük az objektum létrehozás sikerességét vagy ne hozunk létre kezdeti értéket vagy kezdeti érték helyett egy tagfüggvény adjon értéket mely jelenti a hibát. Ezeknek a felhasználói ellenőrzésével kivételkezelhetjük a konstruktorokat. A destruktork meghívódhat a verem visszatérése közben ha elhagyunk egy megfelelő objektumot. Ha egy kivételt nem kapunk el akkor a terminate függvény kerül meghívásra vagy ha a kivételkezelő eljárás a vermet hibásnak találja vagy egy delete kivétel kiváltással próbál véget érni. A terminate függvény meghívódása meghívja az abort függ-

venyt amely kilep a programbol nem feltetlenul szabalyosan es az exit fuggvennyel ellentetben ennek nincs visszateresi erteke arrol hogy szabalyosan allt e le a program vagy nem. Egy elakapott kivétel nem mindig jelenti a program hibáját.

## **10.4. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven es Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.**

A C++ es a Java egyarant objektum-orientált programozási nyelv, megis mind a C++, mind a Java programozási nyelv sok szempontból különbözik egymástól. A C++ programozási nyelv alapozza mind az eljárás, mind az objektum-orientált programozást, így módon hibridnek nevezik. A C++ es a Java programozási nyelvek eltérő tervezési célokkal készültek. A C++ programozási nyelv alkalmazások es rendszerek fejlesztésére került bevezetésre es a C programozási nyelv kiterjesztése. A C++ programozási nyelv az eljárási nyelv tulajdonságai mellett támogatást nyújtott az objektum-orientált programozási szolgáltatásokhoz, a kivételkezeléshez, az általános programozáshoz. A Java programozási nyelv értelmező funkcióval rendelkezik olyan nyomtatási rendszerek számára, amelyek később támogatták a hálózati számítás. A Java programozási nyelv egy virtuális mechanizmusra épül, amely nagyon biztonságos és hordozható a programozási környezetben. A Java programozási nyelv csoportosítva van egy széleskörű könyvtárral, amely a meglevő platform absztrakciójának támogatása érdekében van megvalósítva. A Java programozási nyelv fejlesztésének fő célja egy könnyen használható es széles körben elérhető programozási nyelv kifejlesztése volt. A C++ es a Java programozási nyelvek is statisztikailag tipizált objektum-orientált programozási nyelvek, hasonló inkompatibilis szintaxissal rendelkeznek. A Java programozási nyelv kiterjedt dokumentációval is rendelkezik, amely Javadoc néven ismert. A C++ programozási nyelv mutatókat használ, míg a Java programozási nyelv nem tartalmaz mutatókat. A Java a „korlátozott mutatók” fogalmát használja. A korlátozza minősítő alkalmazható egy adatmutatóra annak jelzésére, hogy a mutató deklarációjának hatálya alatt az összes rajta elérhető adat csak az adott mutatón keresztül, de más mutatón keresztül nem érhető el, így lehetővé teszi a fordítónak bizonyos optimalizálások elvégzését azon feltevés alapján. A C++ programozási nyelvben a program fut es a fordítás a fordító használatával történik. A C++ programozási nyelv fordítója konvertálja a forráskódot gépi szintű nyelvre, ami a C++ programozási nyelvet platformfüggetlen nyelvvé teszi. A Java programozási nyelvben a java forráskódot először bajtkóddra konvertálják a fordításkor. Ezt a bajtkódot azután az értelmező futási idő alatt értelmezi, hogy előallítsa a kimenetet, ez teszi a Java programozási nyelvet platformfüggetlen nyelvvé. A C++ programozási nyelv nem nyújt beépített támogatást a szálakhoz, míg a Java programozási nyelv támogatja a szálakat implicit módon. Noha a C++11 programozási nyelv legújabb megvalósításában támogatja a szálakat is. A Java programozási nyelv nagyjából hasonlít a C++ programozási nyelvre, de nem foglal magában olyan összetett fogalmakat, mint a struktúrák, az operátorok túlterhelése, mutatók, sablonok, egyesítések stb. A Java programozási nyelv szinten nem támogatja a feltételes fordítást (# ifdef / # ifndef típus). A Java programozási nyelvben minden entitás a java tárgya, az alaptípusok kivételével. A Java programozási nyelv egyetlen gyökerhierarchiával rendelkezik, mivel minden a java.lang.Object fajlból származik. A C++ és Java programozási nyelvek támogatják az OOPS koncepciókat. A C++ programozási nyelv rugalmasságot nyújt futási időben es széles típusú hierarchiakat tud végrehajtani. A C++ programozási nyelv a C programozási nyelvre épül es a funkcióival visszafelé kompatibilis. A C++ programozási nyelv egy egyfajta alacsony szintű programozási nyelv néhány magas szintű funkcióval kiegészítve. A C++ programozási nyelv memóriakezelése egy kezi folyamat, amelyet a programozónak kell kezelnie, amely előidezheti a memóriaszivargások es a szegmentációs hibák

kockaztat. A Java programozási nyelv beépített hulladékgyűjtő mechanizmussal rendelkezik amely nyomon követi az objektumok számára kiosztott memóriát és automatikusan felszabadítja azokat amikor már nem használják. A Java programozási nyelv különböző primitíveket és objektumtípusokat kínál és erősen tipizált programozási nyelv, ez lehetővé teszi a primitívumok megfelelő objektumtípusokká történő átalakítását, például egész számú objektumra egész osztály objektum felhasználásával stb. A Java programozási nyelv automatikus polimorfizmust biztosít és korlátozhatja azt az explicit módszer felbírálatainak megtiltásával. Mind a C++, mind a Java programozási nyelv rendelkezik hozzáférési specifikátorokkal amelyek korlátozzák az attribútumok és metódusok hatókörét, az osztályon belül privát, a csomagon belül védett, az osztályon valamint a csomagon kívül nyilvános. A C++ programozási nyelv támogatja a mutatókat, struktúrákat, egyesítéseket, sablonokat, operátorok túlterhelését vagy a mutatók számtani használatát. Java programozási nyelvben ezek nem találhatók meg, hanem referenciák vannak amiket nyers címmel nem lehet visszafejteni és a veszélyes változtatások nem engedélyezettek. A referenciáknak típusaik vannak és biztonságosak. A C++ programozási nyelvben destruktorki meghívódik amikor egy objektum torlódik, Java programozási nyelvben a garbage collection végzi el automatikusan a memóriafelszabadítást. A C++ programozási nyelv támogatja a feltételes összeállítást és beillesztést, a Java programozási nyelv nem. A Java programozási nyelv rendelkezik a függvény túlterheléssel, de nincs operátor túlterhelés. A sztring osztály a + és + = operátorokkal használja a sztringek összekapcsolását és a karakterlánc kifejezések automatikus típuskonverziót használnak, de ez egy speciális beépített eset. A C++ programozási nyelv támogatja a technika túlterhelését és az adminisztrátorok túlterhelését. A Java programozási nyelv beépített támogatást nyújt a dokumentációs megjegyzésekhez (`/ ** ... */`); ezért a Java forrásfájlok tartalmazhatják a saját dokumentációjukat, amelyet egy külön eszköz olvas, általában Java doc és újraformázza HTML-re, ez elősegíti a dokumentáció egyszerű karbantartását. A C++ programozási nyelv nem támogatja a dokumentációs megjegyzéseket. A C++ programozási nyelv protestáló kódot hoz létre és lehet, hogy egy hasonló kód nem fut különböző szakaszokban, a Java programozási nyelv platformfüggetlen.

## 10.5. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba

A C++ és a Java egyaránt objektum-orientált programozási nyelv, mégis mind a C++, mind a Java programozási nyelv sok szempontból különbözik egymástól. A C++ programozási nyelv alátámasztja mind az eljárásit, mind az objektum-orientált programozást, így módon hibridnek nevezik. A C++ és a Java programozási nyelvek eltérő tervezési célokkal készültek. A C++ programozási nyelv alkalmazások és rendszerek fejlesztésére került bevezetésre és a C programozási nyelv kiterjesztése. A C++ programozási nyelv az eljárási nyelv tulajdonságai mellett támogatást nyújtott az objektum-orientált programozási szolgáltatásokhoz, a kivételkezeléshez, az általános programozáshoz. A Java programozási nyelv értelmező funkcióval rendelkezik olyan nyomtatási rendszerek számára, amelyek később támogatják a hálózati számítást. A Java programozási nyelv egy virtuális mechanizmusra épül, amely nagyon biztonságos és hordozható a programozási környezetben. A Java programozási nyelv csoportosítva van egy széleskörű könyvtárral, amely a megfelelő platform absztrakciójának támogatása érdekében van megvalósítva. A Java programozási nyelv fejlesztésének fő célja egy könnyen használható és széles körben elérhető programozási nyelv kifejlesztése volt. A C++ és a Java programozási nyelvek is statisztikailag tipizált objektum-orientált programozási nyelvek, hasonló inkompatibilis szintaxissal rendelkeznek. A Java programozási nyelv kiterjedt dokumentációval is rendelkezik, amely Javadoc néven ismert. A C++ programozási nyelv mutatókat használ, míg a Java programozási nyelv nem tartalmaz mutatókat. A Java a „korlátozott mutatók” fogalmát használja. A korlátozza minősítő alkalmazható egy adatmutatóra annak jelzésére, hogy a mutató deklarációjának hatálya

alatt az összes rajta elérhető adat csak az adott mutaton keresztül, de más mutaton keresztül nem érhető el, így lehetővé teszi a fordítónak bizonyos optimalizálások elvégzését azon feltevések alapján. A C++ programozási nyelvben a program fut és a fordítás a fordító használatával történik. A C++ programozási nyelv fordítója konvertálja a forráskódot gépi szintű nyelvre, ami a C++ programozási nyelvet platformfüggetlen nyelvvé teszi. A Java programozási nyelvben a Java forráskódot először bajtkóddra konvertálják a fordításkor. Ezt a bajtkódot azután az értelmező futási idő alatt értelmezi, hogy előállítsa a kimenetet, ez teszi a Java programozási nyelvet platformfüggetlen nyelvvé. A C++ programozási nyelv nem nyújt beépített támogatást a szálakhoz, míg a Java programozási nyelv támogatja a szálakat implicit módon. Noha a C++11 programozási nyelv legújabb megvalósításában támogatja a szálakat is. A Java programozási nyelv nagyjából hasonlít a C++ programozási nyelvre, de nem foglal magában olyan összetett fogalmakat, mint a struktúrák, az operátorok túlterhelése, mutatók, sablonok, egyesítések stb. A Java programozási nyelv szinten nem támogatja a feltételes fordítást (# ifdef / # ifndef típus). A Java programozási nyelvben minden entitás a Java tárgya, az alaptípusok kivételével. A Java programozási nyelv egyetlen gyökerhierarchiával rendelkezik, mivel minden a java.lang.Object fajlból származik. A C++ és Java programozási nyelvek támogatják az OOPS koncepciókat. A C++ programozási nyelv rugalmasságot nyújt futási időben és széles típusú hierarchiakat tud végrehajtani. A C++ programozási nyelv a C programozási nyelvre épül és a funkcióival visszafelé kompatibilis. A C++ programozási nyelv egy egyfajta alacsony szintű programozási nyelv néhány magas szintű funkcióval kiegészítve. A C++ programozási nyelv memóriakezelése egy kezi folyamat, amelyet a programozónak kell kezelnie, amely eloldozhatja a memóriaszivargások és a szegmentációs hibák kockázatát. A Java programozási nyelv beépített hulladékgyűjtő mechanizmussal rendelkezik, amely nyom követi az objektumok számára kiosztott memóriát és automatikusan felszabadítja azokat amikor már nem használják. A Java programozási nyelv különböző primitíveket és objektumtípusokat kínál és erősen tipizált programozási nyelv, ez lehetővé teszi a primitívumok megfelelő objektumtípusokká történő átalakítását, például egész számú objektumra egész osztály objektum felhasználásával stb. A Java programozási nyelv automatikus polimorfizmust biztosít és korlátozhatja azt az explicit módszer felbírálatainak megtiltásával. Mind a C++, mind a Java programozási nyelv rendelkezik hozzáférési specifikátorokkal, amelyek korlátozzák az attribútumok és metódusok hatókörét, az osztályon belül privát, a csomagon belül védett, az osztályon valamint a csomagon kívül nyilvános. A C++ programozási nyelv támogatja a mutatókat, struktúrákat, egyesítéseket, sablonokat, operátorok túlterhelését vagy a mutatók számtani használatát. A Java programozási nyelvben ezek nem találhatók meg, hanem referenciák vannak, amiket nyers címmel nem lehet visszafejteni és a veszélyes változtatások nem engedélyezettek. A referenciáknak típusaik vannak és biztonságosak. A C++ programozási nyelvben destruktork meghívódik amikor egy objektum torlódik, a Java programozási nyelvben a garbage collection végzi el automatikusan a memóriafelszabadítást. A C++ programozási nyelv támogatja a feltételes összeállítást és beillesztést, a Java programozási nyelv nem. A Java programozási nyelv rendelkezik a függvény túlterhelésével, de nincs operátor túlterhelés. A String osztály a + és += operátorokkal használja a stringek összekapcsolását és a karakterlánc kifejezések automatikus típuskonverziót használnak, de ez egy speciális beépített eset. A C++ programozási nyelv támogatja a technika túlterhelését és az adminisztrátorok túlterhelését. A Java programozási nyelv beépített támogatást nyújt a dokumentációs megjegyzésekhez (/ \*\*... \*/); ezért a Java forrásfájlok tartalmazhatják a saját dokumentációjukat, amelyet egy külön eszköz olvas, általában Java doc és újraformázza HTML-re, ez elősegíti a dokumentáció egyszerű karbantartását. A C++ programozási nyelv prototípust hoz létre és lehet, hogy egy hasonló kód nem fut különböző szakaszokban, a Java programozási nyelv platformfüggetlen. Mindket programozási nyelvben a program a main függvényel kezdődik, de a main függvény deklarálása eltérő. Mindket programozási nyelvnek vannak aritmetikai és logikai operátorai, aritmetikaiak a + - \* / ,relációsak a kisebb nagyobb kisebbegyenlő nagyobbegyenlő egyenlő nemegyenlő. Mindket programozási nyelvben vannak adattípusok, például float, double, a Java programozási nyelvben Boolean van, C++ programozási nyelvben pedig bool. Mindket programozási nyelvben vannak azonos kulcsszavak, például break, continue,

return, static. Mindket programozási nyelvnek hasonló a szintaktikája és mindket programozási nyelvben használhatunk többsoros kommenteket // és /\* jelek segítségével. Mindket programozási nyelv lehetővé teszi több szál egyidejű végrehajtását a multitasking elérése érdekében. Mindket programozási nyelv hasonló ciklusokat, például for while ciklusok és feltételes állításokat, például switch if-else tartalmaz.

DRAFT

## 11. fejezet

# Helló, Arroway!

### 11.1. OO szemlelet

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

```
import java.math.*;
public class PolarGenerator {

 boolean nincsTarolt = true;
 double tarolt;

 public PolarGenerator() {
 nincsTarolt = true;
 }

 public double kovetkezo() {
 if (nincsTarolt) {
 double u1, u2, v1, v2, w;
 do {
 u1 = Math.random();
 u2 = Math.random();
 v1 = 2 * u1 - 1;
 v2 = 2 * u2 - 1;
 w = v1 * v1 + v2 * v2;
 } while (w > 1);
 double r = Math.sqrt((-2 * Math.log(w)) / w);
 tarolt = r * v2;
 nincsTarolt = !nincsTarolt;
 return r * v1;
 } else {
 nincsTarolt = !nincsTarolt;
 }
 }
}
```



```
 return tarolt;
 }
}

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

```
#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
```

```
#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;
}
}
```

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

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

 for (int i = 0; i < 10; ++i)
 std::cout << pg.kovetkezo () << std::endl;

 return 0;
}
```

A módosított polártranszformációs normális generátor segítségével pszeudo-random számokat gyarthatunk, ezek nem teljesen veltelenek mert valamilyen matematikai algoritmus segítségével gyartjuk le őket. A PolarGenerator objektummal két különböző véletlen valószínű számot generalunk le a Math osztály random függvényével amit a következő publikus függvényben definíálunk, de csak egyet kapunk eredményül, a paratlanadik függvényhívásra kapjuk a másikat eredményül amit a párosadik függvényhívaskor hozunk létre és tarolunk le. A következő függvény publikus mivel egy másik számot is tarolunk így többször is meghívhatjuk a létrehozott objektumunkra, ezzel több számot generalhatunk. Jelen esetben tíz számot íratunk

ki a képernyőre a `System.out` osztály `println` függvényének segítségével. Nehány szintaktikai módosítás segítségével átírhatjuk C++-ra. A Sun-os JDK-ban is ez az algoritmus található, ez véletlenszerűen general számokat a tízes alapú logaritmus 0 és 1 közötti végtelen számaira alapozva. Minden `PolarGenerator` osztályból létrehozott objektummal a `nincsTarolt` változó értéket `false`-ra állítjuk a konstruktor segítségével hogy több objektumra is működjön a program. Ennek értéket a következő függvény módosítja az ellenkezőjére. Mivel matematikai függvényeket használunk így importálni kell a `java.math.*` osztályt.

## 11.2. Homokozó

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

```
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.FileNotFoundException;
class LZWBinaryTree
{
 public LZWBinaryTree ()
 {
 root = new Node('/');
 currentNode = root;
 }

 public void insert (char b)
 {
 if (b == '0')
 {
 if (currentNode.getLeftChild () == null)
 {
 Node n = new Node ('0');

 currentNode.newLeftChild (n);

 currentNode = root;
 }
 else
```

```
 {
 currentNode = currentNode.getLeftChild ();
 }
 }

 else
 {
 if (currentNode.getRightChild () == null)
 {
 Node n = new Node ('1');
 currentNode.newRightChild (n);
 currentNode = root;
 }
 else
 {
 currentNode = currentNode.getRightChild ();
 }
 }
}

public int getDepth ()
{
 depth = maxDepth = 0;
 recDepth (root);
 return maxDepth;
}

public double getMean ()
{
 depth = sumOfMean = numberOfNodes = 0;
 recMean (root);
 mean = ((double) sumOfMean) / numberOfNodes;
 return mean;
}

public double getVariance ()
{
 mean = getMean ();
 sumOfVar = 0.0;
 depth = numberOfNodes = 0;

 recVar (root);

 if (numberOfNodes - 1 > 0)
 variance = Math.sqrt (sumOfVar / (numberOfNodes - 1));
 else
 variance = Math.sqrt (sumOfVar);

 return variance;
}
```

```
public void recDepth (Node n)
{
 if (n != null)
 {
 ++depth;
 if (depth > maxDepth)
 maxDepth = depth;
 recDepth (n.getRightChild ());

 recDepth (n.getLeftChild ());
 --depth;
 }
}

public void recMean (Node n)
{
 if (n != null)
 {
 ++depth;
 recMean (n.getRightChild ());
 recMean (n.getLeftChild ());
 --depth;
 if (n.getRightChild () == null && n.getLeftChild () == null)
 {
 ++numberOfNodes;
 sumOfMean += depth;
 }
 }
}

public void recVar (Node n)
{
 if (n != null)
 {
 ++depth;
 recVar (n.getRightChild ());
 recVar (n.getLeftChild ());
 --depth;
 if (n.getRightChild () == null && n.getLeftChild () == null)
 {
 ++numberOfNodes;
 sumOfVar += ((depth - mean) * (depth - mean));
 }
 }
}

public void printTree (PrintWriter os)
{

```

```
 depth = 0;
 printTree (root, os);
 }

 class Node
 {

 Node (char b)
 {
 if(b == '0' || b == '1')
 value = b;
 else
 value = '/';
 };

 Node getLeftChild ()
 {
 return leftChild;
 }

 Node getRightChild ()
 {
 return rightChild;
 }

 void newLeftChild (Node gy)
 {
 leftChild = gy;
 }

 void newRightChild (Node gy)
 {
 rightChild = gy;
 }

 char getValue ()
 {
 return value;
 }

 private char value;

 private Node leftChild;
 private Node rightChild;
 };

 Node currentNode;
```

```
private int depth, sumOfMean, numberOfNodes;
private double sumOfVar;

public void printTree (Node n, PrintWriter os)
{
 if (n != null)
 {
 ++depth;
 printTree (n.getLeftChild (), os);

 for (int i = 0; i < depth; ++i)
 os.print ("---");
 os.print(n.getValue () + "(" + depth + ")\n");
 printTree (n.getRightChild (), os);
 --depth;
 }
}

protected final Node root;
protected int maxDepth;
protected double mean, variance;

public static void usage ()
{
 System.out.println("Usage: lzwtree in_file -o out_file");
}

public static void main (String[] args) throws FileNotFoundException, ←
 IOException
{

 if (args.length < 3)
 {

 usage ();

 return;
 }
}
```

```
String inFile = args[0];

if(!args[1].equals("-o"))
{
 System.out.println(args[1]);
 usage ();
 return;
}

BufferedReader input = new BufferedReader(new FileReader(inFile));

if (input == null)
{
 System.out.println("Nem létezik");
 usage ();
 return;
}
PrintWriter output = new PrintWriter(args[2]);

int b;
LZWBinaryTree bincurrentNode = new LZWBinaryTree();

while ((b = input.read ()) != -1)
{
 if ((char)b == '1' || (char)b == '0')
 bincurrentNode.insert((char)b);
}

bincurrentNode.printTree(output);

output.print("depth " + bincurrentNode.getDepth () + "\n");
output.print("mean " + bincurrentNode.getMean () + "\n");
output.print("var " + bincurrentNode.getVariance () + "\n");

output.close ();
input.close ();

return;
}
};
```



Java-ban nincsenek include-alt header file-ok, helyettük import-olni kell a megfelelő könyvtarakat, osztályokat. A konstruktoroknál a változó inicializálás a : operátorral nem működik, helyette a torzs részben {} között kell megadnunk a megfelelő kezdőértékeket. A memória felszabadítást a garbage collector végzi automatikusan így destruktorként használata nincsen. Az operátor túlterhelés helyett az insert függvényt használjuk. Egy objektum tagfüggvényére hivatkozni -> helyett . -tal kell. A függvények, változók hozzáférhetőséget külön minden függvény, változó esetén kell feltüntetni, nem blokkban írva. Pointerek helyett referencia van, így törölni kell a \* jelöléseket, mivel referenciakkal hivatkozunk az objektumokra. Az sqrt függvényt a Math objektum függvényeként használjuk. A nullptr helyett null-t használunk mivel nincsenek pointerek csak referenciák. Az ostream helyett a PrintWriter objektumot használjuk. Const kulcsszót nem használjuk mivel nincs Java-ban, ehelyett final kulcsszóval használjuk a gyökeret mivel az értéke nem fog változni a program futása során. Masolo konstruktor nincs Java-ban így nem használjuk. A parancssori argumentumok számát args.length segítségével ellenőrizzük. A parancssori argumentumokat használjuk a kódban mivel a FileReader, NufferedReader, Printwriter objektumok létrehozásához szükséges megadni hogy hova illetve honnan történjen az írás olvasás, nem csak a parancssorba irányítjuk a fájlból a dolgokat. Ellenőrizzük meg az equals függvény segítségével hogy a megfelelő parancssori argumentumként használtuk-e az o kapcsolót kimeneti fájl megadásához. A print függvény segítségével a megadott kimeneti fájlba kiírjuk a megfelelő adatokat a mélységhez, átlaghoz. Majd meghívjuk a close függvényeket a bemenet és kimenet lezárására. A webes környezet használatához telepíteni kell a tomcat szerveret, mivel a Java Servlet már elavult az általunk használatos bingeszók számára. Ennek segítségével get lekérdezéssel megkaphatjuk a binfa információit. A tomcat telepítése után a webapps mappába létrehozuk az alkalmazásunk mappáját benne a META-INF és WEB-INF mappákat, a WEB-INF mappában a classes, src, lib, web.xml mappákat. Az src mappába kerül forráskód, a classes mappába a lefordított java kód, a web.xml fájlban megadjuk:

```
<servlet>
<servlet-name>Binfa</servlet-name>
<servlet-class>com.company.Main</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Binfa</servlet-name>
<url-pattern>/get</url-pattern>
</servlet-mapping>
</web-app>
```

Megírjuk a servlet működését:

```
public class Main extends HttpServlet {
 LZWBinaryTree tree;
 String data;
 @Override
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
 data = request.getParameter("data");
 tree = new BinaryTree();
 response.setContentType("text/html;charset=UTF-8");
 for (int i = 0; i < data.length(); i++) {
 tree.addItem(data.charAt(i));
 }
 }
}
```

```
PrintWriter out = response.getWriter();
try {
 out.println("<!DOCTYPE html>");
 out.println("<html><head>");
 out.println("<meta http-equiv='Content-Type' content = 'text/ ↵
 html; charset= UTF-8'>");
 out.println("<body>");
 tree.writeOut(tree.getRoot(), out);
 out.println("
</br>");
 out.println("<p>Elemsszam atlaga " + tree.getElemsszamAtlag() + " ↵
 </p>");
 out.println("<p>Atlag " + tree.getAtlag() + "</p>");
 out.println("<p>Melyseg " + tree.getMelyseg() + "</p>");
 out.println("<p>Szoras " + tree.getSzoras() + "</p>");
 out.println("</body>");
 out.println("</html>");
} finally {
 out.close();
}
}
```

A szerver indítása előtt le kell fordítani a java kódot a -d kapcsoló segítségével, hogy mentse a classe mappába a fájlt, a -cp pedig használja a szerver lib mappájában lévő servlet-api.jar csomagot. A szervert a bin mappából indítjuk a startup.sh fájl segítségével.

## 11.3. Gagyi

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

```
public static void main(String[] args){
 Scanner sc = new Scanner(System.in);
 System.out.println("Kilepeshez irja: x");
 //Integer t = new Integer(127);
 //Integer x = new Integer(127);
 Integer t;
 Integer x;
 if(sc.nextLine().equals("x")){
 x = 127;
 t = 127;
 }
 else{
 t = 128;
```

```
 x = 128;
}
while (x <= t && x >= t && t != x);
}
```

A JDK Integer.java forrásban leírást kapunk az Integer objektumok létrehozásával kapcsolatban, hogy ha [-128,127] közötti számot adunk meg akkor egyseges meglevo memoriacímhez rendeli az objektumot, így az x és t objektum memoriacímre megegyezik tehát nem teljesül a while ciklus `t!=x` része, így kilep a ciklusból. Ha más [-128,127] tartományon kívüli számot adunk meg akkor létrehozza az objektumot a new kulcsszó segítségével, ezek alapján kikerülhető ez az eljárás ha new kulcsszóval hozunk létre Integer-t így ugyanúgy működik minden számmal. Ezt memoriatakarékosság céljából vezethették be. Tehát ha különböző memoriacímre hozzuk létre az objektumainkat akkor a while ciklus minden feltétele teljesül és végtelen ciklust kapunk. A ciklus többi feltétele teljesül mivel a két objektum értékei megegyeznek így nagyobb egyenlő és kisebb egyenlő egyik a másiknál, az egyenlő nem egyenlő pedig nem az értékre hanem a memoriacímre vonatkozik.

## 11.4. Yoda

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

```
public class Main {
 public static void main(String[] args) {
 String a = null;
 //Yoda condition
 if("alma".equals(a)) {
 System.out.println("Egyenlo.");
 }
 //NullPointerException.
 if(a.equals("alma")) {
 System.out.println("egyenlo");
 }
 }
}
```

A `java.lang.NullPointerException` kivétel akkor keletkezik ha null értékhez hasonlítunk azaz nem létező értékhez hasonlítunk mivel ennek nem lenne értelme. Ezt a hibát kijavíthatjuk a Yoda conditions alkalmazásával mivel ugyanazt a feltételt ellenőrizzük maskepp leírva azaz felcseréljük a feltétel jobb és bal oldalán lévő kifejezéseket, nyilván a null-t feltételként csak egyenlőség operátorral használjuk mivel a kisebb, nagyobb feltételnek nem lenne értelme, ekkor egyébként a kacsacsor is fordulna. Tehát a program kiírja az Egyenlo. szöveget a kijelzőre, de az egyenlo sztringet nem.

## 11.5. Kódolás from scratch

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

```
public class PiBBP {
 String d16PiHexaJegyek;
 public PiBBP(int d) {
 double d16Pi = 0.0d;
 double d16S1t = d16Sj(d, 1);
 double d16S4t = d16Sj(d, 4);
 double d16S5t = d16Sj(d, 5);
 double d16S6t = d16Sj(d, 6);
 d16Pi = 4.0d * d16S1t - 2.0d * d16S4t - d16S5t - d16S6t;
 d16Pi = d16Pi - StrictMath.floor(d16Pi);
 StringBuffer sb = new StringBuffer();
 Character hexaJegyek[] = { 'A', 'B', 'C', 'D', 'E', 'F' };
 while (d16Pi != 0.0d) {
 int jegy = (int) StrictMath.floor(16.0d * d16Pi);
 if (jegy < 10)
 sb.append(jegy);
 else
 sb.append(hexaJegyek[jegy - 10]);
 d16Pi = (16.0d * d16Pi) - StrictMath.floor(16.0d * d16Pi);
 }
 d16PiHexaJegyek = sb.toString();
 }
 public double d16Sj(int d, int j) {
 double d16Sj = 0.0d;
 for (int k = 0; k <= d; ++k)
 d16Sj += (double) n16modk(d - k, 8 * k + j) / (double) (8 * k + j);
 return d16Sj - StrictMath.floor(d16Sj);
 }
 public long n16modk(int n, int k) {
 int t = 1;
 while (t <= n)
 t *= 2;
 long r = 1;
 while (true) {
 if (n >= t) {
 r = (16 * r) % k;
 n = n - t;
 }
 t = t / 2;
 if (t < 1)
 break;
 }
 }
}
```

```
 r = (r * r) % k;
 }
 return r;
}
public String toString() {
 return d16PiHexaJegyek;
}
public static void main(String args[]) {
 System.out.print(new PiBBP(1000000));
}
}
```

A BBP algoritmus segítségével kiszámítjuk  $n > 0$  számból a pi hexadecimalis  $n+1$ -edik számjegyet az előtte lévő számjegyek meghatározása nélkül. Ezt az algoritmust 1995-ben találták fel, ez az első formula melyet számítógép segítségével alkottak meg. Az algoritmust a  $\{16^d \text{ Pi}\} = \{4 * \{16^d S1\} - 2 * \{16^d S4\} - \{16^d S5\} - \{16^d S6\}\}$  képletre építettük, ahol  $\{16^d S_j\} = \sum_{k=0}^{d-1} (16^{d \bmod 8k+j/8k+j})$ , ahol  $j$  értékei 1, 4, 5, 6. Ez utóbbi képlet a  $\log_2$  értékek bináris kiszámítását lehetővé tevő formulából adódik, mert észrevették hogy tetszőleges számjegytől kezdhetik számolni a  $\log_2$  értékeit. Az eredeti képletet bővítették  $\bmod k$ -val mivel a törtesz lesz fontos az algoritmus szempontjából. A formula szempontjából az első sum rész lesz a lényeges mivel a második a végtelenig megy, de mivel folyton csökken az értéke így előbb utóbb eléri egy határt aminél elhanyagolható lenne, de a lebegőpontos-aritmetika is véges a gépponton előbb-utóbb. A számlálóbn történő kifejezést kiszámíthatjuk szorzatokra bontással, erre van is egy algoritmus melyet kisebb módosításokkal felhasználtunk. Nem pontos végeredményt kapunk szoftveres és hardveres okok miatt. A `toString` függvény segítségével sztringgé alakítjuk az objektumot, a `floor` függvény pedig a nála kisebb egyenlő legnagyobb egész számot adja eredményül, ez szükséges mivel az eredményt 0 és 1 közötti számként kapjuk meg.

## 12. fejezet

# Helló, Gutenberg!

### 12.1. C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven es Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II.

A C++ es a Java egyaránt objektum-orientált programozási nyelv, mégis mind a C++, mind a Java programozási nyelv sok szempontból különbözik egymástól. A C++ programozási nyelv alapozza mind az eljárási, mind az objektum-orientált programozást, így módon hibridnek nevezik. A C++ es a Java programozási nyelvek eltérő tervezési célokkal készültek. A C++ programozási nyelv alkalmazások es rendszerek fejlesztésére került bevezetésre es a C programozási nyelv kiterjesztése. A C++ programozási nyelv az eljárási nyelv tulajdonságai mellett támogatást nyújtott az objektum-orientált programozási szolgáltatásokhoz, a kivételkezeléshez, az általános programozáshoz. A Java programozási nyelv értelmező funkcióval rendelkezik olyan nyomtatási rendszerek számára, amelyek később támogatták a hálózati számítás. A Java programozási nyelv egy virtuális mechanizmusra épül, amely nagyon biztonságos és hordozható a programozási környezetben. A Java programozási nyelv csoportosítva van egy széleskörű könyvtárral, amely a meglevő platform absztrakcióinak támogatása érdekében van megvalósítva. A Java programozási nyelv fejlesztésének fő célja egy könnyen használható es széles körben elérhető programozási nyelv kifejlesztése volt. A C++ es a Java programozási nyelvek is statisztikailag tipizált objektum-orientált programozási nyelvek, hasonló inkompatibilis szintaxissal rendelkeznek. A Java programozási nyelv kiterjedt dokumentációval is rendelkezik, amely Javadoc néven ismert. A C++ programozási nyelv mutatókat használ, míg a Java programozási nyelv nem tartalmaz mutatókat. A Java a „korlátozott mutatók” fogalmát használja. A korlátozza minősítő alkalmazható egy adatmutatóra annak jelzésére, hogy a mutató deklarációjának hatálya alatt az összes rajta elérhető adat csak az adott mutatón keresztül, de más mutatóon keresztül nem érhető el, így lehetővé teszi a fordítónak bizonyos optimalizálások elvégzését azon feltevése alapján. A C++ programozási nyelvben a program fut es a fordítás a fordító használatával történik. A C++ programozási nyelv fordítója konvertálja a forráskódot gépi szintű nyelvre, ami a C++ programozási nyelvet platformfüggetlen nyelvvé teszi. A Java programozási nyelvben a java forráskódot először bajtkóddra konvertálják a fordításkor. Ezt a bajtkódot ezután az értelmező futási idő alatt értelmezi, hogy előallítsa a kimenetet, ez teszi a Java programozási nyelvet platformfüggetlen nyelvvé. A C++ programozási nyelv nem nyújt beépített támogatást a szálakhoz, míg a Java programozási nyelv támogatja a szálakat implicit módon. Noha a C++11 programozási nyelv legújabb megvalósításában támogatja a szálakat is. A Java programozási nyelv nagyjából hasonlít a C++ programozási nyelvre, de nem foglal magában olyan összetett fogalmakat, mint

a strukturak, az operatorok tulterhelese, mutatok, sablonok, egyesitesek stb. A Java programozasi nyelv szinten nem tamogatja a felteteles forditast (# ifdef / # ifndef tipus). A Java programozasi nyelvben minden entitas a java tárgya, az alaptípusok kivételével. A Java programozasi nyelv egyetlen gyokerhierarchia-val rendelkezik, mivel minden a java.lang.Object fajlból származik. A C++ és Java programozasi nyelvek tamogatjak az OOPS koncepciokat. A C++ programozasi nyelv rugalmassagot nyujt futasi idoben es szel-es tipusu hierarchiakat tud vegrehajtani. A C++ programozasi nyelv a C programozasi nyelvre epul es a funkcioival visszafele kompatibilis. A C++ programozasi nyelv egy egyfajta alacsony szintu programozasi nyelv nehany magas szintu funkcioval kiegészítve. A C++ programozasi nyelv memoriakezelese egy kezi folyamat, amelyet a programozónak kell kezelnie, amely eloidozheti a memoriaaszivargasok es a szegment-acios hibak kockazatat. A Java programozasi nyelv beépített hulladékgyujto mechanizmussal rendelkezik amely nyomon követi az objektumok számára kiosztott memoriát es automatikusan felszabadítja azokat amikor már nem használják. A Java programozasi nyelv kulonbozo primitiveket es objektumtipusokat ki-nal es erosen tipizalt programozási nyelv, ez lehetove teszi a primitivumok megfelelo objektumtipusokka torteno atalakitast, peldaul egesz szamu objektumma egesz osztaly objektum felhasznalasaval stb. A Java programozasi nyelv automatikus polimorfizmust biztosit es korlatozhatja azt az explicit modszer felulbira-latanak megtiltasaval. Mind a C++, mind a Java programozasi nyelv rendelkezik hozzaferesi specifikato-rokkal amelyek korlatozzak az attributumok és metodusok hatokoret, az osztalyon belül privat, a csomagon belül vedett, az osztalyon valamint a csomagon kívül nyilvános. A C++ programozasi nyelv tamogatja a mutatoakat, strukturakat, egyesiteseket, sablonokat, operatorok tulterheleseit vagy a mutatok szamtani hasz-nalatat. Java programozasi nyelvben ezek nem talalhatoak meg, hanem referenciak vannak amiket nyers cimme nem lehet visszafejteni es a veszelyes változtatások nem engedelyezettek. A referenciáknak típusa-ik vannak es biztonsagosak. A C++ programozasi nyelvben destruktor meghivodik amikor egy objektum torlodik, Java programozasi nyelvben a garbage collection vegzi el automatikusan a memoriafelszabaditast. A C++ programozasi nyelv támogatja a feltételes összeállítást és beillesztést, a Java programozasi nyelv nem. A Java programozasi nyelv rendelkezik a függvény tulterhelesevel, de nincs operator tulterheles. A sztring osztaly a + es + = operatorokkal használja a sztringek összekapcsolast es a karakterlanc kifejeze-sek automatikus típuskonverziót használnak, de ez egy specialis beépített eset. A C++ programozasi nyelv tamogatja a technika tulterheleseit es az adminisztratorok tulterheleseit. A Java programozasi nyelv beépített tamogatast nyujt a dokumentacios megjegyzésekhez (/ \*\*... \*/); ezért a Java forrasfajlok tartalmazhatjak a saját dokumentaciojukat, amelyet egy külön eszköz olvas, általában Java doc es ujraformazzak HTML-re, ez elosegiti a dokumentacio egyszeru karbantartast. A C++ programozasi nyelv protestalo kodot hoz létre es lehet, hogy egy hasonló kod nem fut kulonbozo szakaszokban, a Java programozasi nyelv platform-fuggetlen. Mindket programozasi nyelvnel a program a main függvennyel kezdodik, de a main függvény deklaralasa eltero. Mindket programozási nyelvnek vannak aritmetikai es logikai operatorai, aritmetikai-ak a + - \* / ,relaciosak a kisebb nagyobb kisebbbegenlo nagyobbbegenlo egyenlo nemegyenlo. Mindket programozasi nyelvben vannak adattípusok, peldaul float, double, a Java programozasi nyelvben Boolean van, C++ programozasi nyelvben pedig bool. Mindket programozasi nyelvben vannak azonos kulcssza-vak, peldaul break, continue, return, static. Mindket programozasi nyelvnek hasonló a szintaktikaja es mindket programozasi nyelvben használhatunk többsoros kommenteket // es /\* jelek segitsegevel. Mind- ket programozasi nyelv lehetove teszi több szal egyideju vegrehajtasat a multitasking elerése erdekeben. Mindket programozasi nyelv hasonló ciklusokat, peldaul for while ciklusok es felteles allitasokat, peldaul switch if-else tartalmaz. A szamitogepes nyelveknek megkulonboztetjuk a szintjeit, leteznek gepi nyelvek es magas szintu nyelvek. Minden processzornak van egy gepi nyelve, melyen vegrehajtja az utasitasokat, azonban az általunk irt magasszintu nyelveken irt kodon irt utasitasokat a processzor nem tudja vegrehajtani mivel nem gepi nyelven van írva, így at kell alakítanunk gepi nyelvre. A magasszintu nyelvek peldaul a C, melyek saját szintaktikai esz szemantikai tulajdonsagokkal rendelkeznek. A magasszintu nyelvek gepi nyelvekke alakitasara ket modszer van az egyik a forditoprogramos a másik az interpretes. A forditoprogra-

mos atalikitást használjuk mivel ellenorzi a kódot szintaktikailag és szematikailag mielőtt lefordítana, ilyen például a C nyelvhez a gcc, melynek több változata ismert c90, c11, melyeknek neveiben az eljáratra utalnak a számjegyek. A fordítóprogram készít tárgyprogramot, ha szintaktikailag helyes a kódunk, melyet a gcc fordítónál az -o kapcsolóval nevezhetünk el. Az interpretés fordítással a kód rögtön lefut az ellenőrzések után, mivel nem készül tárgyprogram amit később is futtathatunk. A programozási nyelveket két főbb osztályba sorolhatjuk az egyik az imperatív nyelvek, melyekbe algoritmusok segítségével írjuk meg az utasításokat különböző változókat felhasználva és van lehetőségünk memóriafoglalásra is, ezeket gyakorabban használjuk ilyen például a C nyelv. Vannak a deklaratív nyelvek amelyek nagyjából az imperatív nyelvekkel ellenkező tulajdonságokkal rendelkeznek, presze ezeken kívül leteznek még másélv nyelvek amelyek nincs egyseges jellemzőjük. Minden programnyelvnek saját karakterkészlete van mely betűkből, számokból és egyéb karakterekből épül fel. A számok általában egyseges decimális számok. A betűknel elteró lehet hogy külön kezeli e a kis és nagybetűket mint például a C nyelv. A specialis karakterek közül a szokott, tabulatort, entert nem különbözteti meg a fordító, ezeket egysegesen white space-knek nevezzük, tehát mindegy melyiket használjuk szó alkotásra szintaktikai szempontból, azonban a jobb átlathatóság miatt használjuk mindegyiket. A forrásprogramban előforduló lexikális egysegeket a lexikális elemzés során a fordító felismeri és tokenizálja azokat, ilyenek a többkarakteres szimbólumok például C nyelvben a ++, a szimbolikus nevek amiket a programozó hoz létre az egyes elemek azonosítására, ezeknek a neveinek betű karakterekkel kell kezdődniük és nem tartalmazhatnak specialis karaktereket például műveleti jeleket és a kulcsszók például az if melynek a nyelv tulajdonít jelentést, megjegyzések melyek segítik a program értelmezést az olvasó számára, a címkek melyekkel a címezett utasításra a program egy másik részéből hivatkozni tudunk, literalok melyek megadják a változó típusát például a double. Vannak kötött és szabad formátu nyelvek, a kötött formátumu nyelvekben egy sor egy utasítás ha nem fert ki egy sorba külön jelezni kellett a sorvége törlesevel és a programelemeknek is meghatározott helye van a sorban. A szabad formátumu elemeknel egy sorban akárhany utasítás kerülhet és tetszőleges pozíciókban használhatjuk a programelemeket, az utasítások véget pedig pontosveeszóval jelezzük. Az eljárásorientált nyelvekben a lexikális egysegeket a megfelelő módon el kell választanunk egymástól whitespace-szel vagy elhatároló jelekkel például zárójellel. A kifejezések segítségével egy adott értékből, operandusból új értéket határozzunk meg operátorok segítségével, ehhez használhatunk zárójelézést is hogy meghatározzuk a műveleti sorrendet. A ketoperandusú kifejezések alakjait az operátor elhelyezkedése határozza meg, ha elől van akkor prefix alakú, ha közepén infix, ha az operandusok mögött akkor postfix. A műveletek megfelelő sorrendbeli végrehajtása után a kifejezésnek megkapjuk az új értéket és hozzárendelődik a típusa, ezt a kifejezés kiterkelesenek nevezzük. A műveletek végrehajtási sorrendje lehet balról-jobbra, jobbról-balra, balról-jobbra a precedenciatablázat segítségével. Mivel az infix alak operátorai nem azonos érossegűek így az infix alak nem egyértelmű, ehhez használjuk a precedenciatablázatot, melynek műveleti sorrendjét felulírhatjuk zárójelek használatával, a teljesen bezárójelezett alaknak egy műveletvégrehajtási sorrendje van. A kifejezés típusának meghatározására kétféle módszer van, az egyik a típuségyenlőség ekkor a ketoperandusú kifejezés mindeket értéke azonos típusú vagy az operátor határozza meg a típust. Ket programozási eszköz típusa akkor azonos ha egy utasításban deklaráltuk és azonos típussal és a két eszköz összetett típusú, szerkezetük megegyezik. A típuskenyszerítés módszerével a ketoperandusú kifejezéseknek különböző típusú operandusai lehetnek, a műveletek viszont csak azonos típusú operandusokkal végezhetőek el így annak megfelelően átkonvertálódnak. A típuskenyszerítésnek van bővítéses fajtája amikor a konvertáló típus elemei elemei a céltípusnak is, szűkítés esetén viszont nem elemei és ekkor értékcsönkítés vagy kerekítés történik. Az utasítások alkotják az algoritmusok egyes részeit, illetve a fordítóprogram ennek segítségével generalja le a tárgyprogramot, ezek alapján két csoportjuk van deklarációs és végrehajto utasítások. A deklarációs utasítások információt szolgáltat a tárgykód generalásához, a végrehajto utasításokat pedig lefordítja tárgykódrá. A végrehajto utasításokat kilenc csoportba sorolhatjuk. Az értékado utasítás segítségével értéket adhatunk változóknak a program futása alatt. Az üres utasítás segítségével a program egy üres gépi utasítást hajt vég-



re, ez szintaktikailag hasznos. Az ugro utasítás segítségével egy adott címkevel ellátott utasításnak adhatjuk át a vezérleést a GOTO parancs segítségével. Az elágaztató utasítás segítségével a program több tevékenység közül használja a megfelelőt. A ciklusvezérlő utasítás segítségével a program egy adott tevékenységet többször is végrehajthat, akár végtelenszer is. C nyelvben a CONTINUE utasítás az aktuális ciklusszalból kilep nem hatja végre a további utasítást, hanem újabb szálba kezd. A végrehajtando műveletek sorrendjét a vezérlésutató utasítások adják meg. A kifejezések végére pontosvesszőt teszünk hogy utasításként működjenek, egyébként fordításkor szintaktikai hiba kapnánk. Összetett utasításokat létrehozhatunk, ha kapcsos zárójelek közé adunk meg több utasítást, ezek végére nem kell pontosvessző. Az if-else utasítás a paraméterként kapott kifejezés értéke alapján dönti el hogy végrehajtja-e a hozzárendelt utasítást, ha egy az értéke akkor végrehajtja, nulla érték esetén viszont nem hatja végre, hanem ha adunk meg else ágat annak az utasítást hajtja végre. Az if illetve az else végrehajtja az utána következő utasítást vagy utasításokat amit úgy kapcsoszárójelben kell megadnunk. Az if utasításokat egymásba is ágyazhatjuk egymásból következő feltételek esetén, kapcsoszárójel használatával nélkül az else mindig a hozzá legközelebb eső if-hez tartozik. A kapcsos zárójel használatával a beagyazott if utasítások esetén az esle ág melyik if utasításhoz tartozását is meghatározhatjuk. Egy if utasításhoz egy else ág tartozhat ami hatékonyabban működik mint két if utasítás használatával. Az else-if utasításokban megadunk egy kezdő if utasítást majd akárhany else-if utasítást megadhatunk melyek egymás után kiértékelődnek és ha az egyik értéke egy akkor végrehajtja a hozzárendelt utasítást és kilep az else-if utasításból. Ha az if és minden else-if ág értéke nulla akkor ha van else ág akkor az ahhoz rendelt utasítás hajtódik végre, ezt használjuk hibakezezésként is. A switch utasítást paraméterként megadunk egy kifejezést majd egy összetett utasításban megadjunk akárhany case `` : ágat majd egy default: ágat. A case `` részében megadunk egy állandó értékű kifejezést melynek értéke ha megegyezik a switch paraméterként adott értékekkel akkor végrehajtja a case : után megadott utasítást. Ha egyetlen case esetében megadott kifejezés értéke sem egyezik meg a switch paraméterként adott értékekkel akkor a default ághoz rendelt utasítás hajtódik végre. A while utasítás a paraméterként megadott kifejezés értéket ellenőrzi hogy egy-e, ha igen akkor végrehajtja a hozzárendelt utasítást, majd ismét ellenőrzi és egy érték esetén újra végrehajtja az utasítást, majd ezt addig hajtja végre rekurzívan amíg a kifejezés értéke nem lesz nulla. Ha a paraméterként adott kifejezés értéke egy akkor végtelenszer hajtja végre a while a hozzárendelt utasítást. Ha a paraméterként adott kifejezés tartalmaz változót, annak értéket módosítja a while utasításhoz rendelt utasítások. A for utasításnak három kifejezést adhatunk meg paraméterül, melyek közül az első kezdőértéket ad a második kifejezésben vizsgált operandusnak, a második kifejezés operator segítségével eldönti nem nulla vagy nulla az aktuális értéke a kifejezésnek, a harmadik kifejezéssel pedig módosítja az első paraméterként megadott változó értékét. A kifejezések elhagyásával és a három pontosvessző használatával végtelenszer fogja végrehajtani a for utasítás a hozzárendelt utasítást. Minden for utasítást átírhatunk while utasítással úgy hogy a for ciklus paraméterként megadott első kifejezést a while ciklus előtt definiáljuk, a második kifejezés a while utasítás paramétere lesz, a harmadik kifejezést pedig a while utasításhoz rendelt utasításban definiáljuk. A do-while utasítás a do utasításhoz rendelt utasítást hajtja végre majd ellenőrzi a while utasítás paraméterül adott kifejezés értéket hogy nulla-e, ha nem újra végrehajtja a do utasításhoz rendelt utasítást majd ismét ellenőrzi a while utasítás paraméterül adott kifejezés értéket hogy nulla-e, ezt hajtja végre rekurzívan amíg a while utasítás paraméterül adott kifejezés értéke nulla nem lesz. A for és a while utasítással ellentétben nem az utasítás végrehajtása előtt ellenőrzi hogy végrehajtja-e vagy nem, hanem először végrehajtja az utasítást majd ellenőrzi hogy végrehajtja-e újból. A while ciklushoz hasonlóan itt is az utasítás részben változhat a változó értéke, amit operandusként használ a while utasítás paraméterként megadott kifejezés. A do utasításhoz rendelt utasítást célszerű kapcsoszárójelben használni hogy biztosan a while utasításhoz tartozzon. A break utasítás segítségével kilep a for, while, do-while, switch utasításokból a lefordulásakor, tehát nem addig hajtódik végre amíg a kifejezés értéke nulla nem lesz hanem addig amíg a break utasítás le nem fordul. A continue utasítás a for, while, do-while utasításokhoz tartozó utasításból kilep és ismét a paraméterként adott kifejezés vagy kifejezések értéke alapján dönti el hogy újra lefut-e az

ugye for vagy while utasitashoz rendelt utasitas. Tehat a for es a while utasitasokhoz rendelt utasitasoknak az utasitasai a countinou utasitas lefutasaig hajtodnak vegre, nem a teljes utasitas utolso utasitasai. A goto utasitast a tobbszorosen egymasba agyazott utasitasokbol valo kilepesre használjuk úgy hogy az adott utasításban megadott goto címke nevű utasitas vegrehajtodasakor a címke nevű címkehez tartozó utasításra ugrik a program. Címket a címkenéve : utasitas paranccsal hozhatunk létre. A goto utasitas használatával csak az adott függvényben címezett utasitas érhető el. A goto használata kikerülhető megfelelő segéd utasitasok használatával. A könyvtárakban van definiálva a hiba, melyeket megfelelően kezelhetünk, az ilyen definiált hibákat kivételeknek azaz exeption-nak nevezzük. A kivételkezelés úgy történik hogyha egy függvény nem tud kezelni egy hibát kivetelt dob rá a throw függvénnyel melyet a catch függvénnyel elkaphatunk ha megfelelő kivetelt definiáltunk. Hibakezeléssel nélkül a program hiba esetén leáll vagy hibásan működik. A C++ programoknak az alapvető hibakezelése hogy hiba esetén a program leáll, kiveve ha a el nem kapjuk az összes hibát és megfelelően kezeljük hogy ne álljon le. A C programoknál viszont tovább fut hibásan is, és így csak később derülhet ki a hiba esetleges leállás esetén. A kivételkezelés a szinkron kivételek kezelésére szolgál például a ki és bemeneti hibák, az aszinkron eseményeket például a billentyűzet felől érkező hibát szignálokkal kezeljük. A kivétel előfordulást egy osztállyal írjuk le melynek egy objektuma a kivétel. Van úgynevezett vegrehajtási verem mely a throw függvény meghívásával tekerődik vissza a catch függvényig. A kivételek osztályait hierarikusán kezeljük hogy egyszerűbben tudjuk módosítani azokat. A kivételt csak akkor tudjuk elkapni ha a catch parameterének típusa megegyezik a throw-val meghívottnak, vagy ha a catch parametere főosztálya a throw-val meghívottnak, vagy ha a catch parametere és a throw-ra meghívott is mutatótípus melyeknek vagy megegyezik a típusuk vagy a catch parametere bazisosztálya a throw-val meghívottnak, vagy ha a catch parametere referencia és teljesül rá hogy megegyezik a típusa a throw-val meghívottnak, vagy ha a catch parametere referencia és teljesül rá hogy megegyezik a típusa a throw-val meghívottnak, vagy ha a catch parametere referencia és teljesül rá hogy megegyezik a típusa a throw-val meghívottnak, vagy ha a catch parametere referencia és teljesül rá hogy megegyezik a típusa a throw-val meghívottnak. A const szó használatával a kivételek értéket nem változtathatjuk és minden kivétel másolható. Ha nem tudjuk kezelni teljes egészében az elkapott kivételt akkor továbbdobhatjuk a throw utasitas lefutatasával, ekkor nem az elkapott kivetelt dobjuk tovább hanem az eredeti throw által dobottat. A try függvényben definiáljuk a catch függvényeket, melyeknek a sorrendje osztály hierarhia szerint növekvő sorrendben kell legyen mivel ekkor az alosztályok nem kapodnanak el. Fájlkezeléssel a megnyitott fájlt nem mindig zárul automatikusan, mivel ha a hiba az fclose meghívódása előtt van akkor a hibát kezelhetjük a use\_file() függvény visszatérési értéket ad a hibát felismerő függvénynek, de az fclose függvény már nem hívódik meg így a lefoglalt memóriaelfolyik. Ezt a hibát javíthatnánk úgyis hogy a catch függvénnyel elkapunk minden kivételt és meghívjuk benne az fclose függvényt, de ez nagyobb kódoknál bonyolultabb így osztályban definiáljuk a fájlkezelést melynek a destruktora elvégzi a megfelelő memóriafelszabadítást a kivételkezeléstől függetlenül. A használatos objektumokat lokális objektumként hozzuk létre a konstruktor inicializációs listájával, mivel a konstruktor segítségével létrehozott objektumot torli a destruktork és az így létrehozott objektum teljesen jön létre megfelelően megírt konstruktor esetén. Az inicializációs lista használatával a fordító ellenőrzi hogy a létrehozott objektumok létrehozása közben kivétel adódik akkor csak a létrehozott objektum torlik, így nem szükséges a konstruktor íráshoz kivételkezelő kód. Ha csak egy objektum rész jön létre mivel kivétel van a konstruktor parameterben akkor nem jön létre teljesen az objektum így a destruktork nem torli, ehelyett inicializációs lista segítségével vektort használunk az objektum memóriakezeléséhez. Az std könyvtárban van auto\_ptr kisebbkacsacsor nagyobbkacsacsor sablon mely a kisebbkacsacsor nagyobbkacsacsor között megadott típussal hoz létre mutatót az adott objektumra, hasonlóan működik a hagyományos mutatókhoz, de torlésekor megsemmisül az általa mutatót objektum is. Az auto\_ptr egy másik mutatóba másolásval a mutató nem mutat semmire és az auto\_ptr-eket a másolás megváltoztatja, de használhatjuk rá a const kulcsszót melynek segítségével nem lesz másolható. Ezt az auto\_ptr\_ref akadályozza meg és megvalósítja a másolását. Nem std típusokkal való másolása például vektorba másolaskor sérülés veszély áll fenn. Az auto\_ptr által mutatót objektum torlik a kivételkezeléstől függetlenül. Elhelyező utasitas használatával kivételkezeléskor is felszabadul a new által lefoglalt hely a megfelelő delete parancsokkal automatikusan, más

modon lefoglalt memóriát nem szabadit fel automatikusan. Ha elfogy a memória a memóriaszivargások miatt akkor kivételkezeléssel és egy végtelen ciklussal használt malloc függvény segítségével kereshetünk memóriacímeket, melyeket a túlterhelt new függvény ad vissza, ha nem talál hibakezelést használunk. A bad\_alloc kivételt melynek működéséhez marad memória, és a new\_hanler-t hívjuk meg előtte ha nem talál memóriát a malloc. A konstruktoroknak alapvetően nincs visszatérési értékük így hibás állapotú objektumot hozhatunk létre vagy globális változóval ellenőrizzük az objektum létrehozás sikerességét vagy ne hozunk létre kezdeti értéket vagy kezdeti érték helyett egy tagfüggvény adjon értéket mely jelenti a hibát. Ezeknek a felhasználói ellenőrzésével kivételkezelhetjük a konstruktorokat. A destruktorki meghívódhat a verem visszatérése közben ha elhagyunk egy megfelelő objektumot. Ha egy kivételt nem kapunk el akkor a terminate függvény kerül meghívásra vagy ha a kivételkezelő eljárás a vermet hibásnak találja vagy egy delete kivétel kiváltással próbál véget érni. A terminate függvény meghívódása meghívja az abort függvényt amely kilep a programból nem feltétlenül szabályosan és az exit függvénnyel ellentétben ennek nincs visszatérési értéke arról hogy szabályosan állt-e le a program vagy nem. Egy elakadt kivétel nem mindig jelenti a program hibáját.

## 12.2. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba

A Python nyelvet Guido van Rossum alkotta meg 1990-ben, ami objektumorientált és platformfüggetlen így használhatjuk több platformon is például Windows, Unix rendszereken, iPhone, mivel mobil eszközökre is kifejlesztették. Főként prototípusok tesztelésére használjuk, mivel az alkalmazások futtatása és megírása is kevesebb időt vesz igénybe mint a C++ illetve Java nyelv használatánál. Mivel egy köztes nyelv így nincs szükség fordításra és linkelésre sem, és könnyű és könnyen olvasható programokat készíthetünk mivel összetett kifejezéseket használhatunk, a tagolásnál nincs szükség zárójelek használatára, illetve nincs szükség változó és argumentumdefinícióra sem. A Python szkriptnyelvek csoportjába tartozik mivel a kódot a az értelmező, interpreter futás közben értelmezi, nem fordító hoz létre új fájlt amit futtathatna gépi nyelven. Egy utasítás a sor végéig tart ha nem ér be egy sorba akkor a sor végerére írt \ operátorral jelezzük hogy az utasítás a következő sorban folytatódik. A nyelv szintaxisára behúzásalapú tagolás jellemző tehát egy adott utasítások csoportjának véget egy kisebb behúzású sor jelzi, tehát zárójeleket vagy kulcsszavakat nem használunk tagolásra csak whitespaceket. A behúzásokat egysegesen kezeljük vagy szóközt vagy tabulátorokat használunk illetve a szkript, kód első utasítása nem lehet behúzott. Az értelmező a sorokat tokenekre bontja, ami lehet azonosító, kulcsszó, operátor, delimiter, literal. Az azonosító lehet egy változó, osztály, függvénymodul, objektum neve, ami betűvel vagy aláhúzással kezdődhet és számokat is tartalmazhat. A nagy és kis betűket megkülönböztetjük egymástól. Kulcsszavak az and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, yield. Megjegyzések elhelyezésére a # operátort használjuk ami a sor végéig érvényes. A változók típusait nem szükséges megadni mivel a hozzárendelt érték alapján a fordító automatikusan kitalálja azt. Az adattípusok a számok, sztringek, ennesek, listák, szotarok. A számok lehetnek egészek, lebegőpontosak, komplex számok, oktális formátum esetén bevezető 0-t használunk, hexadecimális esetén 0x-et. Sztringeket idézőjelek és aposztrófok között is megadhatunk, és használatával pedig Unicode szöveggel. Az ennesek akár eltérő típusú objektumok vesszővel elválasztva zárójelben. A lista akár különböző típusú elemek vesszővel elválasztott rendezett sorozata szögletes zárójelben, az elemeket az indexükkel azonosítjuk. A szotar akár különböző típusú elemek vesszővel elválasztott rendezetlen sorozata, az elemeket kulcsukkal azonosítjuk. A null értéket None a használatos neve. Minden adat objektum, a változók objektumokra mutató referenciák, ha egy objektumról toroljuk az összes hivatkozást a foglalt memóriaterület automatikusan felszabadul. A

valtozo ertekeket a = jellel adhatjuk meg, a del kulcsszoval pedig torolhetjuk a valtozot. A fuggvényben felvett valtozok lokalisak, a global kulcsszoval tehetjuk oket globalissa, így a fuggvényen kívül használhatjuk oket. A kifejezéseket rovidíthatjuk az alábbi módon  $a=b$   $b=c$  -t  $a=c$  -re. A különbozo beépített típusok közötti konverzió támogatott: int, long, float, complex. A sztringeket, listákat, enneseket együttesen szekvenciának nevezzük melyeken különbozo műveleteket hajthatunk végre és beépített fuggvényeket alkalmazhatunk rajtuk. A len fuggvény visszája a szekvencia hosszát, a min és max fuggvény a szélsőertekeket. A + jellel szekvenciákat fűzhetünk össze, az a in s és a not in s kifejezések tartalmazására adnak választ. A szekvenciák elemeit inexelessel lehet elérni 0-tól indulva, ha negatív indexet használunk akkor a szekvencia végétől számítjuk az indexet, : jel használatával pedig index intervallumot is megadhatunk. A szekvencia elemeit a del kulcsszoval torolhetjük. A listakon vegezhető műveletek: count(e) visszája e előfordulásainak a számat, index(e) visszaadja e első előfordulásának az indexet, append(e) hozzáfűzi e-t a lista végére, extend(l) az l lista elemeit fűzi a lista végére, insert(i,e) beszűri az e elemet az i-edik helyre, remove(e) eltávolítja e első előfordulását a listából, pop[(i)] eltávolítja az i-edik elemet a listából és visszája annak értéket, ha az index nincs megadva az utolsó elemmel vegzi el a műveletet, reverse helyben megforítja az elemek sorrendjét a listában, sort[(f)] sorba rendezi a lista elemeit az f fuggvény segítségével, ha nincs megadva f fuggvény akkor a cmp fuggvényt használja. Szótaraknál is hasonlóan alkalmazzuk a len, del, in operátorokat, a szótárak elemeit a szögletes zárójelben írt kulcs segítségével adhatjuk meg. A szótáron vegezhető műveletek: copy() visszáter a szótár egy másolatával, has\_key(k) igaz értéket kapunk ha a k kulcs szerepel a szótárban, egyébként hamis értéket, items() iteritems() a kulcsérték párok listájával, illetve iteratorával ter vissza, keys() iterkeys() a szótár kulcsait tartalmazó listával, illetve iteratorral ter vissza, values() itervalues() a szótár értékeit tartalmazó listával, illetve iteratorral ter vissza, get(k[,x]) setdefault(k[,x]) visszáter a k kulccsal jelölt elemmel, ha nem létezik az x értékevel ter vissza, ha nem adunk meg x értéket akkor none lesz a visszáterési érték, a setdefault emelet meg be is állítja a k kulcsu elemet x-re, clear() kitorli az összes elemet a szótárból, popitem() kitorol és visszaad egy tetszőleges elemet a szótárból. A print segítségével kiirathatunk az stdout-ra sztringeket vagy valtozokat, ezeket vesszővel elválasztva kell megadni. Az if feltétel: utasítás elif feltétel: utasítás else: utasítást használhatjuk feltételek megadásra. A for in segítségével bármely felsorolható típus elemein végigmehetünk. A range(x,y,z) fuggvény egy listát general egész értékekből x-től y-ig z növekedéssel, az xrange kevesebb memóriaigennyel működik. A while feltétel: utasítás segítségével addig hajtódik végre az utasítás amíg a feltétel igaz. A break kulcsszoval kilep a ciklusból, a continue-val folytatja. Cimkeket a label kulcsszoval helyezhetünk el a kodban, a cimkere pedig a goto kulcsszoval ugorhatunk, a comefrom kulcsszoval pedig a label részhez ugrik. Fuggvényeket a def nev(): utasítás return módon hozhatunk létre. Definíálhatunk osztályokat az alábbi módon class osztálynev (osztályok): osztálytorzs. Az osztályok öröklődhetnek más osztályokból és példányaik az objektumok. Az osztályok már defíniált osztályok vesszővel elválasztott listái, attribútumokat hozzáadhatunk az egyes osztályokhoz és példányokhoz, ha módosítjuk az osztály attribútumát akkor ez a hozzátartozó objektumokra is hatással van. Objektumok és osztályok valtozaira objektum/osztálynev.valtozonev segítségével hivatkozhatunk. Az osztályok fuggvényeit módszernek nevezzük és az első paramtere a self kulcsszó kell hogy legyen, melynek értéke az objektumpéldány melyben a fuggvényt meghívták. Konstruktort is létrehozhatunk ami egy specialis fuggvény így a def \_\_init\_\_(self): módon definíálhatjuk. A specialis fuggvények, valtozok két alulvonás jel közé vannak írva. A Python a fejlesztés megkönnyítése érdekében beépített modulokat tartalmaz: appuifw a felhasználói felület kialakulását támogatja, messaging az SMS és MMS üzenetek kezelése meg, sysinfo a mobilkészületekkel kapcsolatos információk lekerdezésére használható, camera segítségével vegezhetünk a készülék kamerájával kapcsolatos műveleteket, audio segítségével készíthetünk hangfelvételeket és le is játszhatjuk azokat. A Python támogatja a kivételkezelést, a try blokkban keressük a hibát majd a vezérles hiba esetén az except részhez ugrik, ahol használhatunk egy else ágat is, az alábbi formában használjuk try: utasítások except [kifejezés]: utasítások [else: utasítások]. A finally: utasítás a try rész végén fut le, a raise beírt hibákra van.

## **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

## 13. fejezet

# Helló, Arroway!

### 13.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 13.2. 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



### 13.3. Általános

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

### 13.4. C

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

### 13.5. C++

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

### 13.6. Lisp

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

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

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