

Univerzális programozás

Programozás kézikönyv kezdőknek

Ed. BHAX, DEBRECEN,
2019. Május 09, v. 1.0.0

Copyright © 2019 Csontos Róbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Csontos, Róbert	2019. szeptember 29.	

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	2019-03-05	A Turing fejezet befejezése	csrobert
0.0.5	2019-03-12	A Gutenberg fejezet beillesztése	csrobert
0.0.6	2019-03-12	A Chomsky fejezet befejezése	csrobert
0.0.7	2019-03-19	A Caesar fejezet befejezése	csrobert
0.0.8	2019-03-26	A Mandelbrot fejezet befejezése	csrobert

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-04-02	A Welch fejezet befejezése	csrobert
0.1.0	2019-04-09	A Conway fejezet befejezése	csrobert
0.1.1	2019-04-23	A Schwarzenegger fejezet befejezése	csrobert
0.1.2	2019-04-30	A Chaitin fejezet befejezése	csrobert
1.0.0	2019-05-09	A könyv befejezése	csrobert

Ajánlás

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

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	7
2.3. Változók értékének felcserélése	9
2.4. Labdapattogás	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	12
2.6. Helló, Google!	13
2.7. 100 éves a Brun tétel	15
2.8. A Monty Hall probléma	15
3. Helló, Chomsky!	17
3.1. Decimálisból unárisba átváltó Turing gép	17
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	18
3.3. Hivatkozási nyelv	19
3.4. Saját lexikális elemző	20
3.5. l33t.1	20
3.6. A források olvasása	23
3.7. Logikus	24
3.8. Deklaráció	25

4. Helló, Caesar!	26
4.1. double ** háromszögmátrix	26
4.2. C EXOR titkosító	28
4.3. Java EXOR titkosító	29
4.4. C EXOR törő	30
4.5. Neurális OR, AND és EXOR kapu	33
4.6. Hiba-visszaterjesztéssel perceptron	35
5. Helló, Mandelbrot!	36
5.1. A Mandelbrot halmaz	36
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	37
5.3. Biomorfok	38
5.4. A Mandelbrot halmaz CUDA megvalósítása	40
5.5. Mandelbrot nagyító és utazó C++ nyelven	44
5.6. Mandelbrot nagyító és utazó Java nyelven	44
6. Helló, Welch!	45
6.1. Első osztályom	45
6.2. LZW	48
6.3. Fabejárás	52
6.4. Tag a gyökér	56
6.5. Mutató a gyökér	61
6.6. Mozgató szemantika	67
7. Helló, Conway!	75
7.1. Hangyaszimulációk	75
7.2. Java életjáték	75
7.3. Qt C++ életjáték	81
7.4. BrainB Benchmark	82
8. Helló, Schwarzenegger!	83
8.1. Szoftmax Py MNIST	83
8.2. Mély MNIST	83
8.3. Minecraft-MALMÖ	83

9. Helló, Chaitin!	86
9.1. Iteratív és rekurzív faktoriális Lisp-ben	86
9.2. Gimp Scheme Script-fu: króm effekt	87
9.3. Gimp Scheme Script-fu: név mandala	88
10. Helló, Gutenberg!	89
10.1. Juhász István: Magas Szintű Programozási Nyelvek 1	89
10.2. Kernighan-Ritchie: A C programozási nyelv	89
10.3. Levendovszky-Benedek: Szoftverfejlesztés C++ nyelven	90
III. Második felvonás	92
11. Helló, Arroway!	94
11.1. OO szemlélet	94
11.2. Homokozó	97
11.3. "Gagyí"	97
11.4. Yoda	100
11.5. Kódolás from scratch	102
12. Helló, Liskov!	105
12.1. Liskov helyettesítés sértése	105
12.2. Szülő-gyerek	107
12.3. Anti OO	109
12.4. Hello, Android!	110
12.5. Ciklomatikus komplexitás	114
13. Helló, Berners-Lee!	116
13.1. Élmény beszámoló a Python könyvből!	116
13.2. A java és a C++ nyelv összehasonlítása	117
IV. Irodalomjegyzék	118
13.3. Általános	119
13.4. C	119
13.5. C++	119
13.6. Lisp	119

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 (hogy lássuk mást is) példával.

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- C kapcsán a [**KERNIGHANRITCHIE**] könyv adott részei.
- C++ kapcsán a [**BMECPP**] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány **ISO/IEC 9899:2017** kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a **The GNU C Reference Manual**, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [**BMECPP**] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

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

- „, benne a bemutatósa.
- „, benne a bemutatósa.
- „, benne a bemutatósa.
- „, benne a bemutatósa.
- „, benne a bemutatósa.
- „, benne a bemutatósa.

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.

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Turing/Vegtelen_Ciklus

Egy ciklust végtelen ciklusnak nevezünk, ha addig fut, amíg valamilyen külső behatásra meg nem áll. Külső behatás lehet például az ha elmegy az áram és így leáll a számítógép. Az is végtelen ciklus, amikor egy futó program ablakát nyitva tartja az operációs rendszer. Ez akkor áll meg amikor mi bezárjuk az ablakot. Lehet viszont olyan eset is, amikor egy hibából adódóan jön létre. Ilyenkor nem szándékos a végtelen futás, ám külső behatás nélkül ebben az esetben sem fog megállni.

Olyan programot írni ami egy szálat 100%-ra pörget több módon is lehet. Az egyik ilyen mód az alább látott. Amint a program belép a ciklusba megkezdődik a végtelen futás.

Most pedig nézzük a kódot. A jelen látott példában egy **while** ciklust használunk. Ennek a feltételében egy **true** érték látható. Ez a feltétel kimondja hogy addig fusson a ciklus amíg a feltétel igaz. Mivel ez viszont mindig igaz marad, a ciklusmagból soha nem fogunk kilépni. Látható továbbá hogy a kód elején van **#include stdbool.h**. Erre azért van szükség, mivel a C nyelv alapbón nem tud true értéket használni, ezért tehát meg kell hívni hozzá ezt a függvénykönyvtárat.

```
Végtelen ciklus 1 szál 100%
{

#include <stdbool.h>

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


A következő példa azt mutatja hogy egy végtelen ciklus hogyan néz akkor ha a program 0%-ban pörgeti a cpu-t az adott szálon. Az elv az előzőhöz hasonló. Annyi a különbség hogy ebben a kidolgozásban for ciklust használtam ezzel jelezve hogy szándékos a végtelen ciklus. Ahhoz hogy 0%-on pörögjön a cpu a **sleep** parancsot kell használni. Ehhez viszont szükség van az **unistd.h** függvénykönyvtárra. A sleepnél a szám azt határozza meg hogy hány másodpercig sleep-el, viszont a végtelenség miatt ez soha nem fog abba maradni.

```
Végtelen ciklus 1 szál 0%
{

#include <unistd.h>

int main ()
{
    for (;;)
        sleep (1);
    return 0;
}
}
```

Végül pedig az utolsó kód azt mutatja be amikor az összes szálon 100%-ra pörög a cpu. Ehhez az kell hogy a ciklus elé beillesszük a **#pragma omp parallel** parancsot. Ez biztosítja számunkra azt hogy a ciklus minden szálon fut. Viszont hogy ez működjön az is kell hogy a futtatásnál a **-fopenmp** kapcsolót is használjuk.

```
Végtelen ciklus összes szál 100%
{

#include <stdbool.h>

//gcc vegtelenossz.c -fopenmp -o vegossz

int main()
{
    #pragma omp parallel
    while (true)
    {
        ;
    }
    return 0;
}
}
```

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 gondolatmenet az hogy kellene egy olyan program ami ellenőrzi hogy a megadott program lefagy-e vagy nem. Ilyen programot nem lehet készíteni. Elméletben azonban mégis készítsük el azt a programot amely képes erre. Ha lefagy a paraméterül kapott program akkor írja ki hogy lefagy, ha pedig nem akkor indítson egy végtelen ciklust.

A képzelt program elkészült. Most adjuk meg neki önmagát. Teszteljük hogy lefagy-e vagy nem. Mivel önmagát kapta meg és azt tudjuk hogy ez a program nem fagy le ezért indít egy végtelen ciklust, tehát lefagy. Ez viszont ellentmondás. Így tehát lehetetlen olyan programot készíteni ami eldönti minden kapott programról hogy lefagy-e vagy nem.

2.3. Változók értékének felcserélése

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/csere2.c

A feladat az volt hogy készítsünk egy olyan programot ami két változó értékét felcseréli bármilyen logikai utasítás és segédváltozó nélkül. Egy ilyen kódot úgy lehet elkészíteni ha matematikai módszereket alkalmazva cseréljük ki a két számot. Az alább látható kód ezt mutatja be. Bekérünk a felhasználótól két számot majd összeadás és kivonás segítségével felcseréljük a két számot. Ahhoz viszont hogy a standart outputra tudjunk írni és onnan olvasni szükség van az **stdio.h** függvénykönyvtárra.

```
Változó csere matematikai műveletekkel
{
#include <stdio.h>

int main()
```

```
{
    int a = 0;
    int b = 0;
    printf("Adja meg az a szamot: ");
    scanf("%d" , &a );
    printf("Adja meg a b szamot: ");
    scanf("%d" , &b);
    b = b-a;
    a = a+b;
    b = a-b;
    printf("a=%d%s",a, "\n");
    printf("b=%d%s",b, "\n");
}
}
```

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írj egy olyan programot, ami egy labdát pattogtat a karakteres konzolon!

Megoldás forrása:https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Turing/Labda

Az alábbi program a standart outputon imitálja egy labda pattogását. Ehhez szüksége van több függvénykönyvtárra. A lentebb látott kódban láthatóak. Ezek közül már van amelyiket néztük. Most az új a **curses.h**. Erre azért van szükség hogy a kódban látható WINDOW, initscr, getmaxyx és refresh használható legyen. A futtatás során pedig szükség lesz a **-lncurses** kapcsolóra. A részletesebb kód magyarázat pedig a kommentekben olvasható.

```
Labdapattogás if-el
{
#include <stdio.h>
#include <curses.h>
#include <unistd.h>

//gcc labdaif.c -o labda -lncurses

int
main ( void )
{
    WINDOW *ablak;    //létrehozza az ablakot
    ablak = initscr ();    //kiszámolja az ablak méretét

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;
```

```

int mx;
int my;

for ( ;; ) {

    getmaxyx ( ablak, my , mx );    //átadja az my-nak az ablak ←
    magasságát, mx-nek a szélességét

    mvprintw ( y, x, "O" );    //kiírja a labdát

    refresh ();    //meg kell hívni hogy kimenetet kapjunk a ←
    terminálra
    usleep ( 100000 );    //lassítja a labda mozgását micro second-be ←
    mérve
    clear();    //letisztítja az ablakot

    x = x + xnov;    // labda vízszintes koordinátája
    y = y + ynov;    // labda függőleges koordinátája

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

}

return 0;
}
}

```

Nézzük az if nélküli verziót. Az **stdlib.h** könyvtárra szükség lesz az abs függvény használatához. A **define** sorokban határozzuk meg az ablak méretét melyben a labda pattogni fog. A putX egy függvény melyet mi készítettünk és a mainbe kerül majd meghívásra. A koordinátákat kell neki megadni és azok segítségével kiírja a labdát. A mainbe mikor meghívásra kerül akkor használjuk benne az abs függvényt. Erre azért van szükség hogy a távolságok ne lépjenek ki a képernyőről. A többi magyarázatot lásd a kód kommentelésében.

```

Labdapattogás if nélkül
{

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

```

```
#include<unistd.h>

#define SZEL 78      //meghatározza az ablak szélességét
#define MAG 22      //meghatározza az ablak magasságát

int putX(int x,int y)
{
    int i;

    for(i=0;i<x;i++)      // meghatározza a labda függőleges koordinátáját az ↵
        ablakon belül
    printf("\n");

    for(i=0;i<y;i++)      // meghatározza a labda vízszintes koordinátáját az ↵
        ablakon belül
    printf(" ");

    printf("X\n");      //kiírja a labdát és tesz egy új sort

    return 0;
}

int main()
{
    int x=0,y=0;      //a kezdeti koordinátákat beállítja nullára

    while(1)      //végtelen ciklus indul
    {
        system("clear");      // az ablakot tisztítja
        putX(abs(MAG-(x++%(MAG*2))),abs(SZEL-(y++%(SZEL*2))));      //kiszámolja a ↵
            labda aktuális helyét és kiírja a labdát
        usleep(50000);      //meghatározza a labda sebességét
    }

    return 0;
}
```

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Turing/Szohossz_BogoMIPS

A program végig shifteli bitenként az **integer** változót és a végén kiírja hogy hány biten tárolja. Ehhez szükség van a **bitshift** operátorra. Ez egyessével balra shifteli az intet amíg az utolsó bit is nem lesz 0. Ekkor végeredményként kiírja hogy 32 biten tárolja.

```
Szóhossz
{
#include <stdio.h>
int main()
{
    int a=1;
    int n=1;
    while(a<=1)
    {
        n+=1;
    }
    printf("Megoldas:%d%s", n, "\n");
}
}
```

2.6. Helló, Google!

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/pagerank.c

Amikor a Google megalakult a page-rank-ot használták fel arra hogy az oldalakat rangsorolják. Ez alapján állították fel a sorrendet az eredmények megjelenítésekor. A jobb értékkel rendelkező oldal feljebb került a listában így a relevánsabb adatok előrébb kerültek. A page-rank ugyanis azt csinálja hogy az egyes honlapokat állítja sorba az alapján hogy mennyire releváns a rajtuk szereplő információ. A rangsoroláshoz minden honlapnak kiszámít egy értéket és ezeket hasonlítja össze. Az egyes értékeket pedig úgy számolja ki hogy veszi az adott oldalra mutató linkek számát. Valamint összeadja azt hogy az oldalra mutató honlapokból hány link indul ki és ezeket összeadja. Végül a két értéket elosztja egymással. Az ehhez kapcsolódó kód lentebb látható a magyarázattal egybe.

```
Page-Rank
{
#include <stdio.h>
#include <math.h>

void kiir (double tomb[], int db)    //kiírja az eredmény tömböt, a db felel ←
    azért hogy a ciklus hányszor fut le a függvényen belül
{
    int i;
    for (i=0; i<db; i++)
        printf("PageRank [%d]: %lf\n", i, tomb[i]);
}

double tavolsag(double pagerank[], double pagerank_temp[], int db)
{
    double tav = 0.0;
    int i;
```

```
for (i=0; i<db; i++)
{
    if ((pagerank[i] - pagerank_temp[i])<0)
    {
        tav += (-1*(pagerank[i] - pagerank_temp[i]));
    }
    else
    {
        tav += (pagerank[i] - pagerank_temp[i]);
    }
}
return tav;
}

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

    double PR[4] = {0.0, 0.0, 0.0, 0.0}; // ebben lesz benne a végeredmény
    double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0}; //az ↔
        egyes oldalak preztizse

    long int i, j; // ciklus számláló
    i=0; j=0;

    for (;;)
    {
        for (i=0; i<4; i++) //átmásolja a PR-be a PRv-t
            PR[i] = PRv[i];
        for (i=0; i<4; i++) //az L linkmátrixot összeszorozza a PR vektorral
        {
            double temp=0;
            for (j=0; j<4; j++)
                temp+=L[i][j]*PR[j];
            PRv[i]=temp; //a mátrix szorzás eredményét eltárolja a PRv-ben
        }

        if (tavolsag(PR, PRv, 4) < 0.00001) //a feltétel ha teljesül akkor ↔
            belép a az if magjába
            break; //ebben az esetben lép ki a végtelen ciklusból
    }
    kiir (PR, 4); //meghívja a kiir függvényt és kiírja az eredményt
    return 0;
}
```



```
}
```

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/brun.r

A feladat megoldásához meg kell értenünk mik is azok a prímszámok és az ikerprímek. A prímszámok olyan számok amiknek csak két osztójuk van, az 1 és önmaga. Az ikerprímek azok is prímszámok, különlegességük hogy a két szám közötti távolság kettő. Ilyen számpárokról a mai napig nem tudni hogy végtelen sok van-e. A Brun tétel pedig ezekkel a számpárokkal foglalkozik. A tétel azt mondja ki hogy az ikerprímek reciprokának összege a Brun konstanshoz konvergál. Az alábbi kód ennek bemutatására készült.

```
library(matlab)

stp <- function(x) {

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

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/monty.r

A Monty Hall egy valószínűségi problémát vizsgál. Az elv az hogy három lehetőség közül kell választanunk úgy hogy egy jó választás van, a másik két esetben rosszabbul járunk. Ha választottunk egyet utána a másik kettőből egyet megnézünk és kiderül hogy az egy rossz választás. Utána pedig meg van a lehetőség arra hogy újra válasszunk. A kérdés az hogy megéri a két fent maradó lehetőségéből a másikat választani vagy inkább maradjunk az eredeti választásnál. Az elvi megoldás az hogy megéri váltani mivel akkor nagyobb valószínűséggel választjuk a jó megoldást. Ez viszont ellent mond az emberi gondolkodásnak ezért ez paradoxonhoz vezet. A kód pedig erre a problémára ad egy szemléltetést.

```
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)
```

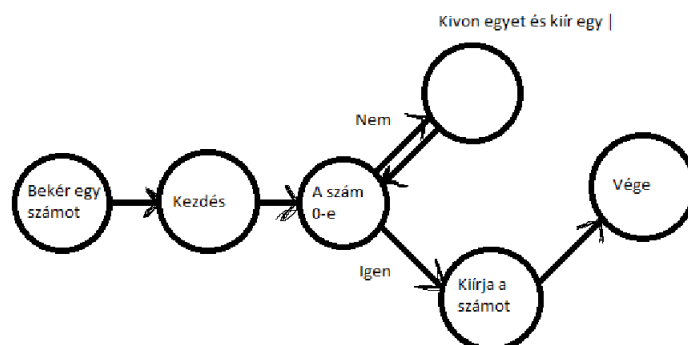
3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

A decimálisból unárisba váltó Turing gép tizes számrendszerből vált egyesbe. Az unáris számrendszer csak arra alkalmas hogy természetes számokat ábrázoljon mivel csak egy karakter létezik benne. Mondjuk egy | karakter. Ezzel pedig nem tudunk mást leírni csak "egyszerű"(természetes) számokat. Tehát mondjuk a 6 ábrázolása a következő képpen néz ki:|||||.



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

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

A formális nyelvtannak több fajtája is van. Van az általános, a környezetfüggő, a környezetfüggetlen és a reguláris. Ezek közül mi a környezetfüggő generatív grammatikával fogunk most foglalkozni. Ez a fajta generatív nyelvtan egy rendezett négyesre épül. A négyes elemei: nemterminális jelek, terminális jelek, előállítási szabályok, produkciós szabályok. A nemterminális jelek között van egy kitüntetett elem, a kezdő elem. Jelen helyzetben ez nem más mint az S.

```
S, X, Y "nemterminális jelek"
a, b, c "terminális jelek"
S -> abc, S -> aXbc, Xb -> bX, Xc -> Ybcc, bY -> Yb, aY -> aaX, aY -> aa
S: innen indulunk
```

```
-----
S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
aYbbcc (aY -> aa)
aabbcc
```

```
-----
S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
aYbbcc (aY -> aaX)
aaXbbcc (Xb -> bX)
aabXbcc (Xb -> bX)
aabbXcc (Xc -> Ybcc)
aabbYbcc (bY -> Yb)
aabbYbbcc (bY -> Yb)
aaYbbbcc (aY -> aa)
aaabbcc
```

```
A, B, C "nemterminális jelek"
a, b, c "terminális jelek"
A -> aAB, A -> aC, CB -> bCc, cB -> Bc, C -> bc
S: innen indulunk
```

```
-----
A (A -> aAB)
aAB (A -> aC)
aaCB (CB -> bCc)
aabCc (C -> bc)
aabbcc
```

```
-----
A (A -> aAB)
aAB (A -> aAB)
aaABB (A -> aAB)
aaaABBB (A -> aC)
```

```
aaaaCBBB (CB -> bCc)
aaaabCcBB (cB -> Bc)
aaaabCBcB (cB -> Bc)
aaaabCBBc (CB -> bCc)
aaaabbCcBc (cB -> Bc)
aaaabbCBcc (CB -> bCc)
aaaabbbCccc (C -> bc)
aaaabbbbcccc
```

3.3. Hivatkozási nyelv

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/Hivatkozasi_nyelv

A feladat az volt hogy mutassuk meg mi a különbség a C89 és C99 verzió között. Ezek a C programozási nyelv két különböző változata. A C99-es egy újabb verzió a 89-hez képest. Ebből adódóan találhatók benne újítások melyek még a C89-ben nem szerepelnek. Ilyen például az hogy a 89-es verzióban még volt lehetséges az hogy a for ciklus fejében deklaráljuk a futó változót. Ebben a verzióban még a ciklus előtt deklarálni kellett és a ciklus fejben csak értéket adtunk neki. A 99-es verzióban viszont már lehetséges volt a fejben deklarálni a futó változót. A futtatás során pedig a **-std=c89** és **-std=c99** kapcsolóval lehet kényszeríteni a fordítót hogy ezeket a verziókat használja.

```
C89
{

#include <stdio.h>

//gcc -std=c89 c89.c -o c89

int main()
{
    int i;
    for (i = 0; i < 5; i++)
    {
        printf("%d ", i+1);
    }
    printf("\n");
    return 0;
}
}
```

```
C99
{

#include <stdio.h>
```

```
//gcc -std=c99 c99.c -o c99

int main()
{
    for (int i = 0; i < 5; i++)
    {
        printf("%d ", i+1);
    }
    printf("\n");
    return 0;
}
```

3.4. Saját lexikális elemző

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/Lexik%C3%A1lis_elemzo

A program bekér egy szöveget a felhasználótól. Ezen lefuttatja a lexert és a szövegben található összes számot "kiemeli" és átalakítja számmá(tehát számként fogja kezelni nem pedig szöveggént). A kilépéskor - melyhez a (ctrl + d) kombinációt használjuk - pedig kiírja hogy hány szám szerepel a szövegbe. Ennek a feladatnak az a lényege hogy ne a saját kódunkat használjuk hanem a lexert használjuk tehát óriások vállán álljunk.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/l33t

A leet egy olyan program amely arra való hogy a beírt szavakban egyes karaktereket kicserél más karakterekre de úgy hogy a szöveg továbbra is olvasható marad. Ennek az online beszélgetésekben van nagy haszna. Hiszen ha egy olyan szót akarunk beírni ami az adott oldal szűrőin nem megy át akkor egy kis módosítással ez megkerülhető. Például ha egy szóban szerepel a betű akkor azt ha kicseréljük mondjuk @ jelre akkor az még érthető ám a szűrő már átengedi és nem fogja cenzúrázni.

A program kódja megadja hogy milyen karaktereket mikre lehet cserélni. Egy tömbben tároljuk az erre vonatkozó információkat. Valamint azt is eltároljuk hogy a beírt szöveg milyen hosszú és ezután végig megyünk a teljes szövegen és átírjuk benne a karaktereket if-ek segítségével.

```
%{
#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", "|\\"}},
    {'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"}},

    // https://simple.wikipedia.org/wiki/Leet
};

```

```
%}
%%
. {

    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;
}
```


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, jelkezelő);
```

A fenti kódhoz képest itt az egyenlőség vizsgálata le lett cserélve a nem egyenlőre. Ez a kicsi változás pedig elég ahhoz hogy az eredmény az ellentéte legyen az eredetinek.

ii.

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

Itt egy for ciklus fog ötször lefutni. A ++i-nek köszönhetően az i értéke előbb fog nőni és csak utána fog kiértékelődni.

iii.

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

Itt egy for ciklus fog ötször lefutni. A i++-nak köszönhetően az i értéke előbb fog kiérőni és csak utána fog nőni.

iv.

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

Ebben a példában is egy for ciklus fog ötször lefutni. Viszont itt már egy tömbben el is tároljuk az i++ értékeit. Első helyen egy memória szemét lesz utána viszont az i++ értékei kerülnek tárolásra.

v.

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

Ebben a kód részletben a for ciklus feltételében egy összekapcsolt feltétel van. Viszont ez a feltétel hibás. Hiszen (*d++ = *s++) ebben a részben értékadó egyenlőség van nem pedig összehasonlító. Így tehát ez nem ad vissza logikai értéket ezért nem lehet eldönteni hogy a feltétel teljesül vagy nem.

vi.

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

Az f() függvény értékét íratjuk ki. Ám mivel ebben a kód részletben nem látjuk magát a függvényt azt viszont igen hogy a megadott értékek felcserélőők így arra lehet következtetni hogy az f()-ben valamilyen kommutatív dolog van.

vii.

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

Itt a printf-el két számot íratunk ki decimális formában. Az egyik az f() által visszaadott érték a másik meg az a paraméter amit a függvénynek átadtunk.

viii.

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

Két szám kiírása történik. Egy memóriacím általi érték és az a változó.

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/forras_olvasas

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

Az Ar egy logikában használatos nyelv mely a matematikai nyelvet foglalja magába.

\text = szöveg kiírása

\forall = univerzális kvantor

\exists = egzisztenciális kvantor

\wedge = konjunkció

\vee = diszjunkció

\neg = negáció

\supset = implikáció

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

Minden x-re létezik olyan y hogy x kisebb mint y és y prím.

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

Minden x-re létezik olyan y hogy x kisebb mint y és y prím és y kettőre \leftrightarrow következője is prím.

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

Létezik y esetén minden x, ha x prím akkor x kisebb mint y.

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

Létezik y esetén minden x, ha y kisebb mint x akkor x nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

  
egész
- ```
int *b = &a;
```


egy mutató ami egy memóriacímre mutat
- ```
int &r = a;
```

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


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

  
egészek tömbjének referenciája (nem az első elemé)
- ```
int *d[5];
```


egészre mutató mutatók tömbje
- ```
int *h ();
```

  
egészre mutató mutatót visszaadó függvény
- ```
int *(*l) ();
```


egészre mutató mutatót visszaadó függvényre mutató mutató
- ```
int (*v (int c)) (int a, int b)
```

  
egészre visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény
- ```
int ((*z) (int)) (int, int);
```


függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/H%C3%A1romsz%C3%B6gm%C3%A1trix>

Ebben a programban alsó háromszögmátrixot hozunk létre. Hogy ezt létre tudjuk hozni először is tisztáznunk kell hogy mi az az alsó háromszögmátrix. Ehhez pedig tudnunk kell hogy mi az a mátrix. A mátrix egy olyan vektor melynek két dimenziója van. Tehát rendelkezik sorokkal és oszlopokkal is. Az alsó háromszögmátrix pedig egy olyan speciális fajtája ennek amelyben a főátló feletti összes érték 0, valamint a mátrix négyzetes.

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

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

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

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

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

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

```
        return -1;
    }

}

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

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

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

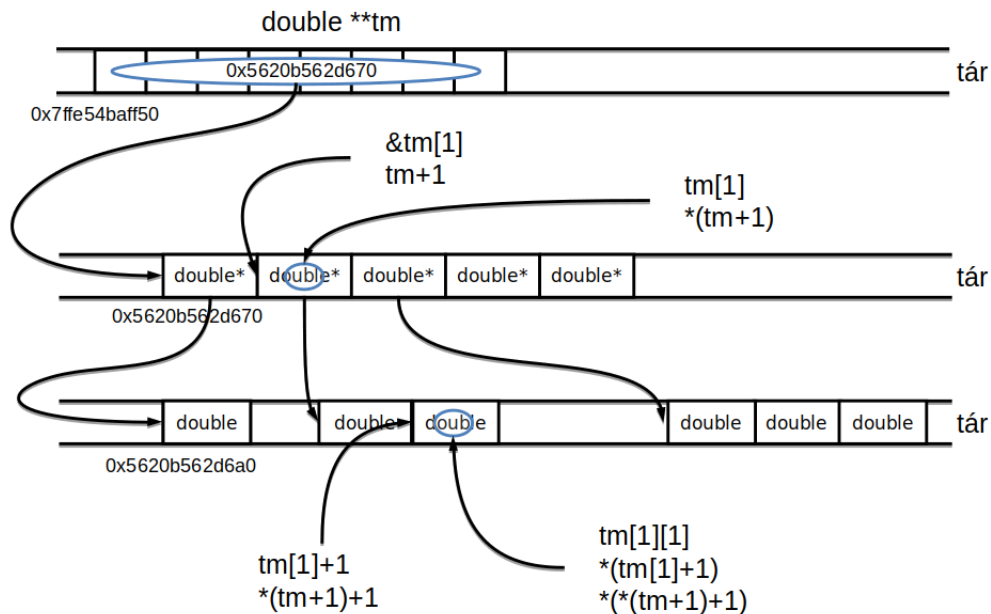
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0; // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

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

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

free (tm);

return 0;
}
```



4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/C_titkos%C3%ADt%C3%B3

Ez a fajta titkosító módszer nagyon régi és elég egyszerű elven alapul. Van egy titkosítandó szövegünk és egy kulcsunk amivel titkosítunk. A módszerlényege hogy a kulcsot - ami lehet egy akármilyen karakterlánc - átváltjuk bitekre és a szöveget is. Ezután a szövegen és a kulcson végrehajtja a kizáró vagyot. Ez úgy működik hogy a szöveg alá helyezzük a kulcs biteit és ahol a bitek megegyeznek oda a titkosítás során 0-t írunk ahol különböznek oda pedig 1-est. A művelet végrehajtása után pedig egy titkosított szöveget kapunk ami értelmezhetetlen. Ahhoz hogy ismét olvashatóvá váljon a kulcs segítségével vissza kell fejteni az eredeti szöveget.

Az alábbi programban erre a titkosítóra látunk egy konkrét kódot. Úgy működik hogy a program megkapja a kulcsot és a titkosítandó szöveget. A szöveget elkezd beolvasni a bufferbe és a kulcs segítségével elkezd a titkosítást. Mikor végzett az adott résszel, akkor azt kiírja egy eredmény fájlba és beolvassa a szöveg következő részét a bufferbe. Ezt a folyamatot addig ismétli amíg a teljes szöveget nem titkosítja.

```
#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]; //ebben a változóban tároljuk a kulcsot
char buffer[BUFFER_MERET]; //ide olvassa be a szöveget

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]; //itt hajtódik végre a ↔
        titkosítás
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }

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

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/Java_titkos%C3%ADt%C3%B3

Az előző feladathoz hasonlóan ez a program is az EXOR-os titkosítást végzi el. A különbség annyi hogy ez a kód java nyelven van írva és tartalmaz objektum orientált részeket. Ilyen például a class. További különbség hogy ez a kód a titkosítandó szöveget nem egy fájlból olvassa be hanem a felhasználónak kell begépelni a standart inputon keresztül. A program további működési elve ugyanaz.

```
public class ExorTitkosító {

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

```
byte [] kulcs = kulcsSzöveg.getBytes();    //ebben a változóban ←  
        tároljuk a kulcsot  
byte [] buffer = new byte[256];           //ide olvassa be a szöveget  
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]);    //itt ←  
            hajtódik végre a titkosítás  
        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();  
  
    }  
  
}  
  
}
```

4.4. C EXOR törő

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/C_t%C3%B6r%C5%91

Ebben a részben az előző két feladat ellentéte szerepel. AMíg ott a titkosítás volt a cél addig ennél a feladatnál a feltörés a cél. A program megkapja a titkosított szöveget és elkezd keresni a feloldáshoz

szükséges kulcsot, amely 5 karakter hosszú és az angol abc betűit tartalmazza. A törés elve a brute force módszer. Egyesével kipróbál minden lehetséges kulcsot amíg meg nem találja a megfelelőt. Amikor végzett a töréssel akkor pedig teszteli is az eredményt. A `tiszta_lehet` azt vizsgálja hogy a megadott kulcsszavak szerepelnek-e a tiszta szövegbe, ha igen akkor valószínűleg helyes a törés.

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 5
#define _GNU_SOURCE

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

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

    return szohossz > 6.0 && szohossz < 9.0
        && strcasestr (titkos, "hogy") && 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 tiszta_lehet (titkos, titkos_meret);
}

int
main (void)
{
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;
    //itt tároljuk azokat a karaktereket amik a kulcs karakterei lehetnek
    char kod[28]= {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o' ←
        ',','p','q','r','s','t','u','v','w','x','y','z'};

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

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

    // osszes kulcs eloallitasa
    for (int ii = 0; ii <= 28; ++ii)
        for (int ji = 0; ji <= 28; ++ji)
            for (int ki = 0; ki <= 28; ++ki)
                for (int li = 0; li <= 28; ++li)
                    for (int mi = 0; mi <= 28; ++mi)
                    {
                        kulcs[0] = kod[ii];
```

```
kulcs[1] = kod[ji];
kulcs[2] = kod[ki];
kulcs[3] = kod[li];
kulcs[4] = kod[mi];

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

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

return 0;
}
```

4.5. Neurális OR, AND és EXOR kapu

R

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/Neur%C3%A1lis>

Ebben a feladatban az agy feldolgozó részét képző neurális háló szimulálása a feladat. A neurális háló lényege hogy képes arra, hogy párhuzamosan információt dolgozzon fel és adott szabályok alapján tanuljon. Megkapja tőlünk az a1 és a2 vektorokat valamint hogy az OR műveletet végrehajtva rajta milyen eredményt kell neki adnia. Ez után pedig megkezdődik a tanulás. A kapott adatok alapján megtanulja hogy az OR művelet esetén milyen eredményt kell neki visszaadnia.

Az AND művelet esetén szintén ez a folyamat megy végbe. Megkapja az alap eseteket és az alapján megtanulja hogy adott bemenet esetén milyen eredményt kell neki adnia.

Az EXOR is hasonlóan működik bár ott van egy kis eltérés. Az eltérés annyiban jelentkezik hogy ott többretegű neuronokkal kell dolgozni.

```
library(neuralnet)

a1 <- c(0,1,0,1) #az első vektor értékei
a2 <- c(0,0,1,1) #a második vektor értékei
OR <- c(0,1,1,1) #az OR művelet eredményei

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) #az első vektor értékei
a2 <- c(0,0,1,1) #a második vektor értékei
OR <- c(0,1,1,1) #az OR művelet eredményei
AND <- c(0,0,0,1) #az AND művelet eredményei

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) #az első vektor értékei
a2 <- c(0,0,1,1) #a második vektor értékei
EXOR <- c(0,1,1,0) #az EXOR művelet eredményei

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)
```

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

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: <https://github.com/nbatfai/nahshon/blob/master/ql.hpp#L64>

Ennél a feladatnál egy kép segítségével a bináris osztályozást fogjuk "megtanítani" a számítógépnek. A program kapni fog egy png fájlt amelyen a Mandelbrot halmaz látható. Ezt a képet adjuk át a perceptronnak és meghatározzuk a rétegek számát a neurális hálóba. Valamint még azt is meg kell adnunk hogy az egyes rétegeken hány neuront szeretnénk létrehozni. Ha ez megvan akkor elindul a program és a neurális háló segítségével megtanulja, és végrehajtja a bináris osztályozást a képen. A megtanuláshoz mint ahogy az előző feladatban is szó volt róla meg kell adni neki az alap eseteket és ebből fogja tudni mit hogyan osztályozzon. A bináris osztályozás jól prezentálható egyesekkel és nullákkal.

5. fejezet

Helló, Mandelbrot!

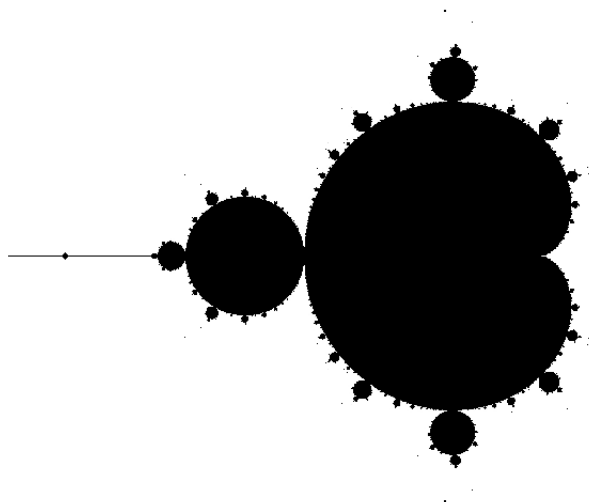
5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Mandelbrot/Mandelbrot>

A Mandelbrot halmaz bemutatásához és annak megértéséhez szükséges hogy egy minimális szinten ismerjük a komplex számokat. A komplex számokat már nem egy számegyenesen ábrázoljuk hanem egy szám síkon. Ez a szám halmaz arra lett kitalálva hogy egy olyan értéket is ki lehessen számolni amit alap esetben nem tudnánk. Ez pedig nem más mint a gyök alatti mínusz érték. A komplex számok körébe értelmezhető már az ilyen kifejezés is. Ez pedig úgy lehetséges hogy a gyök alatt mínusz egy egyenlő lesz az i értékkel. Ebből pedig már fel lehet építeni bármilyen negatív számot.

Maga a Mandelbrot halmaz egy olyan számhalmaz melyben a komplex számokat az alábbi képlettel számoljuk ki: $z_{n+1} = z_n^2 + c$, ($0 \leq n$). A képlet segítségével kapott számok lényege hogy abszolút értékük korlátos, azaz nem tartanak a végtelenbe.



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

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Mandelbrot/Complex_osztaly

Ebben a feladatban az előző feladatban megismert Mandelbrot halmaznak a kódja található az `std::complex` segítségével. Ahhoz hogy a `complex` osztály függvényei használhatóak legyenek a kódban include-olni kell a `complex-et`. A program forrása lentebb tekinthető meg.

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

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

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

    if ( argc == 9 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        a = atof ( argv[5] );
        b = atof ( argv[6] );
        c = atof ( argv[7] );
        d = atof ( argv[8] );
    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↵
            " << std::endl;
        return -1;
    }

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

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

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

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

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

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

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

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

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

            ++iteracio;
        }

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

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

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

5.3. Biomorfok

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Mandelbrot/Biomorf>

A biomorfok másnéven Julia halmazok a Mandelbrot halmazokhoz köthető. Ezen halmazok összefüggnek a Mandelbrot halmazokkal, még pedig úgy hogy a Mandelbrot-os képletben annyi változtatást kell

eszközölni hogy a c-t ami ott egy változó érték volt azt ebben az esetben állandó értékre kell állítani.

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

int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag =  atoi ( argv[3] );
        iteraciosHatar =  atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );
    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↵
            d reC imC R" << std::endl;
        return -1;
    }

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

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

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

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

```

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

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

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

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

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

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

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

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

```

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Mandelbrot/CUDA>

Ebben a feladatban már a korábban megismert Mandelbrot halmazt számoljuk ki ismét. Annyi a megoldásban a változás hogy most a grafikus kártya CUDA magjait használjuk hozzá. Azáltal hogy a CUDA magokat használjuk párhuzamosan nem a processzor egy magját, a számítási műveletek lényegesen gyorsabbak lesznek. Akár 50-70 százalékos gyorsulást is tapasztalhatunk úgy hogy ezt a megoldást használjuk.

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

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

#define MERET 600
#define ITER_HAT 32000

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

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

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

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

```
        imZ = ujimZ;

        ++iteracio;

    }
    return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{

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

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

}
*/

__global__ void
mandelkernel (int *kepadat)
{

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

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

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

}

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

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

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

    dim3 grid (MERET / 10, MERET / 10);
    dim3 tgrid (10, 10);
    mandelkernel <<< grid, tgrid >>> (device_kepadat);
```

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

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

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

    int kepadat[MERET][MERET];

    cudamandel (kepadat);

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

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

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

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

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

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Mandelbrot/Nagyito>

Ennek a feladatnak a megoldásához szükségünk van egy grafikus interfészre, ami nem lesz más mint a QT GUI. Ez egy nagyon elterjedt interfész a C++ nyelvben.

A program ezen megoldásában az a lényeg hogy úgy hozzuk létre a Mandelbrot halmazt hogy az "változzon". Ezt úgy kell érteni hogy ha elkezdjük nagyítani a képet akkor a program újra számolja a Mandelbrot halmazt. Ugyanis ez a halmaz végtelen, aminek egyszerre véges sok elemét számoljuk ki. Viszont ha nagyítunk rajta akkor ott újabb értékek kerülnek kiszámításra. Ha eléggé bele nagyítunk akkor észre fogjuk venni hogy újabb Mandelbrot halmaz jelenik meg az eredetin belül és ez ismétlődik a végtelenségig.

A program futtatásának menete a következőképpen néz ki. Először is telepíteni kell a **sudo apt-get install libqt4-dev**-et. Ez után futtatni kell egy parancsot mely a **qmake -project**. Ha ez megvan akkor a létrejött fájlt meg kell nyitni és abba bele kell illeszteni a következő sort: **QT += widgets**. Majd mentjük a módosítást és kilépünk. Ez után egy újabb parancsot futtatunk ami a **qmake mandelmozzgatoc++.pro**. Ez készít egy make fájlt. Ezt a létrejött fájlt futtatjuk úgy hogy egyszerűen beírjuk a make parancsot a terminálba. Ha ezzel is kész vagyunk akkor pedig a létrejött fájlt már a szokásos ./fájlnév paranccsal futtatjuk.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Mandelbrot/java_nagyito

A feladat megoldása nagyban hasonlít az előzőhöz. A különbség annyi hogy ebben a kódban nem a C++ nyelvet használjuk hanem a java-t. Az eredmény ugyan az lesz mint a C++-os verziónál csak itt a java-s megoldásokat használuk mivel itt azok fognak működni.

A kód futtatásához fel kell telepíteni a **sudo apt-get install openjdk-8-jdk**-t. Ez után két egyszerű sor a terminálba és kész is van. Az első: **javac MandelbrotHalmazNagyító.java**. A második: **java MandelbrotHalmazNagyító**. A második sor beírása után meg is nyílik az ablak benne a Mandelbrot halmazzal. A nagyítás itt is úgy működik mint a C++-os verziónál, annyi a különbség hogy ebben a megoldásban van egy kis hiba. Amikor nagyítunk, a nagyított kép mindig új ablakban nyílik meg.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Welch/Polargen>

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!

A feladat megoldását három C++ fájlban oldottuk meg. A main.cpp, polargen.cpp és polargen.h fájlokban.

Az első fájlunk a main.cpp. Itt hogy használni tudjuk a másik két fájlt be kellett include-olni a polargen.h-t. Most térjünk rá magára a main függvényre. Itt először létrehozunk egy pg változót ami PolarGen osztályú. Azután pedig erre a változóra tízszer meghívjuk a kovetkezo függvényt és az eredményt kiíratjuk a standard outputra.

```
#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 következő fájlunk a polargen.cpp. Ebben a fájlban van kidolgozva a header fájlban deklarált következő nevű függvény. Ennek a függvénynek két visszatérítési értéke lesz. Az egyik az $r \cdot v_2$, a másik az $r \cdot v_1$. Az $r \cdot v_1$ akkor lesz a visszatérítési érték ha nincs tárolt adat, az $r \cdot v_2$ pedig akkor ha van tárolt adat. Az r és a v értékek kiszámításának képlete a kódban látható. A képletben látható rand függvény pedig egy véletlenül generált számot ad meg.

```
#include "polargen.h"

double
PolarGen::kovetkezo ()
{
    if (nincsTarolt)
    {
        double u1, u2, v1, v2, w;
        do
        {
            u1 = std::rand () / (RAND_MAX + 1.0);
            u2 = std::rand () / (RAND_MAX + 1.0);
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);

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

        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

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

Utolsó C++ fájlunk pedig nem más mint a polargen.h. Ebben a fájlban kerül létrehozásra a PolarGen osztály. Az osztály public részében adjuk meg a nincsTarolt változó alapértelmezett értékét, az srand alap értékét, valamint itt kerül létrehozásra a következő nevű függvény. Továbbá az osztály private részében kerül sor a nincsTarolt változó létrehozásra és a tarolt változó létrehozásra. Az hogy ezeket a változókat a private részben hozzuk létre azt jelenti hogy ezeket nem érjük el csak az osztályon belül.

```
#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
```

A C++-os megoldás után most ugyanezt a programot megoldjuk java nyelven is.

```
public class PolárGenerátor {
    boolean nincsTárolt = true;
    double tárolt;

    public PolárGenerátor() {
        nincsTárolt = true;
    }

    public double következő() {
        if (nincsTárolt) {
            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);
            tárolt = r*v2;
            nincsTárolt = !nincsTárolt;
            return r*v1;
        } else {
            nincsTárolt = !nincsTárolt;
        }
    }
}
```

```
        return tárolt;
    }
}

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

6.2. LZW

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Welch/Binfa_c

Ebben a feladatban egy bináris fát kell felépíteni. A bináris fa egy olyan struktúra melyet rendezésre is szoktak használni. Ilyen fajta rendezés például a kupacrendezés. A bináris fa felépítése a következőképp néz ki. A fa tetején áll egy gyökér elem. Innen indul ki a fa és szintenként épül. A lényege az hogy a fában minden elemnek legfeljebb két rákövetkező eleme lehet. Ettől lesz bináris a fa. Az nem jelent gondot ha nincs mindenkinek kettő következő eleme, de legfeljebb annyi lehet. Azokat az elemeket melyeknek már nincs következő eleme levél elemnek szokták hívni.

A kód amit használunk ennél a programnál az LZW azaz Lempel-Ziv-Welch tömörítési algoritmus, amit Terry Welch publikált 1984-ben. Ez az algoritmus a stringtábla tárolásában hozott nagyfokú újítást, hiszen az eddig használt módszer helyett mostmár csak a bájt csoport első bájtja aztán már csak a táblában szereplő bájt csoport indexe áll. A kód a beérkező adatokból épít egy bináris fát. Elkezd vizsgálni a beérkező bináris adatokat. Ha az érték nulla és még nem szerepel a fában akkor beírja a nullát a bal oldali ágra, ha szerepel akkor pedig rááll és az lesz az új gyökér. Valamint az ő gyermekeit is beállítjuk. Ha az érték 1 akkor pedig ugyanez fog lezajlani csak az 1 a jobb oldali ágra kerül.

```
//>gcc z.c -lm -o z          fordítása

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

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

//>itt definiáljuk a binfa típust
```

```
BINF_PTR
uj_elem ()
{
    BINFA_PTR p;

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

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

int
main (int argc, char **argv)
{
    char b;
    int egy_e;
    int i;
    unsigned char c;
    //>BinfaPTR== user által definiált típus
    BINFA_PTR gyoker = uj_elem ();
    gyoker->ertek = '/';
    gyoker->bal_nulla = gyoker->jobb_egy = NULL;
    BINFA_PTR fa = gyoker;
    long max=0;
    while (read (0, (void *) &b, sizeof(unsigned char)))
    {
        for(i=0;i<8; ++i)
        {
            egy_e= b& 0x80;
            if ((egy_e >>7)==0)
                c='1';
            else
                c='0';
        }
        // write (1, &b, 1);
        if (c == '0')
        {
            if (fa->bal_nulla == NULL)
            {
                fa->bal_nulla = uj_elem ();
                fa->bal_nulla->ertek = 0;
                fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
                fa = gyoker;
            }
        }
    }
}
```

```
    }
    else
    {
        fa = fa->bal_nulla;
    }
}

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

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

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

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

/* Átlagos ághossz kiszámítása */
atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
ratlag (gyoker);
atlag = ((double) atlagosszeg) / atlagdb;

/* Ághosszak szórásának kiszámítása */
atlagosszeg = 0;
melyseg = 0;
atlagdb = 0;
szorasosszeg = 0.0;

rszoras (gyoker);

double szoras = 0.0;

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

```
    szoras = sqrt (szorasosszeg);

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

    szabadit (gyoker);
}

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

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

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

double szorasosszeg = 0.0, atlag = 0.0;

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

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

```
    }

    }

}

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

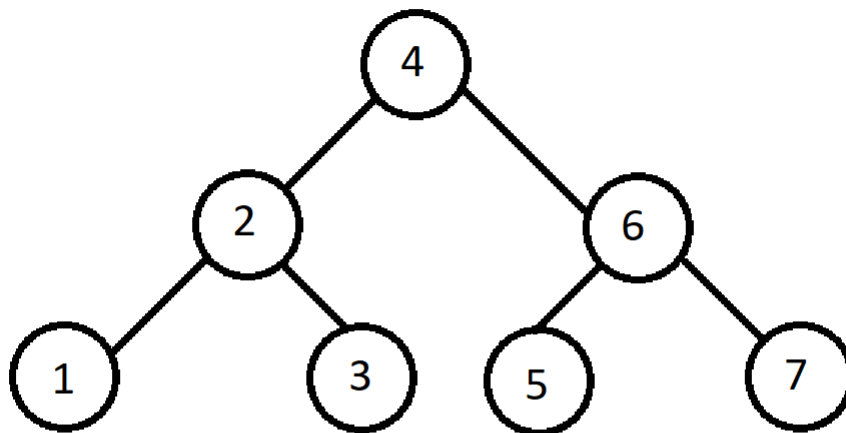
void
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal_nulla);
        free (elem);
    }
}
```

6.3. Fabejárás

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

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Welch/Fabejaras>

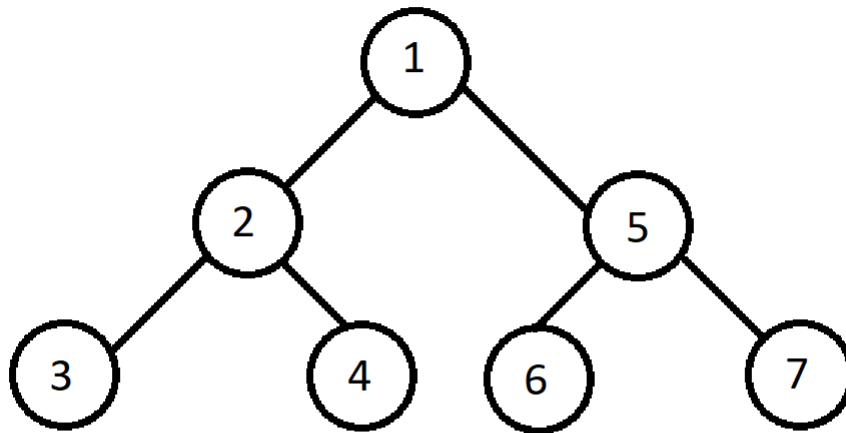
Egy bináris fát három féle módon lehet bejárni. Az első ezek közül az inorder. Általában ezt szokták használni de nem ez az egyetlen bejárési mód. Az inorder bejárás úgy néz ki hogy először a fa bal oldali részfáját járjuk be majd a gyökér elemet végül a fa jobb oldali részfáját.



Ennek a kódja így néz ki:

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

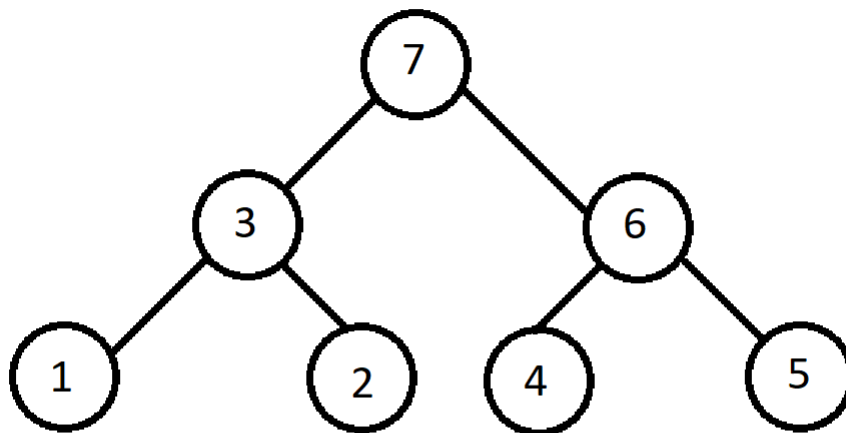
A következő bejárési mód a preorder. Ennek a lényege hogy először a gyöker elem majd a bal oldali részfa végül a jobb oldali részfa bejárása megy végbe. Az alábbi kép ezt mutatja be.



A preorder bejárás kódja:

```
void kiir (Csomopont* elem, std::ostream& os)
{
    if (elem != NULL)
    {
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu() << "(" << melyseg - 2 << ")" << std::endl << "\n";
        ++melyseg;
        kiir (elem->egyenesGyermek(), os);
        kiir (elem->nullasGyermek(), os);
        --melyseg;
    }
}
```

Végül nézzük a harmadik bejárési módot ami a postorder. Itt a bejárás a következőképp alakul. Elsőnek a bal oldali részfat, majd a jobb oldali részfat végül a gyökeret járjuk be.



A postorder bejárás kódja:

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

        kiir (elem->nullasGyermek(), os);
        --melyseg;

        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu() << "(" << melyseg - 2 << ")" << std::endl ←
        ;
    }
}
```

6.4. Tag a gyökér

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Welch/tag_a_gyoker

Az első feladatban már megvalósított binfát kódoljuk most le C++ nyelven. Ebben a forrásban van egy osztályba ágyazott osztály amivel arra akarunk utalni hogy a Csomopont osztályt csak a fán belül értelmezzük. Azon kívül nem akarjuk felhasználni ezért arra ott már nincs is szükség.

A gyoker egy Csomopont osztályba tartozó változó amely tagként szerepel. Ennek azért van jelentősége mivel így a gyoker folyamatosan a memóriában szerepel és így már lehet rá referenciaként hivatkozni.

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

class LZWBinFa
{
public:
    LZWBinFa () : fa(&gyoker) {}

    void operator<<(char b)
    {
        if (b == '0')
        {
            if (!fa->nullasGyermekek ())
            {
                Csomopont *uj = new Csomopont ('0');
                fa->ujNullasGyermekek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->nullasGyermekek ();
            }
        }
        else
        {
            if (!fa->egyenesGyermekek ())
            {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyenesGyermekek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyenesGyermekek ();
            }
        }
    }
};
```

```
    }
}
void kiir (void)
{
    melyseg = 0;
    kiir (&gyoker, std::cout);
}
void szabadit (void)
{
    szabadit (gyoker.egyenesGyermek());
    szabadit (gyoker.nullasGyermek());
}

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 ujEgyenesGyermek (Csomopont * gy) {
        jobbEgy = gy;
    }
    char getBetu() const {
        return betu;
    }
}
```

```

private:
    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &);
    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->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;
    double atlag, szoras;

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

int LZWBInFa::getMelyseg (void)

```

```
{
    melyseg = maxMelyseg = 0;
    rmelyseg (&gyoker);
    return maxMelyseg-1;
}
double LZWBinFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double)atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (&gyoker);

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

    return szoras;
}
void LZWBinFa::rmelyseg (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyGyermek());
        rmelyseg (elem->nullasGyermek());
        --melyseg;
    }
}
void
LZWBinFa::ratlag (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyGyermek());
        ratlag (elem->nullasGyermek());
        --melyseg;
        if (elem->egyGyermek() == NULL && elem->nullasGyermek() == NULL)
        {
```

```
        ++atlagdb;
        atlagosszeg += melyseg;
    }
}

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

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

int
main (int argc, char *argv[])
{
    if (argc != 4) {
        usage();
        return -1;
    }

    char *inFile = *++argv;

    if ((*++argv)+1) != 'o') {
        usage();
        return -2;
    }

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

    unsigned char b;
    LZWBinFa binFa;

    while (beFile.read ((char *) &b, sizeof (unsigned char))) {
        for (int i = 0; i < 8; ++i)
```

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

}

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

binFa.szabadit ();

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

return 0;
}
```

6.5. Mutató a gyökér

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Welch/mutato_a_gyoker

Az előző feladatban használt gyoker elemet most mutatóként fogjuk használni. Ezt úgy lehet megvalósítani hogy az előző kód egy részét módosítjuk. A fa mutatónak ahol az előzőben referenciaként adtuk át a gyokeret most ezt referencia használata nélkül tesszük. Ezen kívül a konstruktorban példányosítjuk és a memóriában is foglalunk neki helyet. Amikor fel akarjuk szabadítani akkor pedig a pont helyett a nyilat használjuk ha a mutató mutatóihoz kell hozzáférni.

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

class LZWBinFa
{
public:
```

```
    LZWBinFa ()
{
    gyoker=new Csomopont();
    fa=gyoker;
};

void operator<<(char b)
{
    if (b == '0')
    {
        if (!fa->nullasGyermeke ())
        {
            Csomopont *uj = new Csomopont ('0');
            fa->ujNullasGyermeke (uj);
            fa = gyoker;
        }
        else
        {
            fa = fa->nullasGyermeke ();
        }
    }
    else
    {
        if (!fa->egyenesGyermeke ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyenesGyermeke (uj);
            fa = gyoker;
        }
        else
        {
            fa = fa->egyenesGyermeke ();
        }
    }
}

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

void szabadit (void)
{
    szabadit (gyoker->egyenesGyermeke());
    szabadit (gyoker->nullasGyermeke());
}

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 *egygyGyermek () const {
        return jobbEgy;
    }
    void ujNullasGyermek (Csomopont * gy) {
        balNulla = gy;
    }
    void ujEgygyGyermek (Csomopont * gy) {
        jobbEgy = gy;
    }
    char getBetu() const {
        return betu;
    }
}
```

private:

```
    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &);
    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;
    int maxMelyseg;
    double atlag, szoras;

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

};

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

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

double LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
```

```
        szorasosszeg = 0.0;
        melyseg = atlagdb = 0;

        rszoras (gyoker);

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

        return szoras;
    }
void LZWBinFa::rmelyseg (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyenesGyermeke());
        rmelyseg (elem->nullasGyermeke());
        --melyseg;
    }
}
void
LZWBinFa::ratlag (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyenesGyermeke());
        ratlag (elem->nullasGyermeke());
        --melyseg;
        if (elem->egyenesGyermeke() == NULL && elem->nullasGyermeke() == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}
void
LZWBinFa::rszoras (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyenesGyermeke());
        rszoras (elem->nullasGyermeke());
        --melyseg;
        if (elem->egyenesGyermeke() == NULL && elem->nullasGyermeke() == NULL)
```

```
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

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

int
main (int argc, char *argv[])
{
    if (argc != 4) {
        usage();
        return -1;
    }

    char *inFile = *++argv;
    if ((*++argv)+1) != 'o' {
        usage();
        return -2;
    }

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

    unsigned char b;
    LZWBinFa binFa;

    while (beFile.read ((char *) &b, sizeof (unsigned char))) {
        for (int i = 0; i < 8; ++i)
        {
            int egy_e = b & 0x80;
            if ((egy_e >> 7) == 1)
                binFa << '1';
            else
                binFa << '0';
            b <<= 1;
        }
    }

    kiFile << binFa;

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

```
    kiFile << "var = " << binFa.getSzoras () << std::endl;

    binFa.szabadit ();

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

    return 0;
}
```

6.6. Mozgató szemantika

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Welch/mozgato_szemantika

A feladat lényege hogy a létrehozott mozgató konstruktor segítségével képesek vagyunk a binfát úgy mozgatni hogy az eredeti fát képesek vagyunk áthelyezni egy másikba. Mivel áthelyezést hajtunk végre így az eredeti fa az áthelyezés végére megszűnik tehát nem lesznek felesleges másolatok a fáról.

Az áthelyezés úgy megy végbe hogy létrehozunk egy új gyökeret és arra áthelyezzük az eredetit. Ez után kiírjuk az új fát és az eredeti nullázásra kerül, tehát ezt már nem is lehet kiírni.

A kódban megtalálható az std::move ami arra szolgál hogy a kódot felkészíti a mozgatásra de magát a mozgatást nem ez a függvény végzi el. Továbbá szerepel még a kódban egy *this parancs is ami arra szolgál hogy az áthelyezett fára állítja a vezérlést, tehát az új már mozgatott binfára.

```
#include <iostream>
#include <cmath>
#include <fstream>
#include <vector>

class LZWBinFa
{
public:
    LZWBinFa ()
    {
        gyoker = new Csomopont ('/');
        fa = gyoker;
    }
    ~LZWBinFa ()
    {
        if (gyoker != nullptr)
        {
            szabadit (gyoker->egyenesGyermekek ());
        }
    }
};
```

```
        szabadit (gyoker->nullasGyermekek ());
        delete(gyoker);
    }
}
LZWBinFa (LZWBinFa&& forras)
{
    std::cout<<"Move ctor\n";
    gyoker = nullptr;
    *this = std::move(forras);
}
LZWBinFa& operator= (LZWBinFa&& forras)
{
    std::cout<<"Move assignment ctor\n";
    std::swap(gyoker, forras.gyoker);
    return *this;
}

void operator<< (char b)
{
    if (b == '0')
    {
        if (!fa->nullasGyermekek ())
        {
            Csomopont *uj = new Csomopont ('0');
            fa->ujNullasGyermekek (uj);
            fa = gyoker;
        }
        else
        {
            fa = fa->nullasGyermekek ();
        }
    }
    else
    {
        if (!fa->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);
    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 ujEgyenesGyermek (Csomopont * gy)
        {
            jobbEgy = gy;
        }
        char getBetu () const
        {
            return betu;
        }
    }
```

```

private:
    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &);
    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->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;
    double atlag, szoras;

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

};

int

```



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

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

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

    rszoras (gyoker);

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

    return szoras;
}

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

void
LZWBinFa::ratlag (Csomopont * elem)
{

```

```
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyenesGyermekek ());
        ratlag (elem->nullasGyermekek ());
        --melyseg;
        if (elem->egyenesGyermekek () == NULL && elem->nullasGyermekek () == NULL <math>\leftrightarrow</math>
            )
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

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

LZWBinFa move(LZWBinFa&& forras)
{
    LZWBinFa b(std::move(forras));
    return b;
}

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

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

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

    char *inFile = *++argv;

    if ((*++argv) + 1) != 'o')
    {
        usage ();
        return -2;
    }

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

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

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

    unsigned char b;
    LZWBinFa binFa;

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

    bool kommentben = false;

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

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

        if (kommentben)
            continue;
    }
```

```
        if (b == 0x4e)
            continue;

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

    }

    kiFile << binFa;

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

    LZWBinFa binFa2 = std::move(binFa);

    kiFile << binFa2;

    kiFile << "depth = " << binFa2.getMelyseg () << std::endl;
    kiFile << "mean = " << binFa2.getAtlag () << std::endl;
    kiFile << "var = " << binFa2.getSzoras () << std::endl;

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

    return 0;
}
```

7. fejezet

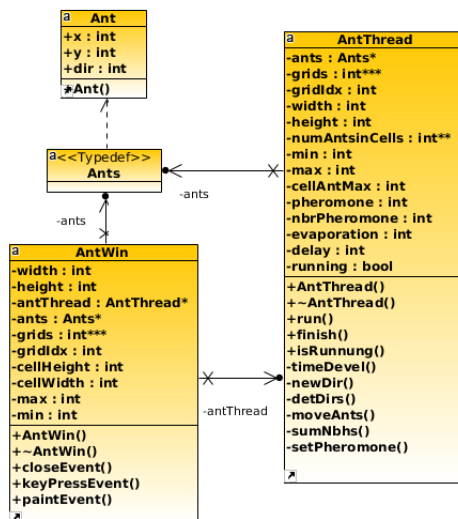
Helló, Conway!

7.1. Hangyaszimulációk

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

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Conway/Hangya>

A feladat mint ahogyan a címében is szerepel egy szimuláció. Az alapelv a hangyákon alapul. A program célja hogy adott pontok között megkeresse a legoptimálisabb utat -akár a hangyák- jelen esetben ez a leg-rövidebb utat. Ezt úgy valósítja meg hogy a mi "hangyáink" elindulnak egy cél felé. Megfigyelhetjük hogy egy idő után ha a többi hangya is arra a célra tart, akkor az útvonalak egyre rövidebbek lesznek míg végül megtalálják a legjobbat. Ha ez megvan, akkor onnantól kezdve minden hangya azt az utat fogja követni.



7.2. Java életjáték

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

A Conway-féle életjáték lényege egy szimulációs játék. A játék egy népesség növekedést és csökkenést szimulál leegyszerűsített formában. Habár játéknak hívják igazából nem igazán tekinthető annak mivel nincs szükség játékosra hozzá. Egyedüli szerepe talán annyi lehet hogy a kiindulási állapotot ő határozza meg de ezen felül nincs rá szükség. Valójában matematikai értelemben ez egy sejtautomata.

A játék három egyszerű szabály alapján működik. Vegyünk egy négyzetrácsos hálót, ez lesz a pálya. Ezen egy egy cellába egy egy sejt tehető bele. Egy sejtnak nyolc szomszédja lehet. A sejtekkel pedig három dolog történhet. Ha a sejtnak 2 vagy 3 szomszédja van akkor a sejt túléli. Ha 3-nál több van akkor túlnépesedés miatt, ha 2-nél kevesebb van akkor pedig elszigetelődés miatt a sejt meghal. Ha pedig egy cella szomszédságában pontosan 3 sejt van, akkor ott létrejön egy új sejt.

A sikkó alakzat lényege hogy folyamatosan önmagába megy át egy bizonyos ciklus alatt. Ezért ez az alakzat nem fog "elbúrjázni" se, de eltűnni se. Mozgása pedig az hogy átlósan felfelé kilő egy alakzatot. Ez a sikkó alakzat a hackerek hivatalos logója is.

```
public class Sejtautomata extends java.awt.Frame implements Runnable {
    public static final boolean ÉLŐ = true;
    public static final boolean HALOTT = false;
    protected boolean [][][] rácsek = new boolean [2][][];
    protected boolean [][] rác;
    protected int rácIndex = 0;
    protected int cellaSzélesség = 20;
    protected int cellaMagasság = 20;
    protected int szélesség = 20;
    protected int magasság = 10;
    protected int várakozás = 1000;
    private java.awt.Robot robot;
    private boolean pillanatfelvétel = false;
    private static int pillanatfelvételSzámláló = 0;

    public Sejtautomata(int szélesség, int magasság) {
        this.szélesség = szélesség;
        this.magasság = magasság;
        rácsek[0] = new boolean[magasság][szélesség];
        rácsek[1] = new boolean[magasság][szélesség];
        rácIndex = 0;
        rác = rácsek[rácIndex];
        for(int i=0; i<rác.length; ++i)
            for(int j=0; j<rác[0].length; ++j)
                rác[i][j] = HALOTT;
        sikkóKilövő(rác, 5, 60);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
            }
        });

        addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent e) {
```

```
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            cellaSzélesség *= 2;
            cellaMagasság *= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel = !pillanatfelvétel;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
            várakozás /= 2;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
            várakozás *= 2;
        repaint();
    }
});

addMouseListener(new java.awt.event.MouseAdapter() {
    public void mousePressed(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});

addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    public void mouseDragged(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});

cellaSzélesség = 10;
cellaMagasság = 10;
try {
    robot = new java.awt.Robot(
        java.awt.GraphicsEnvironment.
            getLocalGraphicsEnvironment().
            getDefaultScreenDevice());
} catch (java.awt.AWTException e) {
    e.printStackTrace();
}
```

```
setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség,
        magasság*cellaMagasság);
setVisible(true);
new Thread(this).start();
}

public void paint(java.awt.Graphics g) {
    boolean [][] rács = rácsok[rácsIndex];
    for(int i=0; i<rács.length; ++i) {
        for(int j=0; j<rács[0].length; ++j) {
            if(rács[i][j] == ÉLŐ)
                g.setColor(java.awt.Color.BLACK);
            else
                g.setColor(java.awt.Color.WHITE);
            g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                      cellaSzélesség, cellaMagasság);
            g.setColor(java.awt.Color.LIGHT_GRAY);
            g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                      cellaSzélesség, cellaMagasság);
        }
    }

    if(pillanatfelvétel) {
        pillanatfelvétel = false;
        pillanatfelvétel(robot.createScreenCapture
            (new java.awt.Rectangle
              (getLocation().x, getLocation().y,
               szélesség*cellaSzélesség,
               magasság*cellaMagasság)));
    }
}

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

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



```
        else if (s >= magasság)
            s = 0;

        if (rács[s][0] == állapot)
            ++állapotúSzomszéd;
        }

        return állapotúSzomszéd;
    }

    public void időFejlődés() {

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

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

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

                if(rácsElőtte[i][j] == ÉLŐ) {

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

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

        rácsIndex = (rácsIndex+1)%2;
    }

    public void run() {

        while(true) {
            try {
                Thread.sleep(várakozás);
            } catch (InterruptedException e) {}

            időFejlődés();
            repaint();
        }
    }
}
```

```
public void sikló(boolean [][] rács, int x, int y) {

    rács[y+ 0][x+ 2] = ÉLŐ;
    rács[y+ 1][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 1] = ÉLŐ;
    rács[y+ 2][x+ 2] = ÉLŐ;
    rács[y+ 2][x+ 3] = ÉLŐ;

}

public void siklóKilövő(boolean [][] rács, int x, int y) {

    rács[y+ 6][x+ 0] = ÉLŐ;
    rács[y+ 6][x+ 1] = ÉLŐ;
    rács[y+ 7][x+ 0] = ÉLŐ;
    rács[y+ 7][x+ 1] = ÉLŐ;

    rács[y+ 3][x+ 13] = ÉLŐ;

    rács[y+ 4][x+ 12] = ÉLŐ;
    rács[y+ 4][x+ 14] = ÉLŐ;

    rács[y+ 5][x+ 11] = ÉLŐ;
    rács[y+ 5][x+ 15] = ÉLŐ;
    rács[y+ 5][x+ 16] = ÉLŐ;
    rács[y+ 5][x+ 25] = ÉLŐ;

    rács[y+ 6][x+ 11] = ÉLŐ;
    rács[y+ 6][x+ 15] = ÉLŐ;
    rács[y+ 6][x+ 16] = ÉLŐ;
    rács[y+ 6][x+ 22] = ÉLŐ;
    rács[y+ 6][x+ 23] = ÉLŐ;
    rács[y+ 6][x+ 24] = ÉLŐ;
    rács[y+ 6][x+ 25] = ÉLŐ;

    rács[y+ 7][x+ 11] = ÉLŐ;
    rács[y+ 7][x+ 15] = ÉLŐ;
    rács[y+ 7][x+ 16] = ÉLŐ;
    rács[y+ 7][x+ 21] = ÉLŐ;
    rács[y+ 7][x+ 22] = ÉLŐ;
    rács[y+ 7][x+ 23] = ÉLŐ;
    rács[y+ 7][x+ 24] = ÉLŐ;

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

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

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

rács[y+ 11][x+ 25] = ÉLŐ;

}

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

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

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

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Conway/QT_elet

Ebben a feladatban szintén az életjáték megvalósítása a cél. Az előző felathoz hasonlóan itt is ugyanazon

az elven dolgozzuk ki a megoldást. Létrehozzuk a rácshálót melybe belehelyezzük az alap sejteket és elindítjuk a program futását. Ha ezzel megvagyunk akkor már nincs más dolgunk mint hogy megfigyeljük ahogyan a sikló-kilövő szépen sorban elkezd átlósan lefelé kilőni az alakzatokat.

7.4. BrainB Benchmark

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Conway/BrainB>

A program célja hogy egy személy koncentrációs képességét felmérje. Ezt úgy valósítja meg hogy a képernyőn több egyforma objektum van amik úgy néznek ki hogy egy négyzet benne egy körrel. Ezekből egyet kiválasztva az egeret a körre kell vinni és azon tartani. Ha ez sikerül akkor egyre több fog belőlük megjelenni és egyre gyorsabban fognak mozogni. A cél az hogy minél tovább tartsd az egeret az elsőnek kiválasztott ponton. Ha nem sikerül és elveszted akkor elkezdenek lassulni és csökken is az objektumok száma.

Ez egy kiváló tesztelése a koncentrációs képességnek. Például az esportolók körében is használható. Fel lehet vele mérni hogy egy játékos mennyire tud koncentrálni a saját karakterére egy tömeg esemény során.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó:

Megoldás forrása:

Ezt a feladatot az SMNIST for Human-nal helyettesítettem.

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Ezt a feladatot az SMNIST for Human-nal helyettesítettem.

8.3. Minecraft-MALMÖ

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Schwarzenegger>

Ebben a feladatban a Minecraftnak a MALMÖ projektével foglalkozunk. Ezt a projektet a Microsoft hozta létre és a mesterséges intelligencia kutatására használják. A MALMÖ célja hogy megtanítsa a programnak a Minecraft karakter irányítását és annak életben tartását. Ehhez a feladathoz pedig a python nyelvet használjuk.

Maga a project megtalálható a GitHub-on és onnan be is lehet szerezni. Megtalálható a project is és hozzá tutorialok is.

Most kezdjük el vizsgálni a kódot.

```

<ServerSection>
<ServerHandlers>
  <FlatWorldGenerator generatorString="3;7,220*1,5*3,2;3;,biome_1"/>
  <DrawingDecorator>
    <DrawSphere x="-27" y="70" z="0" radius="30" type="air"/>
  </DrawingDecorator>
  <ServerQuitFromTimeUp timeLimitMs="360000"/>
  <ServerQuitWhenAnyAgentFinishes/>
</ServerHandlers>
</ServerSection>

<AgentSection mode="Survival">
<Name>ZORZBot</Name>
<AgentStart/>
<AgentHandlers>
  <ObservationFromFullStats/>
  <ContinuousMovementCommands turnSpeedDegs="180"/>
</AgentHandlers>
</AgentSection>
</Mission>' ' '

```

A kód ezen része XML-ként van tárolva. Ebben a részben hozzuk létre magát világot amiben a karakter majd mozog és itt adjuk meg azt is hogy milyen módban induljon el a játék. Ezekhez az alábbi sorokat kell tekinteni. A világ létrehozása: `<FlatWorldGenerator generatorString="3;7,220*1,5*3,2;3;,biome_1"/>`. A játékmód kiválasztása: `<AgentSection mode="Survival">`. A kód többi része további beállításokat tartalmaz. Például azt hogy mennyi ideig tartson a küldetés.

Mielőtt magára a küldetés részére térnénk a kódnak még néhány változóról meg kell határozni hogy mi is. A `stevey`, `stevey` és `stevez` határozzák meg magának a karakternek a pozícióját. A `steveyaw` adja meg hogy a karakter milyen irányba néz és a `stevepitch` pedig magát a nézetet. Az `elotteidx`, `elotteidxj` és `elotteidxb` pedig arra hivatott hogy megvizsgáljuk hogy van-e valamilyen akadály a karakter előtt.

Miután ezeket a változókat létrehoztuk már nincs más hátra mint kezdőértéket adni nekik és utána kezdődhet is a számolás és a karakter mozgatása. A kezdőértékeket az alábbi módon állítjuk be.

```

if "Yaw" in observations:
    steveyaw = observations["Yaw"]
if "Pitch" in observations:
    stevepitch = observations["Pitch"]
if "XPos" in observations:
    stevex = observations["XPos"]
if "ZPos" in observations:
    stevez = observations["ZPos"]
if "YPos" in observations:
    stevey = observations["YPos"]

```

Miután az értékeket beállítottuk és a mozgás halad akkor már csak annyi van hogy a karakternek dönteni kell hogy ha akadály van akkor mit tegyen. Meg tud fordulni, ugrani tud vagy pedig támadni. Valamint az akadály változó értékét növeljük.

```
if nbr[elotteidx+9]!="air" or nbr[elotteidxj+9]!="air" or nbr[elotteidxb ←
    +9]!="air":
    print ("Nincs szabad utam, elottem: ", nbr[elotteidx+9])
    agent_host.sendCommand ("turn" + str(turn))
    akadaly = akadaly + 1
else:
    print ("Szabad az ut!")
    agent_host.sendCommand ("turn 0")
    agent_host.sendCommand ("jump 0")
    agent_host.sendCommand ("attack 0")
    akadaly = 0

if akadaly > 8:
    agent_host.sendCommand ("jump 1")

lepes = lepes + 1
if lepes > 100:
    lepes = 0
if tav < 20:
    prevste vex = ste vex
    prevste vez = ste vez
    prevstevey = stevey
    turn = turn * -1
    agent_host.sendCommand ("attack 1")
```

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Chaitin/Faktorialis>

A feladatban egy faktoriális kiszámoló kódot kellett létrehozni Lisp nyelven. A faktoriális érték egy olyan szám amihez meg kell adni egy természetes számot. A megadott számig pedig összeszorozzuk 1-től kezdődően a természetes számokat. Most pedig nézzük magát a kódot.

```
(princ "szam: ")
(setq a (read))
(setq b 1)
(loop for i from 1 to a
  do( setq b (* b i)))
(princ b)
```

Ez az iteratív megoldása a feladatnak. Elsőnek bekérünk számot a felhasználótól amin végrehajtuk a faktoriális. Ezután egy for ciklus segítségével végig csináljuk a szorzást és végül kiírjuk az eredményt a felhasználónak.

```
(defun faktorialis (n)
  (if (= n 1)
      1
      (* n (faktorialis (- n 1) ))
  ))
(princ "szam: ")
(setq a (read))
(princ (faktorialis a))
```

Ez pedig a rekurzív megoldás. Ebben a kódban elsőnek létrehozunk egy függvényt ami a rekurzív számolást hajtja végre. Ha az érték amit a felhasználó ad az 1 akkor az érték 1 lesz. Ez után kezdődik maga a rekurzív hívás. A lényege az hogy önmagát hívja meg a függvény ezáltal összeszorozva a számokat egészen egyig csökkentve. Amikor a függvény végez a szorzással akkor pedig kiírja az eredményt.

9.2. Gimp Scheme Script-fu: króm effekt

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

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Chaitin/Krom>

Ebben a feladatban az a célunk hogy készítsünk egy scriptet amit beillesztve a GIMP-be króm hatást ad a megadott szövegnek. Az ehhez tartozó kódot bemásolva a GIMP megfelelő mappájába a programban a létrehozásnál egy új menüpont jelenik meg ahol kiválaszthatjuk a beillesztett kódot.

Ha kiválasztjuk a Chrome3-Border2 menüpontot akkor megnyílik egy menü ahol további beállításokat lehet megadni. Ilyen például a kép mérete vagy a szegély vastagsága. De most lássuk a lépéseket mely során létrejön a króm felirat.

Első lépésben meghatározzuk a szöveg színét fehérnek majd hozzáadjuk a képhez és az eltolását is beállítjuk. Utána pedig egyesítjük a képet a szöveggel.

```
(gimp-context-set-foreground '(255 255 255)) ;szín beállítása
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS)))
(gimp-image-insert-layer image textfs 0 0) ;réteg hozzáadása a képhez
(gimp-layer-set-offsets textfs (* border-size 3) 0) ;eltolás beállítása
(set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM-LAYER))) ←
;
```

Ezt követően megadjuk a réteg intenzitását és az elmosást. Majd új színt adunk meg ami a fekete és utána invertáljuk a képet. Ezután felvesszünk egy új réteget amit hozzáadunk a képhez. Amint ez kész, következő lépésben megcsináljuk a színátmenetet. A színátmenet lineáris és nem tartalmaz ismétlést. Utolsó előtti lépésként pedig egyenetlen felületet hozunk létre. Végül pedig meghatározzuk a kép új méreteit és összeillesztjük az egész képet és megjelenítjük.

```
(plug-in-gauss-iir RUN-INTERACTIVE image layer 25 TRUE TRUE) ;elmosás ←
meghatározása
(gimp-drawable-levels layer HISTOGRAM-VALUE .18 .38 TRUE 1 0 1 TRUE) ; ←
réteg intenzitás meghatározása
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE) ;elmosás ←
meghatározása

(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0)) ;új szín ←
megadása
(gimp-selection-invert image) ;kép invertálása

(set! layer2 (car (gimp-layer-new image width (+ height (/ text-height 2)) ←
RGB-IMAGE "2" 100 LAYER-MODE-NORMAL-LEGACY))) ;második réteg ←
létrehozása
(gimp-image-insert-layer image layer2 0 0) ;réteg hozzáadása a képhez
```

```
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ↔
  GRADIENT-LINEAR 100 0 REPEAT-NONE ;színátmenet beállítása
  FALSE TRUE 5 .1 TRUE width 0 width (+ height (/ text-height 2)))

(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 0 ↔
  TRUE FALSE 2) ;egyenetlen felület létrehozása

(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve)) ;színgörbe ↔
  készítése

(gimp-image-scale image new-width (/ (* new-width (+ height (/ text-height ↔
  2))) width)) ;méret beállítása

(gimp-display-new image) ;kép megjelenítése
(gimp-image-clean-all image)
```

9.3. Gimp Scheme Script-fu: név mandala

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

Megoldás forrása: <https://github.com/nagyrobert1996/Prog1/tree/master/Chaitin/Mandala>

Az előző feladathoz hasonlóan itt is a GIMP-hez kell scriptet készíteni. Annyi a különbség hogy most nem króm effektet kell létrehozni hanem egy mandalát. A kódot ugyan úgy mint az előző esetben most is bemásoljuk a GIMP könyvtárba és utána ugyanúgy ott lesz a menüben a mandala is.

A mandala megnyitásakor egy nagyon hasonló almenü nyílik meg mint a króm esetében. Itt is meg lehet adni a szöveget valamint a kép méretet és egyéb dolgokat.

Az előző feladat során megtárgyalt kód csipeteket nem tárgyalom ki újból. Azokat lásd a króm effekt résznél. Ami újítás a ahhoz képest azokat fogom most leírni.

Az első új dolog ebben a scriptben a **gimp-layer-resize-to-image-size**. Ez arra szolgál hogy a szöveget újra méretezi úgy hogy illeszkedjen a kép méretéhez.

Másik újítás a **gimp-item-transform-rotate-simple**. Ennek célja a kódban a szöveg elforgatása adott szöggel.

Szintén új dolog még a **plug-in-autocrop-layer** is. Ez automatikusan körbevágja a szöveget a képen.

Az utolsó új dolog ebben a kódban a **gimp-image-select-ellipse**. Ennek az a feladata hogy a kijelölő tartalmát cserélje le a megadott értékek szerint. A megadott értékek pedig az x és y koordináták valamint a kijelölendő szélesség és magasság.

10. fejezet

Helló, Gutenberg!

10.1. Juhász István: Magas Szintű Programozási Nyelvek 1

1. fejezet

Ez a fejezet a programozás bevezetéséről szól. Abból indul ki a fejezet, hogy miért használunk programokat és hogy miért modellezzük csak a világot. A teljes világunk ugyanis túl összetett, teljességében nem tudjuk feldolgozni, ezért van szükség modellekre. Megismerkedünk a legalapvetőbb fogalmakkal. Például a programozási nyelvek különböző fajtáival és hogy ezek miből épülnek fel. Szó esik továbbá a programnyelvek osztályozásáról, hogy ezek lehetnek imperatív és deklaratívak. A fejezet végén pedig megtudjuk hogyan kell jegyzeteket használni és hogy a jegyzetelésből nem lehet megtanulni programozni.

2. fejezet

Ebben a fejezetben a program kódok alapvető építőelemeiről van szó. Megismerjük hogy egy programnyelv milyen karakterkészletből épül fel, mik azok a lexikális egységek és hogy azon belül milyen fajtái vannak. Megtudjuk továbbá hogyan kell felépíteni a kódot és abban miket lehet használni. Gondolok itt a változók típusaira, egyszerű és összetett típusra egyaránt. Szó van továbbá a mutatókról is és hogy egyes nyelvekben miként kell deklarálni a változókat.

3. fejezet

A fejezetben közelebbről megismerkedünk a C nyelv egy részével. Konkrétan az operátorokkal. Az operátorok olyan jelek melyek segítségével műveleteket lehet leírni a programnyelvben. Ilyen például az összeadás művelete vagy akár a kisebb művelet. Ezen felül kapunk egy precedencia táblázatot is a fejezetben. Így tehát azt is megtudjuk hogy a műveletek között melyik az erősebb, vagyis melyik értékelődik ki hamarabb.

10.2. Kernighan-Ritchie: A C programozási nyelv

1. fejezet

Az első fejezetben megismerjük a C programozási nyelv legalapvetőbb építőelemeit és példákon keresztül megismerjük ezek egyszerű használatát. Felhívnam a figyelmet arra hogy bár alapvető dolgokról van szó de rendkívül jelentősek és nem elhanyagolhatók. Hiszen ezen egyszerű dolgok az alap építőkövei a C programnyelvnek. A fejezetben szó volt a legalapvetőbb dologról, úgy mint képernyőre íratás, változók

típusai, for ciklusok, tömbök. Valamint szó esett arról is hogyan kell szimbolikus állandókat létrehozni a **#define** segítségével és hogyan lehet létrehozni egyszerű függvényeket és azokra hivatkozni. A fejezet végén pedig megnéztük mit jelent az érték szerinti hívás, a tömbök egy speciális fajtáját, a karaktertömböt is átvizsgáltuk és a külső változókról is meg tudtuk hogy mit jelentenek.

2. fejezet

Ebben a fejezetben nagy hangsúlyt fektet a könyv a változókra, a velük végezhető műveletekre, a logikai utasításokra és a fejezet végén az elágazásról is kapunk információt. A fejezet elején megismerjük hogy vannak kulcsszavak melyek nem lehetnek változó nevek valamint hogy egy jó név hogyan épül fel. Azt is meg tudjuk hogy az egyes változók mekkora méreten vannak tárolva, valamint hogy egy állandót hogyan kell létrehozni amit később aztán nem lehet megváltoztatni. Szó esik továbbá arról is hogyan kell létrehozni változókat kezdőérték nélkül, vagy akár kezdőértékkel is. Ezekkel milyen műveletek végezhetőek, úgy mint két int változó összeadása vagy akár szorzása. De azt is megtudjuk hogyan lehet őket összehasonlítani hogy melyik a kisebb vagy egyenlő-e. Két változóval ha valamilyen műveletet akarunk végrehajtani akkor azoknak egy fatájúnak kell lenniük. Ehhez van hogy át kell alakítani őket. Ennek mikéntjére is kapunk segítséget a könyvből. Továbbá megismerjük az inkrementáló, dekrementáló és bitenkénti logikai operátorokat is. A fejezet végén pedig van egy táblázat ami magába foglalja az operátorokat és azok kiértékelési sorrendjét. Végül pedig szó volt a fejezetben az elágazásokról is.

3. fejezet

Ez a fejezet a vezérlési szerkezetekről szól. Megismerjük mit jelent egy utasítás és hogy mi az egy blokk. Ezen kívül részletesebben megismerjük az elágazás mibenlétét. Meg tudjuk hogy az elágazásokat lehetőségünk van egymásba ágyazni vagy akár többszörösen összetett elágazásokat létrehozni. A többszörösen összetett elágazásnak két módja van. Az egyik az else-if, a másik a switch. A két féle elágazás nem egyforma annak ellenére mindegyik arra szolgál hogy föbb felé ágazzunk el. Szó van továbbá a ciklusokról és azok fajtájáról. Megnézzük hogyan épülnek fel a while, a do-while és for ciklusok. Melyiket milyen esetben érdemes használni és hogy miként kell őket kódban felépíteni. A fejezet végén pedig szó van három utasításról. Név szerint ezek a break, a continue és a goto. Ezek arra szolgálnak hogy a ciklusokban apró vagy akár nagyobb ugrásokat tehessünk meg, ha szükséges.

10.3. Levendovszky-Benedek: Szoftverfejlesztés C++ nyelven

1. fejezet

A könyv bevezetése.

2. fejezet

A fejezet célja hogy bemutassa hogy a C++ milyen újításokat hozott C programnyelvhez ezáltal új nyelvet létrehozva. Egyes újítások kényelmi szerepet töltenek be, míg más újítások új lehetőségeket tártak fel. Egyik ilyen újítás hogy a függvényeknél C-ben az üres paraméter lista az tetszőleges paramétert jelent míg C++-ban ez üres paraméter listát jelent. Másik újítás a main függvénynél van. Lehetőség van megadni két paramétert neki amely a parancssori argumentumokban játszik szerepet. Továbbá a C++-ban bevezetésre került a bool típusú változó amely az igaz, hamis eldöntésekben nyújt kényelmi funkciót. További újítás hogy ebben a nyelvben már több helyen van lehetőség változót létrehozni, valamint hogy egy függvénynevet többször is fel lehet használni ha a paraméterlistájuk különbözik. Ez azért nagyon jó mert így nem kell erőltetett neveket kitalálni egy-egy függvénynek. Ezen felül újabb lehetőség hogy függvény deklarációjánál módunkban áll a paramétereknek alapértelmezett értéket megadni, így akár azokkal is dolgozhatunk. A

fejezet végén pedig szó esik arról hogy mit jelent a referencia szerinti átadás és hogy ennek milyen előnyei és hasznai vannak.

3. fejezet

A fejezet témája az objektum orientáltság és annak részei. Elsőnek megemlíti hogy miért van szükség és miért hasznos az ilyen fajta programozás. Ez után szó van a struktúrákról és azok szerepéről. Kitér arra hogy mit jelent maga a struktúra, azon belül is a tagváltozó. Továbbá hogy a tagfüggvények mit jelentenek és azoknak miként lehet hatáskört beállítani. A fejezet beszél továbbá az adatrejtésről is. Ez azt jelenti hogy be lehet állítani hogy a program egyes részei **public** vagy **private** beállítást kapjanak. Erre azért lehet szükség mert így meg lehet adni hogy milyen szinten lehet az adott részt felhasználni. Bemutatja továbbá a konstruktort és a destruktort is. Ezeknek jelentős szerepe van az osztályokban. A konstruktor feladata hogy inicializálja a változókat ezáltal egy kezdőértéket állítva a változóknak. A destruktorkor épp az ellentéte. Az ő feladata hogy amikor már nincs szükség a változóra akkor felszabadítja a helyüket. A dinamikus adattagokat tartalmazó osztályokról is szó van a fejezetben. A dinamikus adattagokat úgy lehet létrehozni hogy használjuk a **new** kulcsszót. Ezzel nem csinálunk más csak helyet foglalunk az adattagnak a memóriában. Ha pedig már nem lesz rá szükség akkor a helyet a **delete** kulcsszóval lehet felszabadítani. Szó esik még ezen adattagok támogatásáról és a másolókonstruktorokról is. A másolókonstruktorok szerepe hogy lehetőséget biztosítanak egy ugyan olyan felépítésű adatszerkezet lemásolására mint az eredeti. A friend függvények és osztályok is szerepelnek a fejezetben. Ezek olyan függvények és osztályok amelyek alaptól nem az általunk használt osztálynak a részei ám még is lehetőségünk van hozzáférni a másik osztály védett részeihez azáltal hogy használjuk a **friend** kulcsszót. A fejezet végéhez közeledve megismerjük még azt is hogy mi a különbség az inicializálás és az értékadás között, valamint a statikus tagok szerepéről is megtudunk néhány dolgot. Legvégül pedig arra is kitér ez a rész hogy az egymásba ágyazott definíciók hogyan néznek ki. Gondolok itt olyanra mint például az osztályon belüli osztály.

4. fejezet

Ebben a fejezetben a fő téma a konstansok. A programozásban sokféle konstans lehet használni és itt ezeket vesszük sorra. Elsőnek a konstans változókról értekezik a fejezet. Leírja azok szerepét és hogy hogyan lehet őket használni. Ez után a pointerek körében lévő konstansok szerepét írja le. Megtudjuk hogy kétféle konstans pointer van, az egyik mikor az érték konstans, a másik mikor maga a pointer konstans. Továbbá vannak még konstans függvényparaméterek, tagváltozók, tagfüggvények. Ezeknek mind más-más szerepük van de mindegyik lényege az hogy úgy mondva állandók tehát nem lehet őket megváltoztatni. Van azonban egy kivételes eset is amit megemlít a fejezet. Ez a mutable. Ezt egy konstans tagfüggvényen belül használjuk ha abban meg akarjuk változtatni egy változó értékét, ám azt akarjuk hogy maga a függvény mégis konstansként legyen értelmezve. Ekkor használjuk a **mutable** kulcsszót. A fejezet vége felé pedig szó van a konstans és a nem konstans közötti konverzióról is. Nem konstansot lehetőség van konstanssá tenni fordítva viszont ez nem lehetséges. Legvégül pedig szóba kerülnek az inline függvények és azok szerepe is.

III. rész

Második felvonás

**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.

11. fejezet

Helló, Arroway!

11.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.!

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf(16-22 fólia)

Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: `source/labor/polargen`)

Megoldás forrása:

A feladat a módosított polártranszformációs normális generátor létrehozása volt Java nyelven. A módosított polártranszformációs normális generátor legenerál két pseudorandom számot melyből az egyiket kiírja a standard outputra a másikat pedig eltárolja a **double tarolt** változóban.

A pseudorandom számok olyan számok melyek nagyon hasonlítanak a véletlen számokra, ám mégsem teljesen azok. Hiszen ezek a számok egy matematikai képlet segítségével vannak generálva egy sorozat részletéből, így ez visszafejthető.

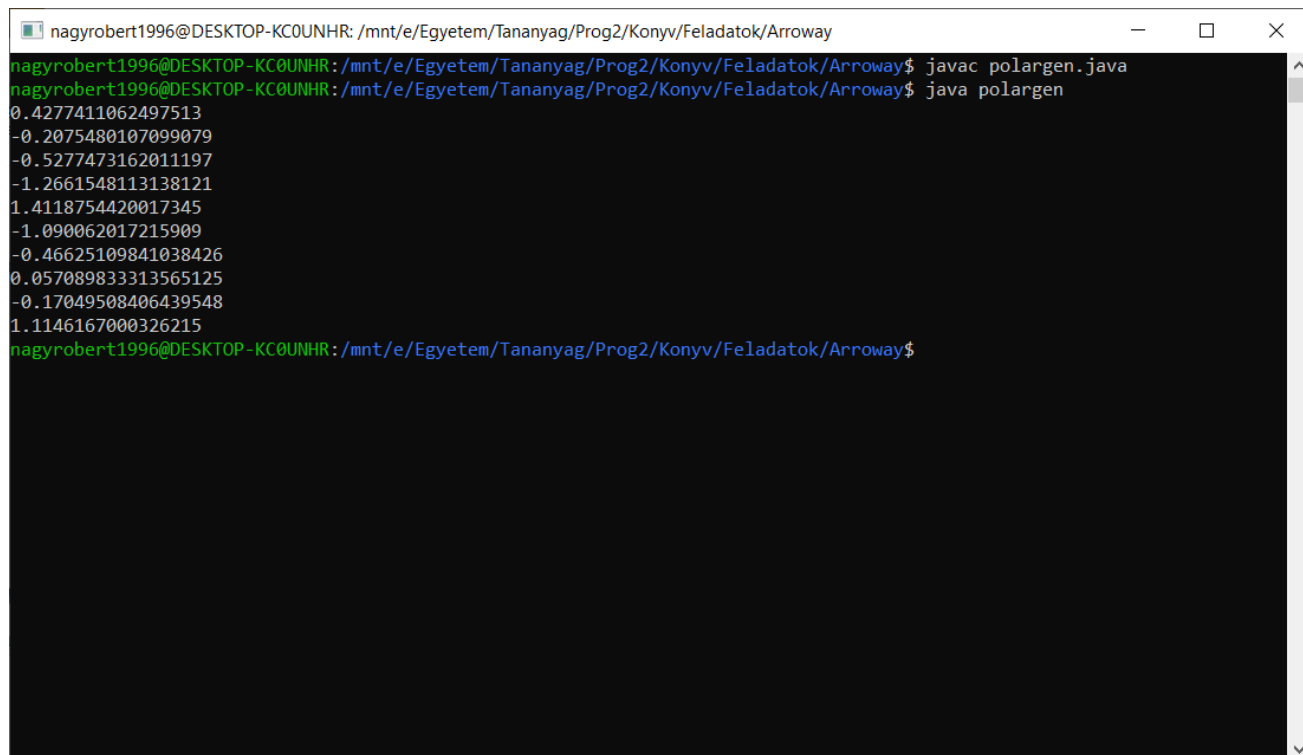
A generátor pedig ilyen számokból állít elő kettőt melyből az egyiket eltárolja. A program futása során 10-szer hívódik meg a **kovetkezo** nevű függvény, amely elvégzi a generálást ha nincs tárolt szám és kiírja. Ha van tárolt szám akkor a generálás nem történik meg csak kiírja a már eltárolt számot. Ez a megoldás azért jó mert így a kód lefuttatása során, bár 10 szám lesz kiírva, a generálás csak 5-ször történik meg, ezzel is gyorsítva a folyamatot. Hiszen a processzornak nem kell minden alkalommal végig menni a generálás lépésein.

```
public class polargen {  
  
    boolean nincsTarolt = true;  
    double tarolt;  
  
    public polargen() {  
        nincsTarolt = true;  
    }  
}
```



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

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



```
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ javac polargen.java
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ java polargen
0.4277411062497513
-0.2075480107099079
-0.5277473162011197
-1.2661548113138121
1.4118754420017345
-1.090062017215909
-0.46625109841038426
0.057089833313565125
-0.17049508406439548
1.1146167000326215
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$
```

Most pedig lássuk mi a különbség a JDK-ban lévő generátor és a mi generátorunk között. A JDK-ban lévő generátor nagyrészt ugyan úgy működik mint a mi módosított polártranszformációs normális generátorunk annyi különbséggel, hogy abban a megoldásban gondoltak a párhuzamosításra is. Ezt bizonyítja az is, hogy a classban lévő metódus létrehozásánál a **synchronized** kulcsszó szerepel. Ennek feladata, hogy felügyelje a párhuzamos futásokat. Ezt úgy valósítja meg, hogy amíg az egyik szálon fut a generálás, addig a többi szál számára nem teszi lehetővé a generáláshoz való kód hozzáférését.

Egy másik megoldás ami szintén része a JDK-nak hogy a **StrictMath** osztályt használja. Eerre azért van szükség hogy bármilyen környezetben vanimplementálva a kód azeredmény mindig ugyan az legyen. Hiszen ha csak a **Math** osztályt használnánk akkor az eredmény változhatna a környezettől függően.

```
private double nextNextGaussian;
private boolean haveNextNextGaussian = false;

synchronized public double nextGaussian() {
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1;    // between -1.0 and 1.0
            v2 = 2 * nextDouble() - 1;    // between -1.0 and 1.0
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}
```

```
}  
}
```

11.2. Homokozó

Írjuk át az első védési programot (LZW binfa) C++ nyelvről Java nyelvre, ugyanúgy működjön! Mutasunk rá, hogy gyakorlatilag a pointereket és referenciákat kell kiirtani 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!

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.3. "Gagyi"

Az ismert formális „while ($x \leq t \ \&\& \ x \geq t \ \&\& \ t \neq x$);” 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!

Megoldás forrása:

Ebben a feladatban egy egyszerű példán keresztül megmutatásra kerül hogy a Javában az Integer osztályban a gyorsítótárban előre le vannak generálva a számok -128 és 127 között. Ez abból látszik, hogy mind a két Integer típusú változó értéke 6 ám a program még sem lép be a végtelen ciklusba.

```
private static class IntegerCache {  
    static final int low = -128;  
    static final int high;  
    static final Integer cache[];  
  
    static {  
        // high value may be configured by property  
        int h = 127;  
        String integerCacheHighPropValue =  
            sun.misc.VM.getSavedProperty("java.lang.Integer. ↵  
                IntegerCache.high");  
        if (integerCacheHighPropValue != null) {  
            try {  
                int i = parseInt(integerCacheHighPropValue);  
                i = Math.max(i, 127);  
                // Maximum array size is Integer.MAX_VALUE  
                h = Math.min(i, Integer.MAX_VALUE - (-low) - 1);  
            } catch (NumberFormatException nfe) {
```

```
        // If the property cannot be parsed into an int, ↵
        ignore it.
    }
}
high = h;

cache = new Integer[(high - low) + 1];
int j = low;
for(int k = 0; k < cache.length; k++)
    cache[k] = new Integer(j++);

// range [-128, 127] must be interned (JLS7 5.1.7)
assert IntegerCache.high >= 127;
}

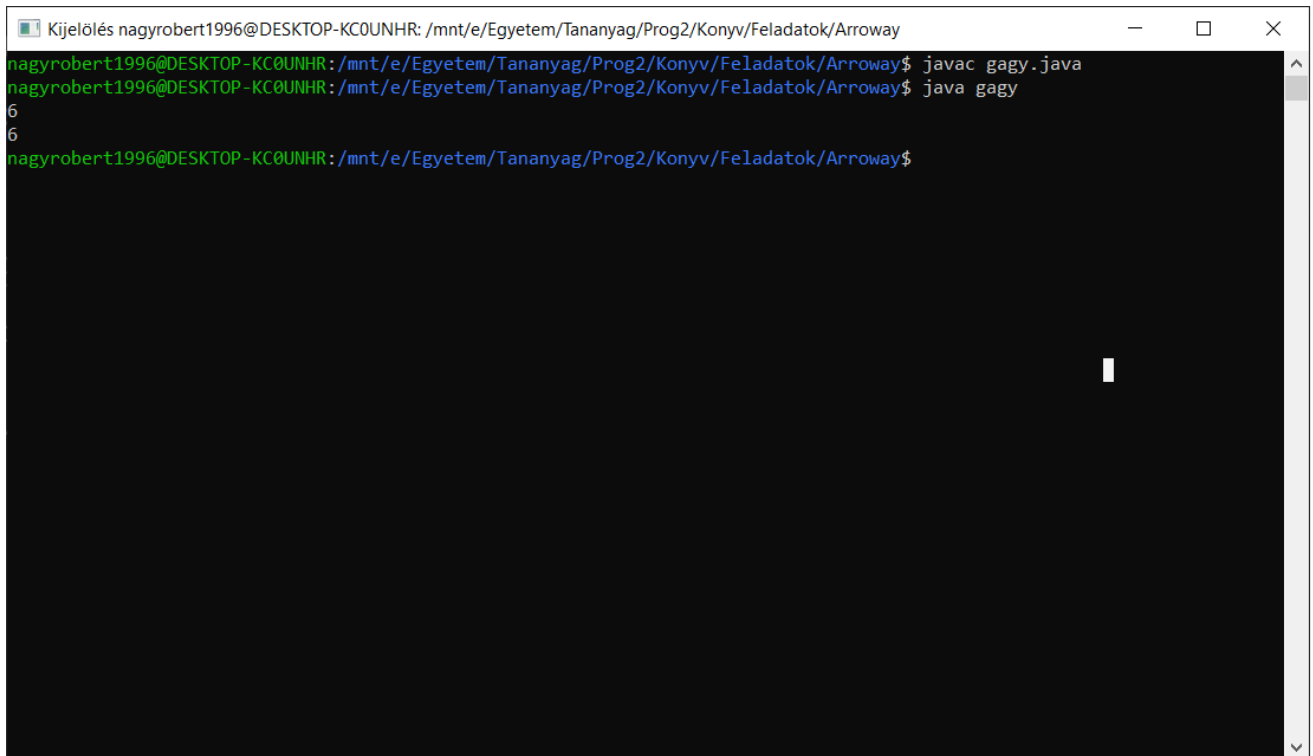
private IntegerCache() {}
}
```

Ez azért van mert a ciklus feltétel vizsgálatakor, az Integer osztály esetében a memóriacím vizsgálata történik. Ebből kifolyólag pedig mivel a 6-os előre le van generálva így a két példány ugyanarra a memóriacímre címez és így az utolsó feltétel nem teljesül, tehát nem fog belépni a ciklusba.

```
public class gagy {

    public static void main(String[] args) {
        Integer x = 6;
        Integer t = 6;
        System.out.println(x);
        System.out.println(t);
        while (x <= t && x >= t && t != x) ;

    }
}
```



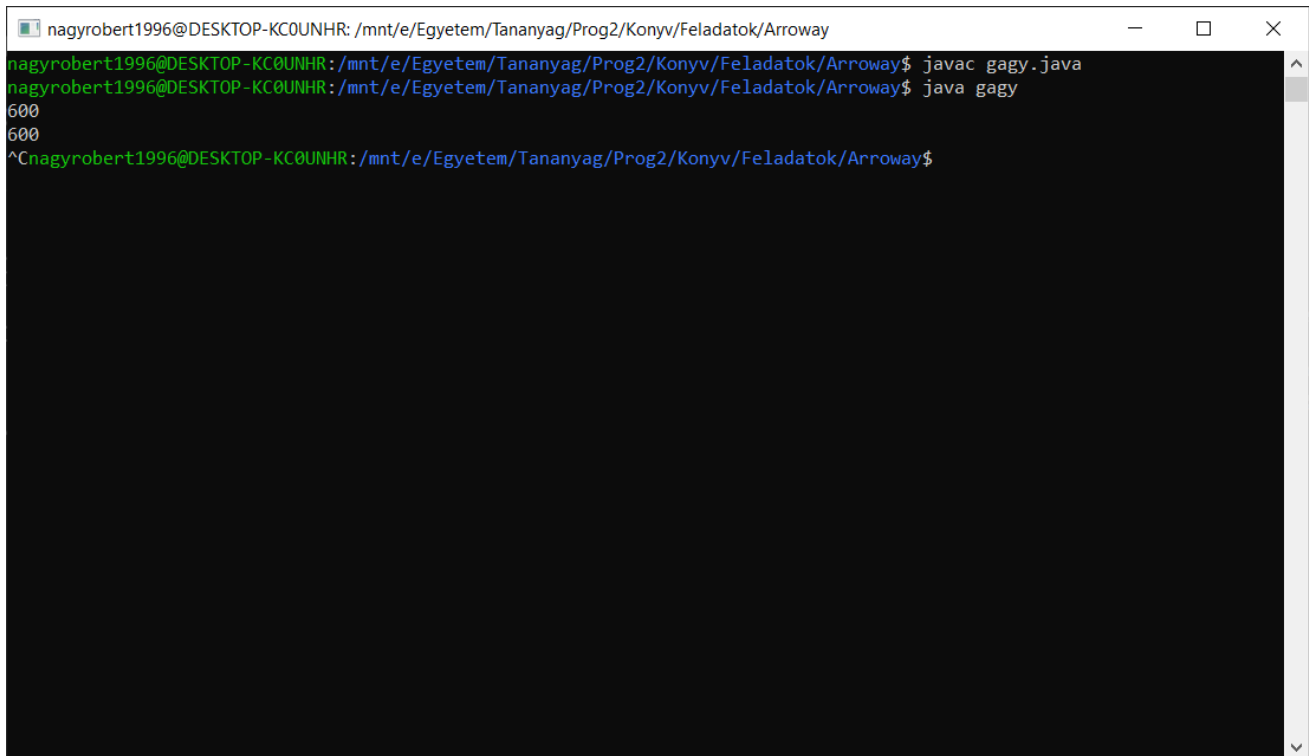
```
Kijelölés nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ javac gagy.java
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ java gagy
6
6
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$
```

Most pedig lássuk mi történik ha olyan számokat választunk amelyek már nincsenek előre legenerálva. Ha például a két értéket átírjuk 600-ra akkor már az összes feltétel teljesülni fog így belép a végtelen ciklusba. Ez azért történik meg mert két új példányt hoz létre a program amiknek a memóriacíme már nem egyezik meg és így az utolsó feltétel is teljesül.

```
public class gagy {

    public static void main(String[] args) {
        Integer x = 600;
        Integer t = 600;
        System.out.println(x);
        System.out.println(t);
        while (x <= t && x >= t && t != x) ;

    }
}
```

A screenshot of a terminal window with a black background and green text. The window title is 'nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway'. The terminal shows the following commands and output:

```
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ javac gagy.java
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ java gagy
600
600
^Cnagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$
```

11.4. Yoda

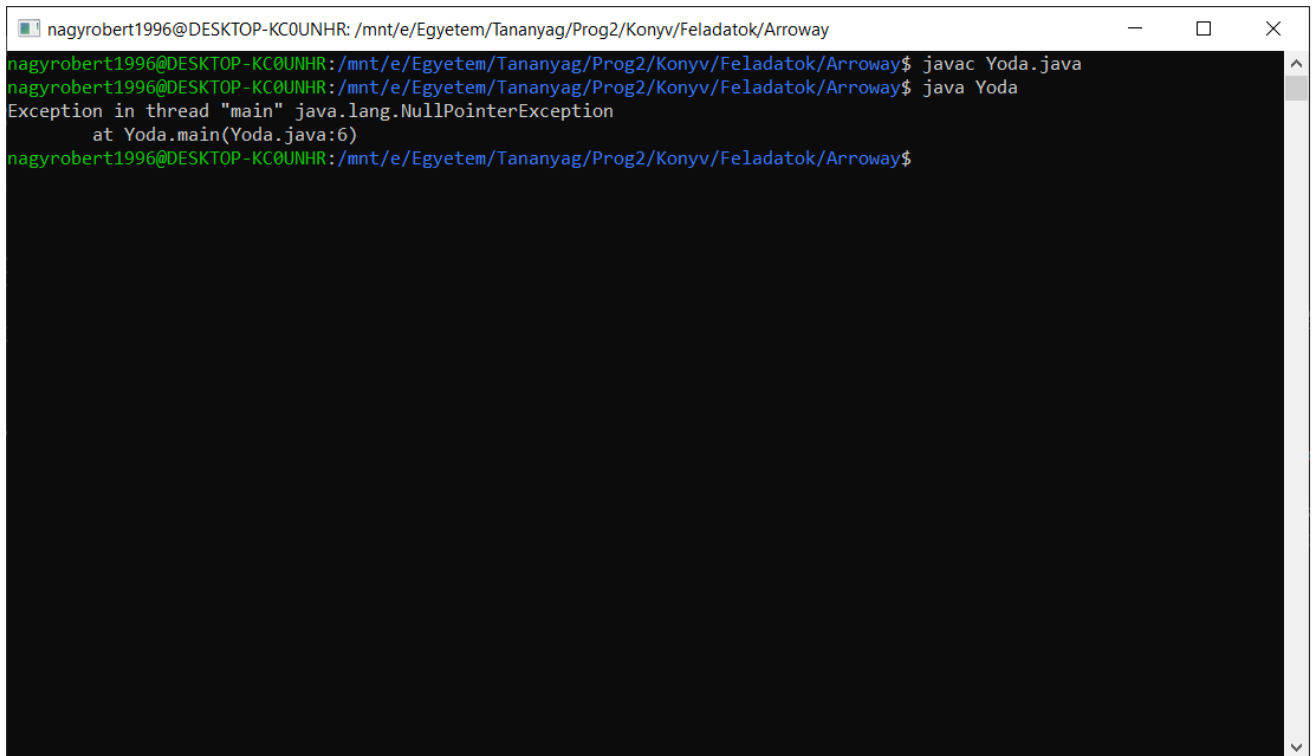
Írjunk olyan Java programot, ami `java.lang.NullPointerException`-el leáll, ha nem követjük a Yoda conditions-t!
https://en.wikipedia.org/wiki/Yoda_conditions

Megoldás forrása:

A feladat lényege hogy megmutassuk mi a Yoda conditions. Ezt nagyon könnyű megmutatni egy egyszerű példával. Kell hozzá egy null értékű változó és egy feltétel vizsgálat. A lényeg az, hogy ha egy null pointerre hívunk meg feltétel vizsgálatot akkor az `java.lang.NullPointerException`-el leáll, tehát hibát kapunk eredményül. Viszont ha a Yoda conditions-t követjük és felcseréljük a sorrendet akkor már lefut a program hiba nélkül. Az mindegy hogy egy konstansra vagy literálra hívjuk meg a vizsgálatot a lényeg az hogy ne null pointer legyen. Az eredmény amit vissza ad az `false` lesz, viszont nem történik hiba mivel nem a null pointerre lett a feltétel vizsgálat meghívva hanem csak paraméterként lett átadva a módszernek.

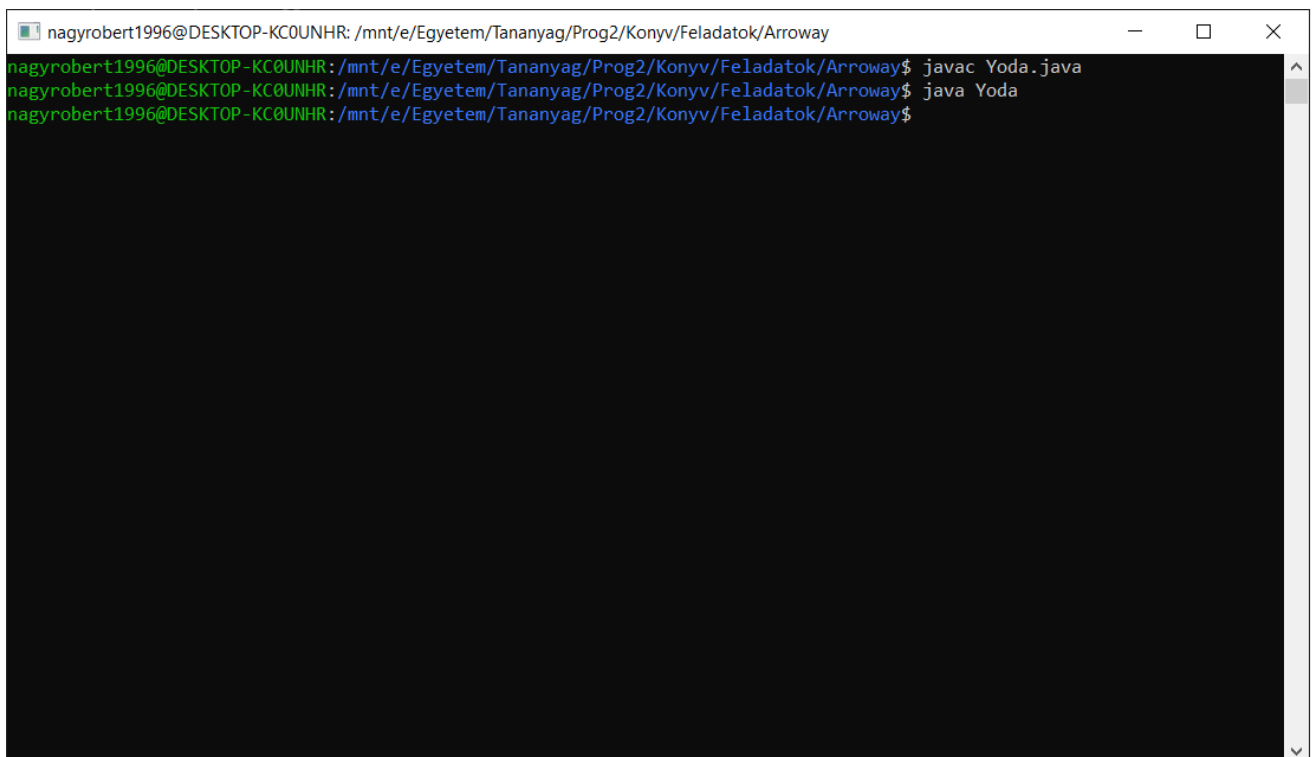
```
public class Yoda
{
    public static void main (String[]args)
    {
        String myString = null;
        if (myString.equals("proba"))
        {
            System.out.println("egyenlő a probával");
        }
        /*if ("proba".equals(myString))
        {
            System.out.println("egyenlő a null értékkel");
        }
    }
}
```

```
        } * /  
    }  
}
```



```
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway  
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ javac Yoda.java  
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ java Yoda  
Exception in thread "main" java.lang.NullPointerException  
    at Yoda.main(Yoda.java:6)  
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$
```

Ez történik ha a null pointerre van meghívva a vizsgálat.



```
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway  
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ javac Yoda.java  
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ java Yoda  
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$
```

Itt pedig azt látjuk amikor csak paraméterként van átadva. Hiba nem történt csak a vizsgálat eredménye false lett.

11.5. Kódolás from scratch

Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbpalg.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> (mert ha csak lemásolod, akkor pont az a fejlesztői élmény marad ki, melyet szeretném, ha átélnél).

Megoldás forrása:

A BBP algoritmus arra szolgál, hogy a Pi értékét határozzuk meg hexadecimális formában a $d+1$. helytől. Mindezt úgy teszi hogy nincs tudomása a korábbi jegyekről.

A program PiBBP nevű függvénye arra szolgál hogy kiszámítsuk a $\{16^d \text{ Pi}\} = \{4 * \{16^d S1\} - 2 * \{16^d S4\} - \{16^d S5\} - \{16^d S6\}\}$ képletet, ahol is a $\{\}$ a törtrészt jelöli. Továbbá ebben a függvényben történik meg a hexadecimális számrendszerbe történő átváltás és az eredmény belehelyezése a bufferbe is. Az átváltás és a bufferbe helyezés mind a while ciklusban történnek meg.

```
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();
}
```

A következő metódus a d16Sj. Ez egy segéd metódus ami arra szolgál hogy a fenti képletben lévő S1, S4, S5 és S6 értéket kiszámítsa.

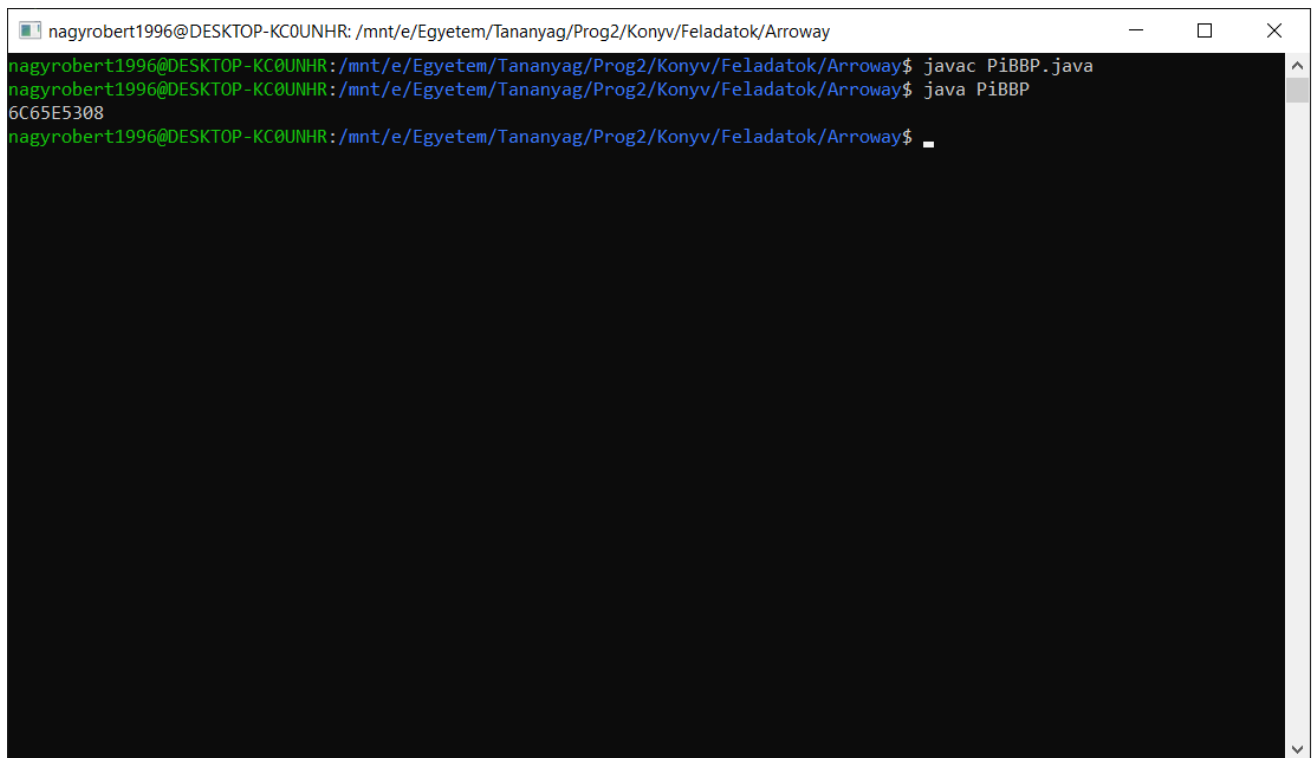

```
public double d16Sj(int d, int j) {  
  
    double d16Sj = 0.0d;  
  
    for(int k=0; k<=d; ++k)  
        d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);  
  
    return d16Sj - StrictMath.floor(d16Sj);  
}
```

Az utolsó metódus a main-en kívül amiről szót kell ejteni az n16modk. Ez a metódus bináris hatványozást végez azért hogy a S értékeket ki lehessen számítani.

```
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;  
}
```

Végezetül pedig a main függvényben történik egy PiBBP objektum létrehozás és a d érték meghatározása, ami jelen helyzetben 1000000. Így tehát a Pi értékei az 1000001. helytől kezdődően lesznek kiszámítva.

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



```
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ javac PiBBP.java
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$ java PiBBP
6C65E5308
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Arroway$
```

12. fejezet

Helló, Liskov!

12.1. Liskov helyettesítés sértése

Írjunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés.

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/BatfaiBarki/madarak/)

Megoldás forrása:

A Liskov elv lényege az objektumorientált programok helyes tervezése. A könnyű érthetőség kedvéért egy példán keresztül nézzük meg ezt. Vegyük például azt hogy van egy madár osztályunk. Ebben létrehozunk egy tulajdonságot ami kimondja hogy a madarak tudnak repülni. Ez után ha származtatunk két osztályt a madár osztályból, legyen egy sas és egy pingvin osztály, akkor az megsérti a liskov elvet. A elv szerint ugyanis rossz a tervezés. Ez abból derül ki hogy mivel származtatott osztályról van szó a pingvin osztály megkapja a reülés tulajdonságot. Ez viszont hiba, mivel a pingvin nem tud repülni. Ha nem akarjuk megsérteni az elvet akkor a megoldás az hogy újra kell tervezni a programot. Ezt például úgy lehet megtenni ha a repülés tulajdonságát elkülönítjük egy repülő madarak osztályba és ez után a sas a repülő madarak osztályába fog tartozni a pingvin pedig csak a madarak osztályába. Így a tervezés már jó lesz és nem lesz megsérve a Liskov elv.

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

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

class Sas extends Madar
{};

class Pingvin extends Madar
{};
```

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

        Sas sass = new Sas();
        kod.fgv ( sass );

        Pingvin pingvin = new Pingvin();
        kod.fgv ( pingvin );

    }
}
```

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

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

class Sas : public Madar
{};

class Pingvin : public Madar
{};

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

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin );

}
```

12.2. Szülő-gyerek

Írjunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ősön keresztül csak az ős üzenetei küldhetők!

https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)

Megoldás forrása:

A feladat megoldása során szükség van két osztályra a kódban ahhoz hogy bemutassuk a feladatot. Az egyik osztály lesz az ős osztály, ennek a neve **Szulo**. A másik osztály egy származtatott osztály amely a **Gyerek** nevet kapta.

Az ős osztály lényege hogy egy általánosabb egységet hozzunk létre, amely egy nagyobb egységbe zárja a kódban létrehozott példányokat. Így a benne szereplő tulajdonságok egy tágabb tartományt hoznak létre. Ilyen például az, ha egy programban le akarjuk modellezni az emlősöket. Az emlős lesz az ős osztály amely egy tágabb tartomány és minden benne szereplő tulajdonság igaz lesz az emlősökre.

A származtatott osztály egy szűkebb tartomány amely már több megkötést tartalmaz viszont rendelkezik az ős osztály tulajdonságaival is. Erre példa ha mondjuk származtatott osztályként létrehozunk ember osztályt vagy delfin osztályt. Mindkettő több megkötést tartalmaz mint az emlős, de akár az ember akár a delfint nézzük mind a kettő rendelkezik az emlős osztály tulajdonságaival is.

A feladat megoldása során egy egyszerűbb példa kerül bemutatásra. Lesz amár említett Szulo és Gyerek osztály. Az ős osztály a szülő lesz, a származtatott osztály pedig a gyerek. A gyerek osztály tudja használni a szülő osztály metódusát is, viszont a szülő nem tudja a gyereket.

Most pedig lássuk a kódokat. Először is a Java nyelven megírtat.

```
class Szulo
{
    public void szulo_vagyok()
    {
        System.out.println("Én vagyok a szülő.");
    }
};

class Gyerek extends Szulo
{
    public void gyerek_vagyok()
    {
        System.out.println("Én vagyok a gyerek.");
    }

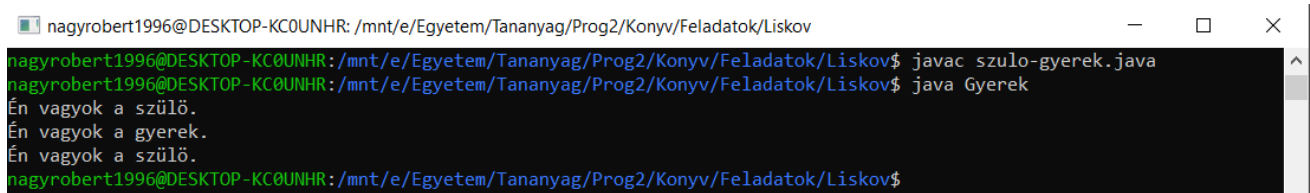
    public static void main(String[] args)
    {
        Szulo apa = new Szulo();
        Gyerek joska = new Gyerek();
        apa.szulo_vagyok();
        joska.gyerek_vagyok();
        joska.szulo_vagyok();
    }
}
```

```

        apa.gyerek_vagyok();    //ez így nem működik mivel az ősz ←
                                osztály nem fér hozzá a származtatott osztályhoz ezáltal ←
                                a benne lévő metódusokhoz sem
    }
};

```

Létrehozásra kerül a két osztály, majd a mainben mindegyik osztályból egy-egy példány. Láthatjuk hogy a kódban jelölve van a származtatás. A Gyerek osztály létrehozásánál ott van az **extends** kulcsszó. Ez arra szolgál hogy mögötte feltüntessünk egy másik osztályt amely által az meg lesz jelölve őznek. Az által pedig hogy van ősz osztálya ő egy származtatottá válik. Végül személtetés céljából a példányokra meghívásra kerül a szülő osztály metódusa is valamint a gyerek osztály metódusa is. A gyerek osztály tudja használni a szülő osztály függvényét, míg a szülő nem tudja használni a gyerek osztályét.

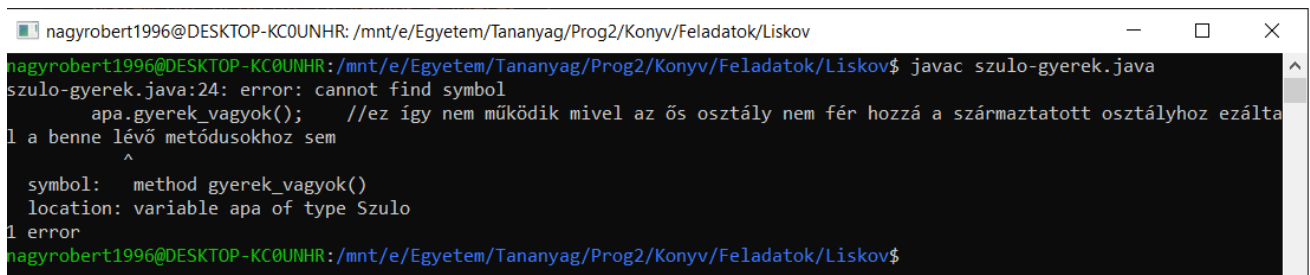


```

nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ javac szulo-gyerek.java
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ java Gyerek
Én vagyok a szülő.
Én vagyok a gyerek.
Én vagyok a szülő.
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$

```

Ha mégis megpróbáljuk az ősz osztálynak a példányára meghívni a származtatott osztály függvényét akkor az hibát fog eredményezni.



```

nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ javac szulo-gyerek.java
szulo-gyerek.java:24: error: cannot find symbol
    apa.gyerek_vagyok();    //ez így nem működik mivel az ősz osztály nem fér hozzá a származtatott osztályhoz ezáltal
    ^
1 a benne lévő metódusokhoz sem
symbol:   method gyerek_vagyok()
location: variable apa of type Szulo
1 error
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$

```

Most pedig következzen a C++-os megoldás. Látható hogy nagy különbség nincs a Javához képest. Ami változik az annyi, hogy másként van megjelölve a származtatás. A futás során is ugyanarra az eredményre jutunk.

```

#include <iostream>

class Szulo
{
public:
    void szulo_vagyok()
    {
        std::cout << "Én vagyok a szülő." << std::endl;
    }
};

class Gyerek : public Szulo
{
public:
    void gyerek_vagyok()

```

```

        {
            std::cout << "Én vagyok a gyerek." << std::endl;
        }
    };

    int main ( int argc, char **argv )
    {
        Szulo szulo;
        Gyerek gyerek;
        szulo.szulo_vagyok();
        gyerek.gyerek_vagyok();
        gyerek.szulo_vagyok();
        //szulo.gyerek_vagyok();    ez így nem működik mivel az ő ←
        osztály nem fér hozzá a származtatott osztályhoz ezáltal a ←
        benne lévő metódusokhoz sem

    }

```

12.3. Anti OO

A BBP algoritmussal a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10^6 , 10^7 , 10^8 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/-hu/tartalom/tkt/javat-tanitok-javat/apas03.html#id561066>

Megoldás forrása:

Jegy	Java	C	C++	C#
10^6	1,256	1,390625		1,287183
10^7	14,548	16,26563		14,84537
10^8	166,828	187,2031		171,0567

A táblázat alapján látható hogy a futási idő eredményei változnak attól függően hogy milyen program nyelvet használunk. A leggyorsabb a Java nyelven megírt kód volt, bár ez nem meglepő hiszen az beépített optimalizálóval rendelkezik a kódok gyorsítása érdekében. Leglassabb eredményt pedig a C nyelv hozott ahol a 10^8 -on eredmény kiszámításához több mint 3 perc kellett.

```
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ javac PiBBPBench.java
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ java PiBBPBench
6
1.256
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ javac PiBBPBench.java
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ java PiBBPBench
7
14.548
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ javac PiBBPBench.java
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ java PiBBPBench
12
166.828
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$

nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ gcc pi_bbp_bench.c -lm -o pi
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ ./pi
6
1.390625
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ gcc pi_bbp_bench.c -lm -o pi
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ ./pi
7
16.265625
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ gcc pi_bbp_bench.c -lm -o pi
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ ./pi
12
187.203125
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$

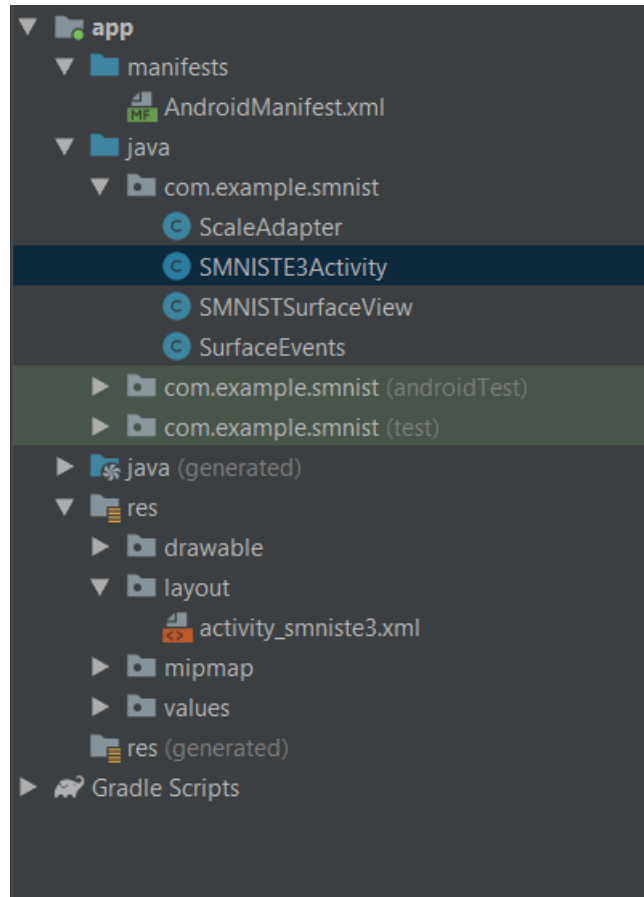
nagyrobert1996@DESKTOP-KC0UNHR: /mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ mcs PiBBPBench.cs
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ mono PiBBPBench.exe
6
1.287183
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ mcs PiBBPBench.cs
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ mono PiBBPBench.exe
7
14.845366
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ mcs PiBBPBench.cs
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$ mono PiBBPBench.exe
12
171.056654
nagyrobert1996@DESKTOP-KC0UNHR:/mnt/e/Egyetem/Tananyag/Prog2/Konyv/Feladatok/Liskov$
```

12.4. Hello, Android!

Élesszük fel az SMNIST for Humans projektet! <https://gitlab.com/nbatfai/smnist/tree/master/forHumans/-/SMNISTforHumansExp3/app/src/main> Apró módosításokat eszközölj benne, pl. színvilág.

Megoldás forrása:

A projekt felélesztéséhez és futtatásához szükség van egy Android fejlesztői környezetre. Ez jelen helyzetben az Android Studio. Ebben létre kell hozni egy új projektet majd bele kell helyezni az SMNIST for Humans kódját amit a feladat leírásában található linkről lehet elérni. Ha ez megvan akkor az alábbi mappa és fájl szerkezet látható.



Miután a szükséges fájlok a megfelelő helyekre kerültek, szükség lesz arra is hogy néhány változtatást véghez vigyünk a kódban ahhoz hogy megfelelően működjön a projekt. Elsőnek nézzük az **AndroidManifest.xml** fájlt. Nem kell nagy változtatást véghez vinni, mindössze annyit hogy az `<activity android:name=".MainActivity">` sorban a MainActivity-t ki kell cserélni az SMNISTE3Activity kulcsszóra.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.smnist">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".SMNISTE3Activity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Ha ezzel megvagyunk akkor a következő fájl amiben módosítást kell végrehajtani az **SMNISTE3Activity**. Itt a kód elejében kell egy kicsit változtatni ahhoz hogy jó legyen. Az eredeti osztály definíciót kell átírni a következő módon.

```
package com.example.smnist;  
import androidx.appcompat.app.AppCompatActivity;  
  
public class SMNISTE3Activity extends AppCompatActivity {
```

Az utolsó dolog amin módosítani kell ahhoz hogy működjön és lefusson a kód az, az **activity_smniste3.xml**. Itt is egy sort kell kicserélni és ha ez megvan, utána futtatható lesz a kód. Az **SMNISTSurfaceView** elérési útját kell kijavítani a helyesre.

```
<com.example.smnist.SMNISTSurfaceView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"/>  
</RelativeLayout>
```

Miután ezeket a módosítások végre lettek hajtva, az **SMNIST** már futtatható lesz egy telepített virtuális telefonon.

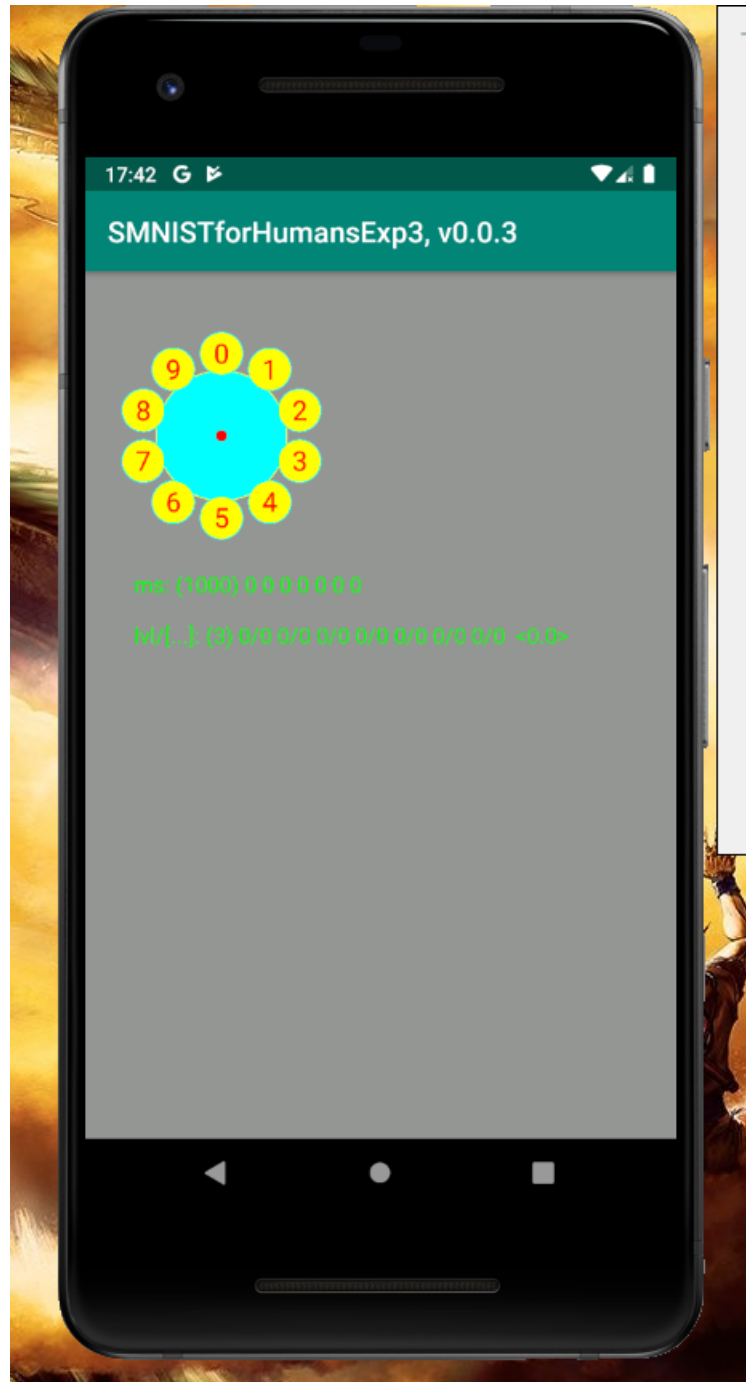
Most pedig lássuk min kell változtatni ahhoz hogy a színvilág megváltozzon a programban. Ehhez az **SMNISTSurfaceView.java**-ban kell néhány módosítást végezni. Először is a háttér színét változtassuk meg. Ennek kivitelezéséhez az alábbi kód részletben kell az rgb színkódot a megfelelőképpen módosítani.

```
int[] bgColor =  
{  
    android.graphics.Color.rgb( red: 150, green: 150, blue: 150),  
    android.graphics.Color.rgb( red: 100, green: 100, blue: 100)  
};
```

Végezetül pedig ahhoz hogy az ábra színei is megváltozzanak, a **cinit** függvényben kell a setColor-ok paraméterét módosítani a kívánt színre.

```
private void cinit(android.content.Context context) {  
  
    textPaint.setColor(android.graphics.Color.RED);  
    textPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);  
    textPaint.setAntiAlias(true);  
    textPaint.setTextAlign(android.graphics.Paint.Align.CENTER);  
    textPaint.setTextSize(50);  
  
    msgPaint.setColor(android.graphics.Color.GREEN);  
    msgPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);  
    msgPaint.setAntiAlias(true);  
    msgPaint.setTextAlign(android.graphics.Paint.Align.LEFT);  
    msgPaint.setTextSize(40);  
  
    dotPaint.setColor(android.graphics.Color.RED);  
    dotPaint.setStyle(android.graphics.Paint.Style.FILL_AND_STROKE);  
    dotPaint.setAntiAlias(true);  
    dotPaint.setTextAlign(android.graphics.Paint.Align.CENTER);  
    dotPaint.setTextSize(50);  
  
    borderPaint.setStrokeWidth(2);  
    borderPaint.setColor(android.graphics.Color.CYAN);  
    fillPaint.setStyle(android.graphics.Paint.Style.FILL);  
    fillPaint.setColor(android.graphics.Color.YELLOW);  
}
```

Végül ha minden jól ment akkor az alábbi eredmény látható.



12.5. Ciklomatikus komplexitás

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

Megoldás forrása:

A ciklomatikus komplexitás lényege hogy egy képlet alapján, számban kifejezve megmondja egy adott programkódról annak összetettségét. A képlet úgy épül fel, hogy egy gráfot képezünk a kódból majd ennek segítségével számolunk. Ha kész a gráf akkor a következőt kell tenni. A gráf éleinek a számából ki kell

vonni a csúcspontok számát majd hozzá kell adni a program egységek számánk kétszeresét.

Egyszerű programok esetében akár manuálisan is ki lehet számolni a komplexitást, de összetettebb programok esetében már egyszerűbb egy segédprogrammal vagy online található kalkulátorral számolni. Most az utóbbi esetet használjuk. Ha ellátogatunk a <http://www.lizard.ws/#try> oldalra akkor ott online is ki lehet számoltatni egy kódnak a komplexitását. Ennek a feladatnak az elvégzéséhez a BinfA java kódján lett lefuttatva az ellenőrzés. Az eredmény az alábbi lett.

Function Name	NLOC	Complexity	Token #	Parameter #
LZWBinFa::LZWBinFa	3	1	9	
LZWBinFa::egyBitFeldolg	22	4	104	
LZWBinFa::kiir	4	1	26	
LZWBinFa::kiir	4	1	22	
LZWBinFa::Csomopont::Csomopont	5	1	21	
LZWBinFa::Csomopont::nullasGyerek	3	1	8	
LZWBinFa::Csomopont::egyGyerek	3	1	8	
LZWBinFa::Csomopont::ujNullasGyerek	3	1	11	
LZWBinFa::Csomopont::ujEgyGyerek	3	1	11	
LZWBinFa::Csomopont::getBetu	3	1	8	
LZWBinFa::kiir	15	3	107	
LZWBinFa::getMelyseg	5	1	21	
LZWBinFa::getAtlag	6	1	32	
LZWBinFa::getSzoras	12	2	68	
LZWBinFa::rmelyseg	11	3	51	
LZWBinFa::ratlag	12	4	66	
LZWBinFa::rszoras	12	4	78	
LZWBinFa::usage	3	1	14	
LZWBinFa::main	64	14	401	

13. fejezet

Helló, Berners-Lee!

13.1. Élmény beszámoló a Python könyvből!

A Python egy nagyon sokoldalú platform független programozási nyelv. Leginkább kód részletek tesztelésére és prototípusok készítésére szokták használni de nem csak erre alkalmas. Mivel lehetséges objektumorientált programokat is írni benne, így összetettebb programokat is meg lehet benne írni. Továbbá tartalmaz sok beépített eljárást és képes összetett adatszerkezetekkel is dolgozni. Ilyen például a lista vagy a szótár, amelyről a későbbiekben még ejtek szót.

Most viszont lássuk a nyelv főbb jellemzőit. A legtöbb program nyelvnél a program megírása közbe szükség van a fordításra is a program futtatása előtt. A Python nyelvnél viszont erre nincs szükség mivel ott ezt a lépést elvégzi az interpreter. Továbbá egyéb előny az is hogy nincs szükség a kód tagolásánál zárójelezésre. Mivel a Pythonban a behúzások jelentik a tagolást. Egyedüli kitétel az az hogy a tagolásnak egységesnek kell lenni. Tehát ha először tabulátorral tagoltunk, utána végig azzal kell. Egy másik hasznos tulajdonság hogy az utasításokat nem kell lezárni. Egy utasítás a sor végig tart, ha új utasítást akarunk beírni akkor egyszerűen azt egy új sorba kell írni.

A Python többféle változóval és adatszerkezettel rendelkezik. A változók lehetnek számok és stringek valamint itt is létezik a NULL értéknek megfelelő változó típus. A számokon belül is lehetnek egészek és lebegőpontosak is. Ami viszont érdekesebb, hogy nincs szükség külön megadni az egyes változók típusait, mivel ezt a Python automatikusan megállapítja az alapján hogy mit tárolunk a változóban. Ahogy már korábban is említettem vannak összetett adatszerkezetek is. Ilyenek a listák, a szótárok vagy az ennesek. Ezek azért nagyon jók mert egy listán belül például több adatot is el tudunk tárolni, még akkor is ha azok nem egy típusúak. Így tehát egyszerre tartalmazhat akár számot és stringet is. Valamint dinamikus a méretük így folyton bővíthető a megfelelő méretűre. A szótár és az ennesek is ilyenek annyi különbséggel hogy a szótáraknak másképp működik az indexelése, az enneseknek pedig fix a tartalmuk és nem lehet módosítani azt. A Python nyelv továbbá tartalmaz beépített függvényeket amik nagyban megkönnyítik a lista használatát, mivel sok hasznos funkció előre meg van írva és már csak használni kell.

A program nyelvi eszközei nagyban hasonlítanak a többi programozási nyelvre. Itt is megtalálható az elágazás, a ciklusok több fajtája és a címkék is. Az elágazás ugyan úgy működik mint más nyelvekben és a ciklusok is nagyon hasonlóak. Ami eltér egy kicsit a többi nyelvtől az a for ciklus mert annak a megadása során lehet másképp is megadni a for-t ha listákra akarom alkalmazni a ciklust. Továbbá a címkék is használhatók a Pythonba akárcsak a többi nyelvben. Meghatározunk egy címkét és a goto utasítással oda lehet ugrani a címkéhez hogy onnan folytatódjon a program futása.

A függvények és az osztályok használata is teljes mértékben kivitelezhető. Ezek használata nagyban hasonlít például a C++ nyelvre. Vannak különbségek de a működési elvük nagyrészt ugyanaz. A különbségek egyike például az hogy a függvényeknél akárcsak a változóknál nincs szükség arra hogy meg mondjuk milyen típusú lesz az elem amit vissza fog téríteni. Ezen kívül a függvényeknél lehetőség van arra is hogy amikor meghívom, akkor a beadott változónak a meghívás közben határozzam meg az értékét. Most lássunk pár dolgot az osztályokról. Az osztályok az objektumorientált programozás részét képezik. Ez már egy magasabb szintű programozás részét képezi. Rengeteg dologra lehet használni de a könyv nem sok dologra tér ki. Az egyik dolog amire kitér hogy az osztályok képesek más osztályoktól örökölni.

13.2. A java és a C++ nyelv összehasonlítása

Legyen szó akár a C++, akár a Java programozási nyelvről, mind a kettő objektumorientált. Ez azt jelenti, hogy lehetséges összetettebb kódokat írni bennük és bonyolultabb problémákat lemodellezni velük. Mind a két nyelv rendelkezik az objektumorientált nyelvek alapelveivel: egységbe záras, adatrejtés és öröklés. Az objektumorientált nyelvek egyik fő eleme az osztályok, és noha mind a két nyelvben megtalálható, vannak eltérések a C++ és Java általi kivitelezésben. Míg a C++ nyelvben lehet olyan futtatható kódot írni amibe nem kell osztályt használni, addig ez a Java nyelvben nem kivitelezhető. A Java nyelvben ugyanis a legkisebb önálló egységek az osztályok.

Az osztályoknak lehet létre hozni példányait, amik az adott osztály tulajdonságaival rendelkeznek. Ilyen például az ha létrehozunk egy fa nevű osztályt. A fa osztálynak meglehet adni, hogy milyen tulajdonságokat lehessen benne tárolni. Úgy mint a fa fajtája, mérete, felhasználási területe. Ezeket az adatokat általában az osztályban létrehozott változóknál tároljuk, melyek az osztályon kívülről legtöbbször függvényekkel lehet megadni vagy módosítani.

Az osztályok változóit ugyanis a legtöbb esetben védetté teszik, hogy ne lehessen olyan értéket megadni ami helytelen. És mivel a függvényeket úgy szokták megadni, hogy tartalmaz hibakezelést, ez kivédi a program futása közbeni ilyen típusú hibákat. A változók védelmét a **private** kulcsszóval szokták kivitelezni. Ez ugyanis lehetővé teszi hogy ami a private részben szerepel azt ne lehessen máshonnan elérni csak az osztályon belülről. Létezik egy másik kulcsszó is ami a **public**. Az ehhez tartozó részek azok amelyek elérhetők az osztályon kívülről is, így általában ide kerülnek a függvények amelyeket majd fel lehet használni.

Ha alapból a programozó nem állítja be az osztálybéli változók elérhetőségét, akkor minden változó private lesz. Ezt azonban lehet módosítani. A C++ nyelvben egyszerűen az osztály public részében hozom létre a változót, míg a Java nyelvben az osztály létrehozásánál kell ezt jelezni, szintén a public kulcsszóval.

Kezdőértékkel is el lehet látni az osztályokban szereplő változókat. Viszont ha ezt nem tesszük meg akkor a C++-ban a kezdőérték egy véletlen érték lesz, Java-ban pedig 0. Viszont ezek az értékek akár hibát is eredményezhetnek amikor egy példány létrehozásra kerül. Azért hogy ez ne történjen meg mind a két programnyelv a konstruktorokat használja.

A konstruktorok a példány létrehozásakor automatikusan lefutnak és a céljuk az hogy a példányosítás során kezeljék a kezdőértékeket. Ha a programozó nem hoz létre konstruktort amely kezeli a kezdeti értékeket akkor a programnyelvbe alapból beépített konstruktor fog lefutni.

IV. rész

Irodalomjegyzék

13.3. Általános

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

13.4. C

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

13.5. C++

[BMECPP] Benedek, Zoltán És 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 , 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.