

Univerzális programozás

https://github.com/puskasmate/Prog2_forrasok

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

Copyright © 2019 Puskás Máté

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

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

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

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

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

COLLABORATORS

| | <i>TITLE :</i> | | |
|---------------|--|-------------------|------------------|
| | Univerzális programozás | | |
| <i>ACTION</i> | <i>NAME</i> | <i>DATE</i> | <i>SIGNATURE</i> |
| WRITTEN BY | Bátfai, Norbert ÁCs Puskás, Máté | 2019. december 9. | |

REVISION HISTORY

| NUMBER | DATE | DESCRIPTION | NAME |
|--------|------------|--|-------------|
| 0.0.1 | 2019-02-12 | Az iniciális dokumentum szerkezetének kialakítása. | nbatfai |
| 0.0.2 | 2019-02-14 | Inciális feladatlisták összeállítása. | nbatfai |
| 0.0.3 | 2019-02-16 | Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába. | nbatfai |
| 0.0.4 | 2019-02-19 | Aktualizálás, javítások. | nbatfai |
| 0.0.5 | 2019-05-09 | Feladatok, könyv kész. | Puskás Máté |
| 0.0.5 | 2019-11-20 | Második felvonás feladatok, könyv kész. | Puskás Máté |

Ajánlás

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

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

Tartalomjegyzék

| | |
|--|----------|
| I. Bevezetés | 1 |
| 1. Vízió | 2 |
| 1.1. Mi a programozás? | 2 |
| 1.2. Milyen doksikat olvassak el? | 2 |
| 1.3. Milyen filmeket nézzek meg? | 2 |
| II. Tematikus feladatok | 3 |
| 2. Helló, Turing! | 5 |
| 2.1. Végtelen ciklus | 5 |
| 2.2. Lefagyott, nem fagyott, akkor most mi van? | 6 |
| 2.3. Változók értékének felcserélése | 8 |
| 2.4. Labdapattogás | 9 |
| 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS | 12 |
| 2.6. Helló, Google! | 13 |
| 2.7. A Monty Hall probléma | 15 |
| 2.8. 100 éves a Brun téTEL | 17 |
| 3. Helló, Chomsky! | 20 |
| 3.1. Decimálisból unárisba átváltó Turing gép | 20 |
| 3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen | 21 |
| 3.3. Hivatkozási nyelv | 21 |
| 3.4. Saját lexikális elemző | 23 |
| 3.5. l33t.l | 25 |
| 3.6. A források olvasása | 28 |
| 3.7. Logikus | 29 |
| 3.8. Deklaráció | 30 |

| | |
|---|------------|
| 4. Helló, Caesar! | 33 |
| 4.1. double ** háromszögmátrix | 33 |
| 4.2. C EXOR titkosító | 36 |
| 4.3. Java EXOR titkosító | 37 |
| 4.4. C EXOR törő | 38 |
| 4.5. Neurális OR, AND és EXOR kapu | 41 |
| 4.6. Hiba-visszaterjesztéses perceptron | 43 |
| 5. Helló, Mandelbrot! | 45 |
| 5.1. A Mandelbrot halmaz | 45 |
| 5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal | 48 |
| 5.3. Biomorfok | 50 |
| 5.4. Mandelbrot nagyító és utazó C++ nyelven | 52 |
| 6. Helló, Welch! | 57 |
| 6.1. Első osztályom | 57 |
| 6.2. LZW | 59 |
| 6.3. Fabejárás | 63 |
| 6.4. Tag a gyökér | 69 |
| 6.5. Mutató a gyökér | 77 |
| 6.6. Mozgató szemantika | 85 |
| 7. Helló, Conway! | 87 |
| 7.1. Hangyszimulációk | 87 |
| 7.2. Qt C++ életjáték | 103 |
| 7.3. BrainB Benchmark | 112 |
| 8. Helló, Gutenberg! | 128 |
| 8.1. Programozási alapfogalmak | 128 |
| 8.2. Programozás | 134 |
| III. Második felvonás | 136 |
| 9. Helló, Arroway! | 138 |
| 9.1. OO szemlélet | 138 |
| 9.2. "Gagyí" | 140 |
| 9.3. Yoda | 142 |
| 9.4. Kódolás from scratch | 144 |

| | |
|--|------------|
| 10. Helló, Liskov! | 148 |
| 10.1. Liskov helyettesítés sértése | 148 |
| 10.2. Szülő-gyerek | 150 |
| 10.3. Anti OO | 152 |
| 10.4. Ciklomatikus komplexitás | 164 |
| 11. Helló, Mandelbrot! | 166 |
| 11.1. Reverse engineering UML osztálydiagram | 166 |
| 11.2. Forward engineering UML osztálydiagram | 167 |
| 11.3. BPMN | 172 |
| 12. Helló, Chomsky! | 174 |
| 12.1. ENCODING | 174 |
| 12.2. l334d1c4^5 | 176 |
| 12.3. Full screen | 179 |
| 13. Helló, Stroustrup! | 183 |
| 13.1. JDK Osztályok | 183 |
| 13.2. Másoló-mozgató szemantika és Összefoglaló egybevonva | 186 |
| 14. Helló, Gödel! | 190 |
| 14.1. Gengszterek | 190 |
| 14.2. STL Map érték szerinti rendezése | 193 |
| 14.3. Alternatív tabella | 195 |
| 15. Helló, Valami! | 200 |
| 15.1. FUTURE Tevékenység Editor | 200 |
| 15.2. SamuCam | 202 |
| 15.3. BrainB | 206 |
| 16. Helló, Lauda! | 208 |
| 16.1. Port scan | 208 |
| 16.2. AOP | 210 |
| 16.3. Junit teszt | 212 |

| | |
|--|------------|
| 17. Helló, Calvin! | 215 |
| 17.1. MNIST | 215 |
| 17.2. Deep Mnist | 218 |
| 17.3. Android telefonra a TF objektum detektálója | 220 |
| 18. Helló, Berners-Lee! | 224 |
| 18.1. Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba | 224 |
| 18.2. Java 2 Útikalauz Programozóknak 5.0 | 226 |
| IV. Irodalomjegyzék | 228 |
| 18.3. Általános | 229 |
| 18.4. C | 229 |
| 18.5. C++ | 229 |
| 18.6. Lisp | 229 |

Táblázatok jegyzéke

10.1. A táblázat: 164

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 alkalmi igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

Minden esetre 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 gyereknek, 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 gyereknek, 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 más is) példával.

Hogyan nyomjuk?

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

Ha megvannak a könyv DocBook XML forrásai, akkor az alább látható **make** parancs ellenőrzi, hogy „jól formázottak” és „érvényesek-e” ezek az XML források, majd elkészíti a 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 találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

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!

Egy magot 100%-ban kétféleképpen dolgoztathatunk. Használhatunk hozzá while illetve for ciklust is.

Megoldás while ciklussal:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Turing/2.1/egymag100-w.c

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

int main() {
    while(true) {

    }
    return 0;
}
```

A két megoldás közül ez az "elegánsabb".

Megoldás for ciklussal:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Turing/2.1/egymag100-f.c

```
#include <stdio.h>

int main() {
    for ( ; ; ) {

    }
    return 0;
}
```

A két módszer közül viszont ez az egyértelműbb megoldás, ugyanis a két db ";" egyértelművé teszi a szándékunkat.

Több mag 100%-on:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Turing/2.1/osszesmag100.c

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

int main(){

#pragma omp parallel
#pragma omp while

    while(1){

    }
return 0;
}
```

Ha több magot szeretnénk 100%-on dolgoztatni, a parallel pragma-t kell segítségül hívunk, amely az őt követő utasításokat több szálra hajtja végre, ezáltal az összes magot képesek leszünk megdolgoztatni, és mivel a pragma után végtelen ciklus van, ezért azt 100%-on tesszük. A kód elején meg kell hívunk az "omp.h" headert. A **gcc osszesmag100.c -o osszesmag100 -fopenmp** parancssorral fordítható le.

Egy mag 0%-on:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Turing/2.1/egymag0.c

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

int main(){

while(1){

sleep(100);
}
}
```

Ha a processzor magok közül egyet szeretnénk használni, azt is 0%-on, azt csak a sleep függvény használatával lehetjük meg. A sleep függvény az "unistd.h" header-ben található meg, így a program elején meg kell hívunk ezt a headert. A sleep függvény "elaltatja"az adott processzormagot, így a végtelen ciklusunk 0%-ban fogja használni a processzormagot.

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

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

Megoldás videó:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

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

}
```

Mit 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.

Egy szóval nem létezik olyan program, amely el tudja döntenı egy másik programról, hogy az lefog e vagy sem.

2.3. Változók értékének felcserélése

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

Két változó értékét segédváltozó és logikai utasítás nélkül 2 féleképpen cserélhetjük fel.

Az egyik egyszerű megoldás összeadás és kivonás segítségével valósítható meg:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Turing/2.3/csereossszeg.c

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

int main(){

    int a =10;
    int b =5;

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

    printf("A változók felcserélve: %d, %d\n", a, b);

    return 0;
}
```

Egy másik, az elsőhöz nagyon hasonló megoldás, amely összeadás és kivonás helyett szorzás és osztás segítségével jöhet létre:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorial/bhax_textbook/Turing/2.3/csereszorzat.c

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

int main(){
    int a=10;
    int b=5;

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

    printf("A változók felcserélve: %d, %d\n", a, b);

    return 0;
}
```

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! (Hogy mit értek pattogtatás alatt, alább láthatod a videónkon.)

Megoldás forrása: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorial/bhax_textbook/Turing/2.4/labdapattogif.c

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

A grafikus megjelenítést a terminálban a curses mód teszi lehetővé, amelyet az initscr() függvény használatával lehet bekapcsolni.

```
int x = 0;
int y = 0;

int xnov = 1;
int ynov = 1;

int mx;
int my;
```

Az x és y a sor és az oszlopok kezdete. Az "xnov" és az "ynov" azt mondja meg, hogy vízszintesen és függőlegesen mennyit "lépjén" a labda.

```
for (;;) {
```

```
getmaxyx ( ablak, my , mx );

mvprintw ( y, x, "O" );

refresh ();
usleep ( 100000 );

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

A getmaxyx() függvény lekérdezi az adott ablak sorainak és oszlopainak számát. Az mvprintw() függvény a megadott x és y koordinátkra mozog a kurzorral és - ebben az esetben - az "O" karaktert írja ki, a refresh() függvény pedig folyton frissíti az átadott paraméterek értékeit, mivel az x és y értéke folyamatosan változik, ugyanis vízszintesen "xnov", függőlegesen "ynov" értéket lép, így ezeket x és y értékéhez hozzáadjuk. A usleep() függvény egy értékül megadott mikrosecond-nyi időre megállítja a végtelen ciklusunk futását, ezáltal tudjuk követni a labda mozgását. Ez az érték minél nagyobb, annál lassabban pattog a laba.

```
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;
}
```

Végül pedig feltételes utasítások következnek, amelyek azt vizsgálják meg, hogy a labda elérte-e az ablak tetejét, alját, bal oldalát és jobb oldalát, és ha igen, a léptetés értékét a -1 szeresére változtatja meg, ami miatt a labda az ellenkező irányba pattog tovább.

A programot a **gcc labdapattogif.c -o labdapattogif -lncurses** parancssorral futtathatjuk le.

Az if nélküli változat:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutor/bhax_textbook/Turing/2.4/labdapattog.c

```
int magassag[my];
int szelesseg[mx];
```

Ebben a változatban tömbök segítségével határozzuk meg a pálya méretét.

```
//Feltételes utasítások helyett most for ciklusokat használunk
```

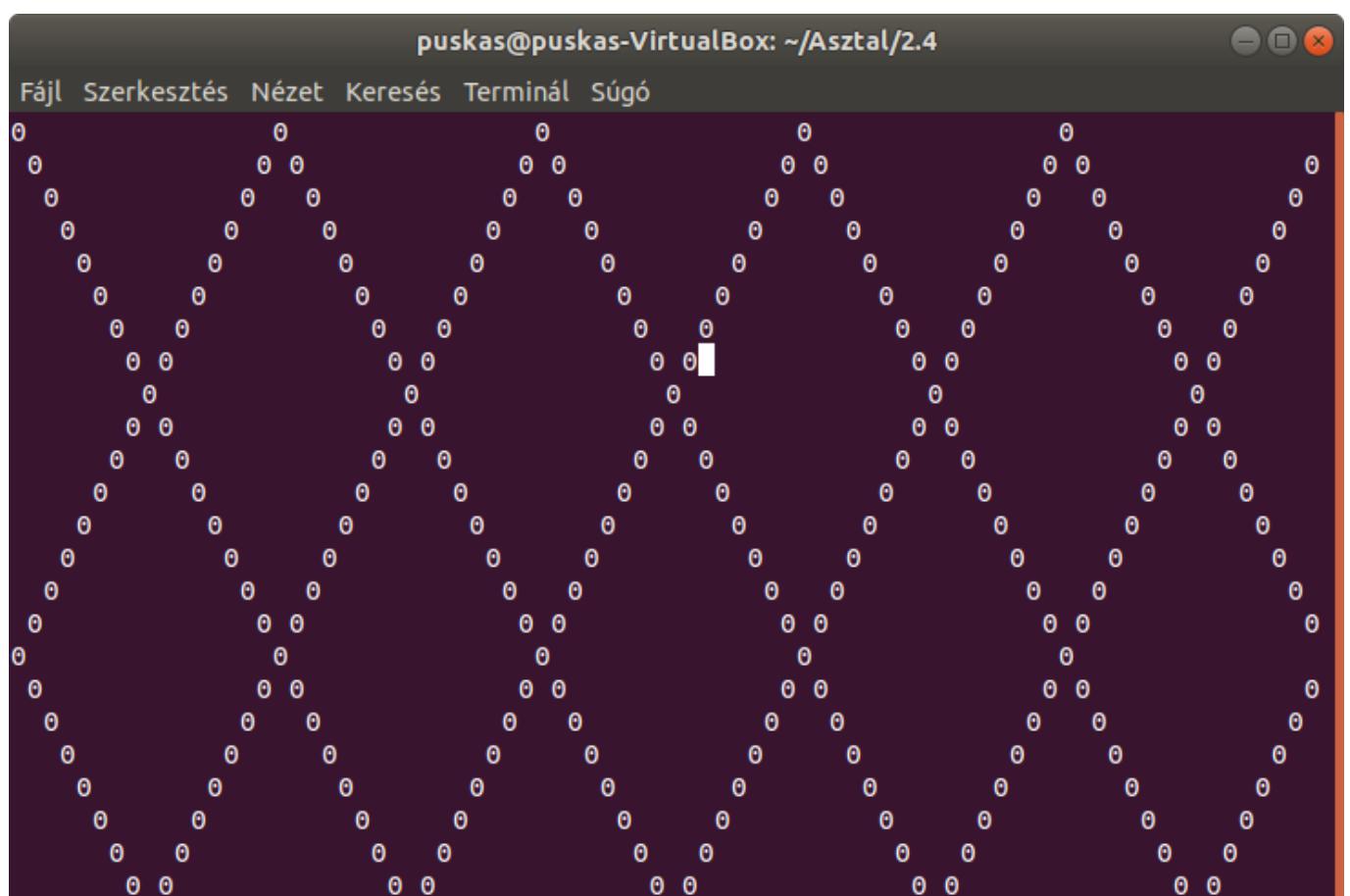
```
for(int i=0;i<mx;i++) {
    szelesseg[i]=1;
}
```

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

//A keret elérése miatt a kezdő értéket és az utolsó előtti értéket ←
megváltoztatjuk a többi elem abszolút értékére

```
szelesseg[0]=-1;  
magassag[0]=-1;  
szelesseg[mx-1]=-1;  
magassag[my-1]=-1;
```

A két forráskód között a lényeges különbség itt látható. Ebben a változatban feltételes utasítás nélkül vizsgáljuk meg, hogy a labda elérte e a pálya széleit, tetejét vagy alját. Elindítunk egy for ciklust, amely a magasság maximumáig minden elemet 1-gyel tesz egyenlővé, majd ugyanezt megcsináljuk a szélességgel is, majd a kezdőértékeket és végértékek előtti értékeket (azért végérték -1, hogy csak hozzájön a pálya keretéhez) -1-gyel tesszük egyenlővé. Ugyanis ha nem az utolsó előtti értéket, hanem az utolsó értéket tesszük -1-gyel egyenlővé, az azt eredményezné, hogy a labda minden ugyanazon az íven pattog (ahogyan az alább látható).



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!

Szóhossz:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Turing/2.5/szohossz.c

```
#include <stdio.h>

int main(void) {

    int a = 5;
    int b = 0;

    while (a != 0) {
        a=a<<1;
        b++;
    }

    printf ("A szóhossz ezen a gépen: %d bites\n", b);
    return 0;
}
```

A "szohossz.c" program megvizsgálja, hogy az adott változó hány bittel rendelkezik. Ehhez egy while ciklusban addig tolja a változó bitjeit egyel balra, amíg az értéke 0 nem lesz. minden egyes toláskor egy változó értékét növeljük, és ez a változó tárolja azt az értéket, amire kíváncsiak vagyunk.

BOGOMIPS:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Turing/2.5/bogomips.c

```
void
delay (unsigned long long loops)
{
    for (unsigned long long i = 0; i < loops; i++);
}
```

A "delay" függvényt úgy deklaráljuk, hogy ez a függvény 1-től a paraméterként adott értékig egyesével elszámol.

```
while ((loops_per_sec <= 1))
{
    ticks = clock ();
    delay (loops_per_sec);
    ticks = clock () - ticks;
```

A while ciklusban a ciklus feltételenél 2 hatványokat tologatunk. A "ticks" változó értéke a program kezdete óta eltelt időt tartalmazza. Kiszámoljuk a "delay" függvény értékét a "loops_per_sec"-re. Ezután a "ticks" változóba ismét lekérem a "clock" függvényt az aktuális processzor órajelet, majd ebből levonom a "ticks" eddigi értékét, ezáltal megkapom, hogy a "delay" függvény lefutásához mennyi időre lesz szükség.

```
if (ticks >= CLOCKS_PER_SEC)
```

Ezután megnézem, hogy az eltelt idő nagyobb e mint a "CLOCKS_PER_SEC". Tehát ez a ciklus addig megy, amíg a "delay" függvény lefutásához 1.000.000 (ugyanis a time.h header-ben ennyi a "CLOCKS_PER_SEC" értéke) "óraütés" lesz szükséges.

```
loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
       (loops_per_sec / 5000) % 100);

return 0;
}

printf ("failed\n");
return -1;
}
```

Ha a "ticks" elérte, vagy túllépte a "CLOCKS_PER_SEC" értékét, a "loops_per_sec" értéke arra változik meg, hogy mennyi lenne az értéke akkor, ha nem 2 hatványokkal számolnánk.

2.6. Helló, Google!

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

Megoldás forrása:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorial_bhax_textbook/Turing/2.6/pagerank.c

```
void kiir (double tomb[], int db)
{
    for (int i=0; i<db; i++) {
        printf("PageRank [%d]: %lf\n", i, tomb[i]);
    }
}
```

Ez egy egyszerű kiíratási függvény, amely a paraméterül kapott PageRank értékeket fogja kiíratni.

```
double tavolsag (double PR[], double PRv[], int n){
double tav = 0.0;

for(int i=0; i<n; i++) {
    tav += (PR[i] - PRv[i]) * (PR[i] - PRv[i]);
}

return sqrt(tav);
}
```

A "tavolsag" függvény paraméterül két PageRank értéket kér, és az oldalak számát. Egy for ciklusban végigmegy az elemeken, és előállítja azok távolságát, majd a távolság négyzét adja vissza.

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

A main függvényben egy 2 dimenziós tömb segítségével létrehozzuk az oldalak linkjének mátrixát. Itt az egyes oszlopok oldalakat tárolnak.

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

A PR tömb tárolja az oldalak PageRank értékét, ezt kezdetben kinullázzuk, a PRv tömb pedig a következő PageRank értéket tárolja el, ez minden esetben 1/4.

```
int i, j;  
  
for (;;) {  
    for(int i=0; i<4; i++) {  
        PR[i] = PRv[i];  
    }  
  
    for (int i=0; i<4; i++) {  
        double tmp = 0.0;  
  
        for (int j=0; j<4; j++) {  
            tmp += L[i][j] * PR[j];  
        }  
        PRv[i] = tmp;  
    }  
}
```

Itt a PageRank értékek kiszámolása látható. Egy oldal PageRank értékét úgy számoljuk ki, hogy végigmegyünk az "L" mátrixban az oldalhoz tartozó soron, és megszorozzuk a PageRank-jával, amit láthatjuk, ez az első körben mindenhol 1/4-edet jelent. Majd vesszük ezek összegét, és a következő PageRank értéket (PRv tömb) is kiszámoljuk az előbb leírtak alapján.

```
if (tavolsag (PR, PRv, 4) < 0.0000000001) {  
    break;  
}  
  
kiir(PR, 4);  
  
return 0;  
}
```

Végezetül pedig megvizsgáljuk a "tavolsag" függvény segítségével, hogy 4 oldalra a PageRank értékek közötti távolság kisebb lesz e mint 0.0000000001, ha igen, kilépünk a ciklusból és a kiir() függvény segítéssel kiíratjuk a PageRank értékeket.

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyben_a_monty_hall-paradoxon_kapcsan

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

A Monty Hall probléma a nevét egy amerikai műsorvezetőről kapta. A paradoxon alapjául a vetélkedő utolsó feladata szolgál. Ebben a feladataban ugyanis a játékosnak 3 ajtó közül kell választania egyet. A három ajtó közül kettő mögött 1-1 kecske áll, a harmadik mögött pedig egy luxusautó. A műsorvezető (aki minden esetben tud arról, hogy melyik ajtó mögött mi van) a maradék két ajtó közül kinyit egyet, amely mögött biztos, hogy egy kecske áll. A problémát az ez által létrejött kérdés jelenti, ami pedig az, hogy a versenyzőnek érdemes-e megváltoztatnia választását. A józan ész azt diktálja, hogy teljesen mindegy, de valószínűségszámítással bebizonyítható ennek az ellentéte, ez szüli tehát a pradoxont.

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

A kiserletek_szama változó értéke ahogy a nevében is benne van a kísérletek száma lesz. A "kiserlet" vektor pedig kiválasztja a sample függvény segítségével, hogy az adott ajtók közül melyik mögött van a nyeremény. A "jatekos" vektorban az egyes kísérletekben a játékos által választott ajtókat tároljuk el ugyanúgy a sample függvény használatával, és a "replace=T"-vel egy ajtó (szám) többszöri szereplését is lehetővé tesszük mind a két vektorban. A "musorvezeto" vektor hosszúságát pedig beállítjuk, hogy egyenértékű legyen a kísérletek számával, ugyanis a műsorvezetőnek minden kísérletben kell ajtót választania.

```
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]))
  }
}
```

Ezután egy for ciklussal végig megyünk az egyes kísérleteken, és megvizsgáljuk, hogy az adott kísérletben a játékos eltalálta-e, hogy melyik ajtó mögött található a nyeremény. Ha igen, a "mibol" vektor értékét beállítjuk úgy, hogy a setdiff() függvény segítségével a 3 ajtó közül kiveszük annak az ajtónak a számát, amely mögött a nyeremény található, és mivel jelen esetben a "nyerő" ajtó megegyezik a játékos által

választott ajtóval, így a játékos által választott ajtót nem lehet kivenni a lehetőségek közül. Amennyiben viszont a játékos nem találta el a nyertes ajtót, a "mibol" vektor (ugyanúgy a setdif() függvény segítségével) a nyertes ajtó és a játékos által választott ajtót kiveszi a lehetőségek közül, így a műsorvezetőnek a másik "kecskés" ajtót kell kinyitnia.

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

A "mibol" vektorból tehát előáll hogy a műsorvezetőnek melyik ajtót kell kinyitnia.(Ha a játékos eltalálta a nyereményt, a két "kecskés" ajtó közül szabadon választhat, hogy melyiket szeretné kinyitni.)

```
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)]  
  
}
```

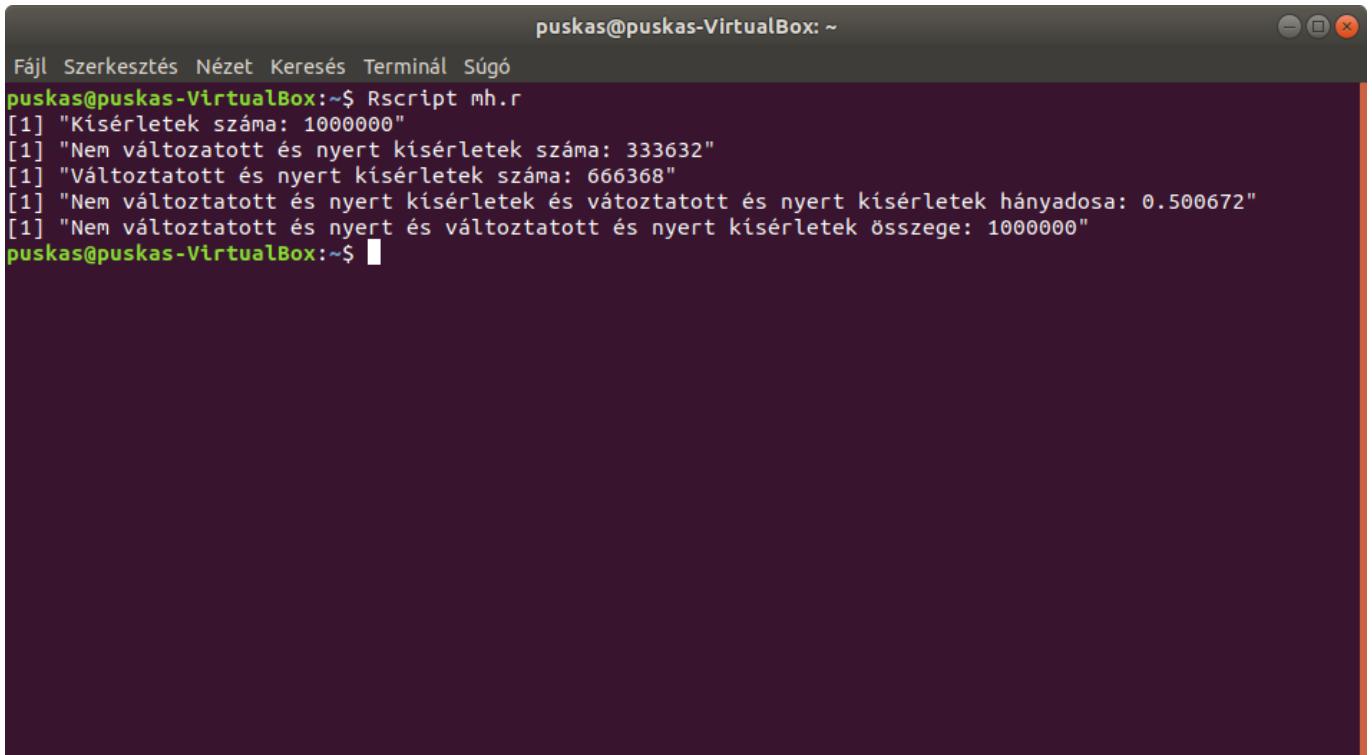
Ezután a "nemvaltoztatesnyer" vektor megnézi, hogy mely esetekben találta el a játékos a nyertes ajtót. Ez a szám azt fogja eredményül adni, hogy a kísérletek közül hány esetben nyert a játékos a választása megváltoztatása nélkül. A "valtoztat" vektorban azokat az eseteket tároljuk, amelyben a játékos megváltoztatja a döntését. A "holvalt" vektorban a setdiff() függvény segítségével megállapítjuk, hogy a 3 ajtó közül ha kivesszük a műsorvezető és a játékos által választott ajtót, melyik ajtóra változtathatta meg a játékos a döntését.

```
valtoztatesnyer = which(kiserlet==valtoztat)
```

A "valtoztatesnyer" vektor pedig azokat az eseteket tárolja, amely esetekben a játékos megváltoztatta döntését és így nyer.

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

Végül pedig a kiíratások kövezkeznek.

A screenshot of a terminal window titled "puskas@puskas-VirtualBox: ~". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ Rscript mh.r
[1] "Kísérletek száma: 1000000"
[1] "Nem változatott és nyert kísérletek száma: 333632"
[1] "Változatott és nyert kísérletek száma: 666368"
[1] "Nem változatott és nyert kísérletek és változatott és nyert kísérletek hányszáma: 0.500672"
[1] "Nem változatott és nyert és változatott és nyert kísérletek összege: 1000000"
puskas@puskas-VirtualBox:~$
```

Láthatjuk, hogy a változatott és nyert esetek száma közel a duplája a nem változatott és nyert esetek számának.

2.8. 100 éves a Brun téTEL

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

Léteznek olyan számok, amelyek csak 1-gyel és önmagukkal oszthatók, ezeket a számokat prímszámoknak nevezzük. minden pozitív egész szám előírható prímszámok szorzataként. Léteznek ún. ikerprímek is. Két prímszámot akkor nevezünk ikerprímeknek, ha különbségük 2. Brun tétele azt mondja ki, hogy az ikerprímek reciprokainak összegéből létrejött sorozat véges számot eredményez, vagy pedig végtelen, de véges számhoz konvergál a sorozat, és ez a szám lesz a Brun-konstans.

A Brun-téTEL demonstrálása R-ben:

```
stp <- function(x) {
```

Új függvényünket "stp" néven deklaráltuk. Ez a függvény "x"-re fogja kiszámolni a Brun-konstans értékét.

```
primes = primes(x)
diff = primes[2:length(primes)] - primes[1:length(primes)-1]
```

A "primes" egy olyan vektor lesz, ami a prímszámokat tárolja el 2-től x-ig. A "diff" vektor pedig a "primes" vektor második elemétől az utolsó eleméig tartó sor értékeiből vonja ki a "primes" vektor első elemétől az utolsó előtti eleméig tartó sor értékeit.

```
idx = which(diff==2)
```

Az idx vektor pedig a "diff" vektor azon elemeit foglalja magába, melyeknek az értéke 2.

```
t1primes = primes[idx]
t2primes = primes[idx]+2
```

Most, hogy tudjuk mely elemek különbsége 2 a "t1primes" vektor a "primes" vektor azon elmeit fogja felvenni, amelyek az "idx" vektor elemeivel megegyező sorszámmal rendelkeznek. A "t2primes" pedig ugyan azt az értéket 2-vel megnövelte (ikerprím!!) veszi fel. A két vektor tagjai képzik tehát az ikerprím párokat.

```
rt1plust2 = 1/t1primes+1/t2primes
return(sum(rt1plust2))
```

A Brun-tétel az ikeprímek reciprokösszegéről szól, így az "rt1plust2" vektor elemei a "t1primes" és a "t2primes" vektor elemeinek reciprokösszegei lesznek.

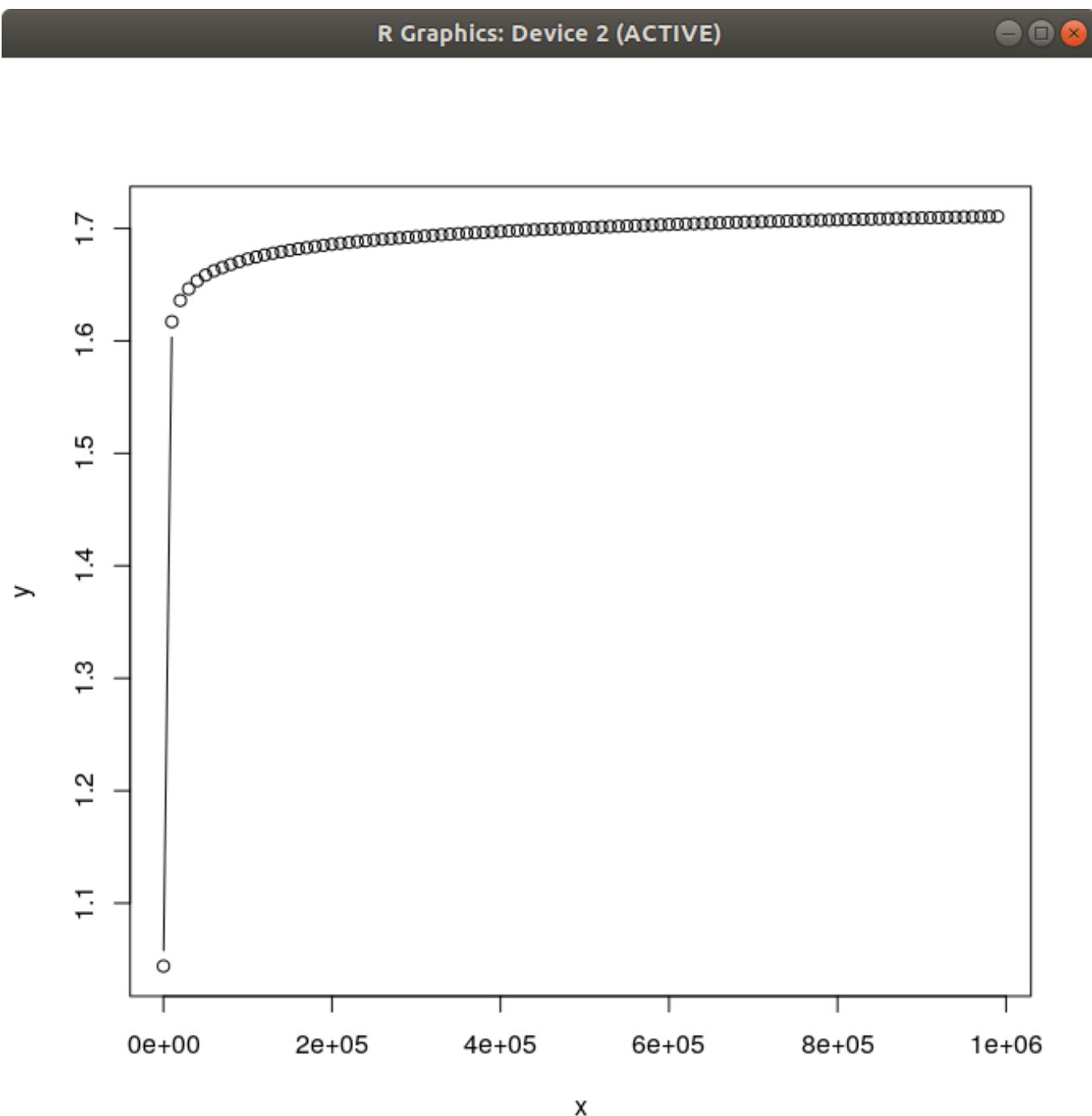
```
return(sum(rt1plust2))
}
```

A végén ezen vektor elemeit a sum() függvény segítségével összeadjuk, ezzel megkapjuk a Brun-konstans értékét.

A matlab segítségével készíthetünk egy grafikont is, mely a következő parancsok beírásával érhető el:

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

Pl.: stp(92)-re a következő grafikont kapjuk.



3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Az alábbi nagyon egyszerű program decimálisból vált át számokat unárisba.

```
#include "std_lib_facilities.h"

int main() {

    int a;
    std::cout<<"Adj meg egy számot!"<<std::endl;
    std::cin>>a;

    int t[a];
    for(int i=0;i<a;i++) {
        t[i]=1;
        t[a-1]=0;
    }

    for(int j=0;j<a;j++) {
        std::cout<<t[j];

    }

    return 0;
}
```

A program bekér egy számot, feltölt egy, a bekért szám hosszúságú tömböt csupa 1-esekkel, és a tömb legutolsó elemét 0-vá teszi. Ezután pedig kiírja a kapott tömb elemeit, ami a megadott szám unárisba

átváltott értékének felel meg.

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 környezetfüggű generatív grammatica elvét Noam Chomsky fogalmazta meg. A generatív grammatica a nyelvtant, mint ismeretet közelíti meg. Chomsky nézete, hogy a tudás és az ismeret legnagyobb része öröklött, erre pedig azzal hivatkozik, hogy a gyerekeknek elég csak a nyelvük jellegzetességeit megtanulni, és azt is elég gyorsan teszik. A másik érve, hogy nagy különbség van a bizonyos nyelvi ingerek és a gazdag nyelvtani ismeretek között. Elméletei nagy hatással voltak a nyelvészetre, viszont voltak kritikus hangok is.

3.3. Hivatkozási nyelv

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

Utasítások:

A programban utasítások sora jelenik meg, ezek a program futásakor sorban hajtódnak végre, viszont van lehetőség eme sorrendet "felrúgni". Az utasításokat több csoportra bonthatjuk:

1. A kifejezés utasítás

Ilyen jellegű a legtöbb utasítás. Alakja: kifejezés;

BNF-ben: $\langle \text{kifejezes} \rangle ::= \langle \text{kifejezes} \rangle$

2. Az összetett utasítás (blokk)

para

Ez egy utasítás helyett a blokban lévő összes utasítást végrehajtja.

BNF-ben: $\langle \text{utasitasok} \rangle ::= \{ \langle \text{utasitasok} \rangle \}$

3. A feltételes utasítás

A gép megvizsgálja, hogy a feltétel teljesül-e, ha igen, az első utasítást/utasításblokkot hajtja végre, ha nem, a második (else) utasítást/utasításblokkot.

BNF-ben: $\langle \text{if then else statement} \rangle ::= \text{if} (\langle \text{expression} \rangle) \langle \text{statement no short if} \rangle \text{else} \langle \text{statement} \rangle$

4. A while utasítás

Addig hajtja végre az utasítást/utasításokat amíg a while-ban lévő feltétel nem teljesül. Ez a ciklus előbb vizsgál, aztán hajtja végre az utasítást (elől tesztelős ciklus).

BNF-ben: $\langle \text{while utasitas} \rangle ::= \text{while} (\langle \text{feltetel} \rangle) \langle \text{utasitas} \rangle$

5. A do utasítás

Lényegében ugyanaz mint a while utasítás, abban különbözik, hogy ez a ciklus előbb végrehajtja az utasítást, utána végzi el a vizsgálatot, tehát a ciklus egyszer mindenkorban lefut (hátul tesztelős ciklus).

BNF-ben: *<do utasitas>* ::= **do** *<feltetel>* **while** (*<kifejezes>*) ;

6. A for utasítás

A for ciklusban felveszünk egy ciklusváltozót, amit általában 0 kezdőértékkal veszünk fel, ezután a feltétel következik, ebben általában azt mondjuk meg, hogy meddig hajtsa végre az utasítást a ciklus, ezután a ciklus továbblétése miatt a ciklusváltozó értékét egyel megnöveljük.

BNF-ben: *<for utasitas>* ::= **for** (*<for ciklusvaltozo>*? ; *<kifejezes>*? ; *<ciklusvaltozo noveles>*?) *<utasitas>*

7. A switch utasítás

A megadott kifejezéstől függően egy másik utasításnak adja át a vezérlést. A lehetséges utasításokat "case" előtaggal címkézzük meg. Előfordulhat olyan is hogy csak egy utasításra ugorhat át a végrehajtódás.

BNF-ben: *<switch utasitas>* ::= **switch** (*<kifejezes>*) *<case utasitas>*:

<switch utasitas> ::= **case** *<kifejezes>* : | **default** :

8. A break utasítás

A break utasítás után befejeződik a break-et körülvevő for, while, do, switch utasítás végrehajtása, és a végrehajtás a break után következő utasításra ugrik át.

BNF-ben: *<break utasitas>* ::= **break** *<identifier>*? ;

9. A continue utasítás

A continue utasítás után a végrehajtódás a continue követő legelső for, while, do utasítás ciklusfolytatónak részére fog átugrani.

BNF-ben: *<continue kifejezes>* ::= **continue** *<identifier>*? ;

10. A return utasítás

A return utána érték kerül visszatérésre.

BNF-ben: *<return utasitas>* ::= **return** *<kifejezes>*? ;

11. A goto utasítás

Az végrehajtódás feltétel nélkül a goto utasítás után megcímkézett utasításra ugrik át.

BNF-ben: *<goto utasitas>* ::= **goto** *<azonosito cimke>* ;

12. A címkézett utasítás

Bármelyik utasítást megelőzhetik. Az azonosítója a goto utasítás célpontjaként szolgál.

BNF-ben: *<cimkezett utasitas>* ::= *<azonosito>*? :

13. A null utasítás

Alakja: ;

A null utasítás hordozhat címkét, vagy szolgálhat ciklusok számára üres ciklustörzsként is.

```
#include <stdio.h>

//úgysem fogsz lefutni

int main(){

printf("Hello, World!");
return 0;
}
```

Ez a kis kódcsipet C89 szabvány használata mellett nem fog lefordulni, ugyanis ez a szabvány nem ismeri a kommentelésre használt dupla per jelet. C99-es szabvány használatával gond nélkül lefordul a program.

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 használunk, azaz óriások vállán állunk és ne kispályázzunk!

A lexikális elemző nagy részét nem magunk írjuk meg, mi csak egy kódcsipetet adunk át, amelyben megadjuk, hogy mit keressen a program.

A lexikális elemző karaktereket olvas be a kimenetről, és ezeket konvertálja tokenekké, szimbólumoká. Mindig a leghosszabb karaktersorozatból állít elő egy szimbólumot. Ezután ezeket a szimbólumokat egy általunk előre megírt módon elemzi.

Megoldás forrása:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorial_bhax_textbook/Chomsky/szamokszama.1

```
% {
#include <stdio.h>
int a = 0;
%}
```

A programban itt tudunk definiálni. Ez a rész be fog kerülni a lexer által generált C nyelvű forráskódba. Itt csak meghívtuk az "stdio.h" headert és felvettünk egy változót, ami később a szavak számát fogja összeszámolni, ezért kinulláltuk.

```
szjegyek [0-9]
```

```
%%
```

Itt definiáljuk, hogy milyen tokeneket keresünk, ez jelen esetben azt jelenti, hogy itt definiáltuk, hogy számjegyeket keresünk. A program első részének végét a "%%" jelenti.

```
{szjegyek}* (\.{szjegyek}+) ? {
    ++a;
    printf("\n");
```

```
    printf("[Valós szám:%s %f]", yytext, atof(yytext));
}
%%
```

Itt állítjuk be, hogy milyen szabály alapján keressen a program. Az {szjegyek}* rész azt jelenti, hogy valamilyen számjegyet keresünk, a "*" -gal pedig megadtuk, hogy bármennyi lehet belőle. A ("\\. {szjegyek}+) részben a "." -ot a "\" jel segítségével levédetjük, hogy a program csak az olyan pontokat keresse, amelyek mögött számjegy(ek) állnak. A "+" jel pedig arra utal, hogy legalább 1 számjegy legyen a "." mögött. Mivel ez a rész zárójelek között van, a bezáró zárójel mögött pedig "?" található, ezért ezt úgy értelmezzük, hogy ez a rész elhagyható. (Ugyanis egy egész szám nem tartalmaz tizedesvesszőt.) Ezután ha találtunk egy ilyen szimbólumot az "a" változó (ami a valós számok számát tartalmazza) értéket megnöveljük 1-gyel. Ezután a kapott karakterláncot string-ként kiíratjuk, és az atof() függvény segítségével a paraméterként kapott stringből double típusú változót képzünk, majd ezt is kiíratjuk.

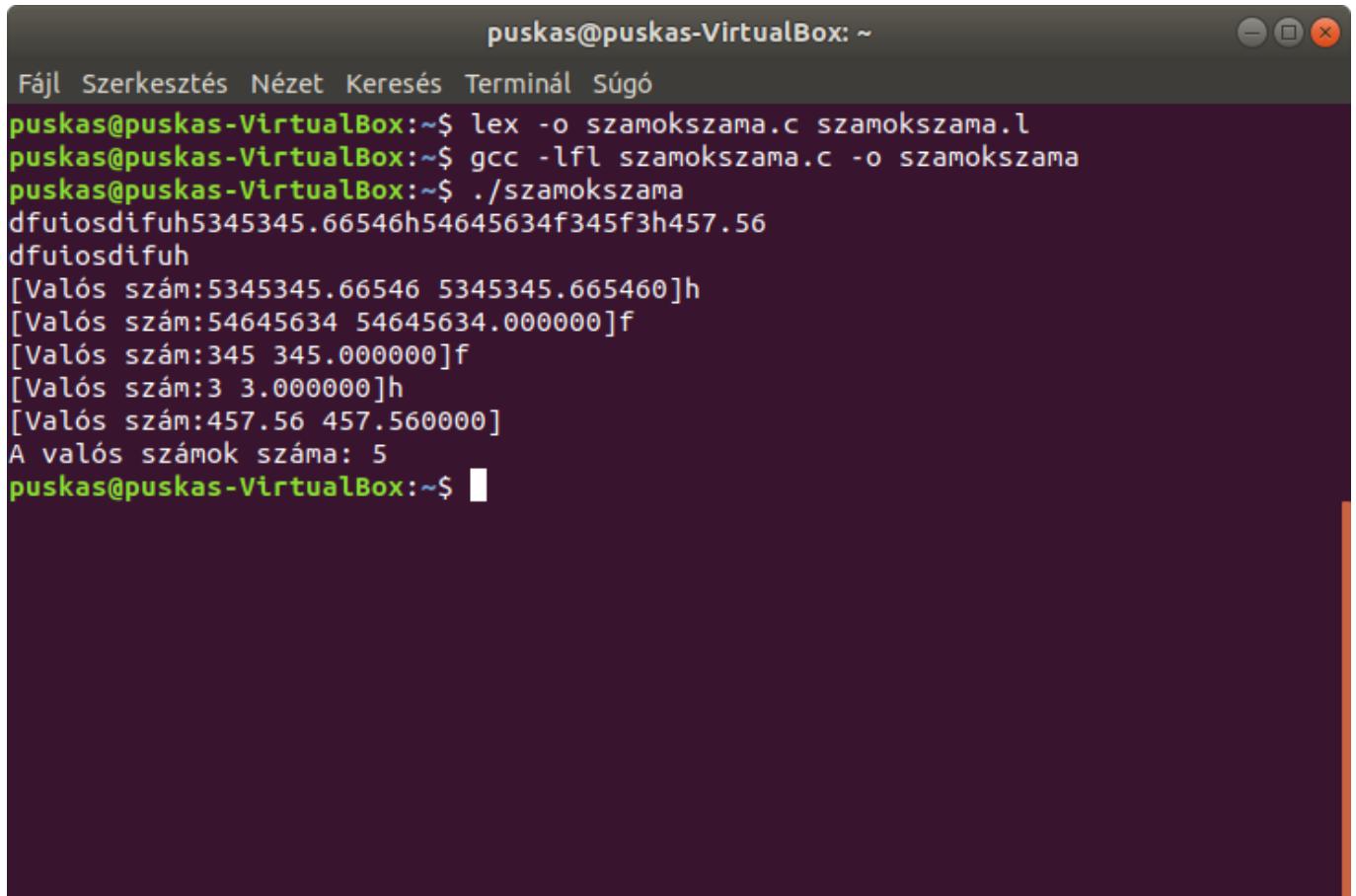
```
int
main ()
{
    yylex ();
    printf("A valós számok száma: %d\n", a);

    return 0;
}
```

Majd pedig szintén egy C forráskód következik, amelyben meghívjuk ezt az "yylex()" függvényt, és kiírjuk a valós számok számát.

A programot a következő parancssorok beírásával tudjuk lefuttatni.

lex -o szamokszama.c szamokszama.lEzzel egy C forráskódú állományt képzünk, amit gcc segítségével a következőképpen fordítunk le: **gcc -lfl szamokszama.c -o szamokszama**Végül pedig lefuttatjuk: **./szamokszama**. És alább láthatjuk, hogy programunk megfelelően működik. Ha meg akarjuk kapni a valós számok számát, ne lőjjük le a programot, hanem használjuk a ctrl+d billentyűkombinációt, ami a program végét jelenti.



A screenshot of a terminal window titled "puskas@puskas-VirtualBox: ~". The window shows the following command-line session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ lex -o szamokszama.c szamokszama.l
puskas@puskas-VirtualBox:~$ gcc -lfl szamokszama.c -o szamokszama
puskas@puskas-VirtualBox:~$ ./szamokszama
dfuiosdifuh5345345.66546h54645634f345f3h457.56
dfuiosdifuh
[Valós szám:5345345.66546 5345345.665460]h
[Valós szám:54645634 54645634.000000]f
[Valós szám:345 345.000000]f
[Valós szám:3 3.000000]h
[Valós szám:457.56 457.560000]
A valós számok száma: 5
puskas@puskas-VirtualBox:~$
```

3.5. l33t.l

Lexelj össze egy l33t cipher-t!

Megoldás forrása:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorial_bhax_textbook/Chomsky/l33t.l

A l33t.l egy olyan program, amely a bekért karakterlánc karaktereit egy előre definiált módon más karakterekre cseréli.

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

Ebben a programban szükségünk lesz a fenti headerere. A "time.h" headerre azért van szükség, mert egy karaktert több módon is átírhatunk, és a lehetőségek közül random szeretnénk választani.

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

struct cipher {
    char c;
    char *leet[4];
```

```
    } 1337d1c7 [] = {  
  
    {'a', {"4", "4", "@", "/-\\"}},  
    {'b', {"b", "8", "|3", "|{}"}},  
    {'c', {"c", "(", "<", "{}"}},  
    {'d', {"d", "|)", "|]", "|{}"}},  
    {'e', {"3", "3", "3", "3"}},  
    {'f', {"f", "|=", "ph", "|#"}},  
    {'g', {"g", "6", "[", "[+"}}},  
    {'h', {"h", "4", "|-", "|-", "|-"}},  
    {'i', {"1", "1", "|", "!"}}},  
    {'j', {"j", "7", "_|", "_/"}}},  
    {'k', {"k", "|<", "1<", "|{"}}},  
    {'l', {"l", "1", "|", "|_"}},  
    {'m', {"m", "44", "(V)", "|\\/|"}},  
    {'n', {"n", "|\\|", "/\\/", "/V"}},  
    {'o', {"0", "0", "()", "[]"}},  
    {'p', {"p", "/o", "|D", "|o"}},  
    {'q', {"q", "9", "O_", "(,)"}}},  
    {'r', {"r", "12", "12", "|2"}},  
    {'s', {"s", "5", "$", "$"}},  
    {'t', {"t", "7", "7", "'|'"}}},  
    {'u', {"u", "|_|", "(_)", "[_]"}},  
    {'v', {"v", "\\", "\\", "\\"}},  
    {'w', {"w", "VV", "\\\\"}, "(/\\)"}},  
    {'x', {"x", "%", ")("}, ")("}},  
    {'y', {"y", "", "", ""}},  
    {'z', {"z", "2", "7_", ">_"}},  
  
    {'0', {"D", "0", "D", "0"}},  
    {'1', {"I", "I", "L", "L"}},  
    {'2', {"Z", "Z", "Z", "e"}},  
    {'3', {"E", "E", "E", "E"}},  
    {'4', {"h", "h", "A", "A"}},  
    {'5', {"S", "S", "S", "S"}},  
    {'6', {"b", "b", "G", "G"}},  
    {'7', {"T", "T", "j", "j"}},  
    {'8', {"X", "X", "X", "X"}},  
    {'9', {"g", "g", "j", "j"}},  
  
};  
  
%}  
%%
```

Ez a rész lesz a definiáló rész a programunkban. A "L337SIZE" konstans értékét a "l337d1c7" tömb méretének és egy darab cipher struktúra méretének hányadosa adja. Deklarálunk egy "cipher" nevű struktúrát, melyben felveszünk egy karaktert, a "char* leet[4]"-ben pedig megmondjuk, hogy a char* egy 4 elemű tömbre mutat, ugyanis egy karaktert egy 4 elemből álló tömb egyik tagjára cserélünk. Majd egy tömbbe egyesével felvesszük a cserálni kívánt karaktereket, ezek mellé pedig a cserelehetőségek kerülnek.

A programot a nevek kivételével ugyanazon parancsok beírásával tudjuk futtatni. Alább láthatjuk, hogy tökéletesen működik.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ lex -o l33t.c l33t.l
puskas@puskas-VirtualBox:~$ gcc -lfl l33t.c -o l33t
puskas@puskas-VirtualBox:~$ ./l33t
Remélem Zidane rendbe tudja szedni a Real Madridot.
r3|\\|él3m z!d4n3 r3ndb3 tu||j4 sz3dn1 4 r34l |||4dr1d0t.
puskas@puskas-VirtualBox:~$
```

```
. {

    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);  
  
}  
%%
```

Ez a blokk lesz programunkban a szabály blokk. A "." karakter/karaktereket helyettesít. Úgy értelmezzük ezt a részt, hogy: ha legalább egy karakter érkezik be, akkor végrehajtja az alatta lévő kód részletet. Felvesszünk egy "found" nevű változót, melynek kezdőértékét 0-ra állítjuk, ez azért lesz felelős a későbbiekben, hogy megnézze, hogy a bekért karakterlánc átírható e. A for ciklusunk végigmegy a "l337d1c7" tömb char típusú elemein (az angol abc betűi és 0-9 számjegyek), a "tolower()" függvény használatával nem jelent gondot programunknak a nagybetűs beírt szöveg sem. Ezután egy random számot generálunk 1 és 100 között, mely segítségével a fent látható módon választunk a "l337d1c7" tömb "leet"-re vonatkozó elemei közül. Majd a break segítségével kilépünk a ciklusból, és vizsgáljuk a következő karaktert. Ezután a "found" értékét 1-re változtatjuk, majd erre nézünk egy vizsgálatot, és ha a bekért karakterlánc karaktereit nem lehet más karakterekkel kicserálni, kiírja az eredetit.

```
int  
main()  
{  
    srand(time(NULL)+getpid());  
    yylex();  
    return 0;  
}
```

A main függvényünkben pedig csak inicializáljuk a random szám generálást, és meghívjuk a lexikális elemzőt.

3.6. A források olvasása

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

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



Bugok

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

i.

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

Amennyiben a SIGINT nem volt figyelmen kívül hagyva, a jelkezelő függvény fogja kezelní.

ii.

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

Ez egy for ciklus, amely a benne lévő utasítást ötször fogja végrehajtani.

iii.

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

Az fentebb látható ciklussal egyezik meg.

iv.

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

Ez a for ciklus egy tömb elemeit fogja megadni oly módon, hogy a tömb aktuális eleme mindenkor nagyobb legyen mint az i jelenlegi értéke. Viszont a tömb első eleme memóriaszemét lesz. Ez azért történik, mert a "tomb[i]=i++" részben előbb használjuk az i-t, majd módosítjuk. Ugyanis ha egy paramétert különböző helyen használunk illetve módosítunk, eredményépp egy ismeretlen kifejezést, memóriaszemet kapunk.

v.

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

Ez a kód hibás, a pointerek miatt az és operátor jobb oldalán nem logikai értéket kapunk.

vi.

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

Itt ismét külön helyen módosítunk és használunk egy paramétert, hibához vezet.

vii.

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

Ez a kiíratási függvény két számot írat ki a standard kimenetre, az egyik szám egy adott szám valamelyen függvény által módosított értéke, a másik pedig maga az adott szám.

viii.

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

Ez a függvény két számot írat ki. Az egyik szám egy valamilyen függvény által módosított szám, mely paraméterül egy "a"-ra mutató értéket kapott. A másik szám pedig maga az "a".

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

Létezik olyan y, amely minden x-re ha x prím akkor x kisebb mint y.

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

Létezik olyan y, amely minden x-re ha y kisebb mint x akkor x nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

Forrás:https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Chomsky/deklaracio.cpp

- egész

```
int alma=15;
```

- egészre mutató mutató

```
int *körte=&alma;
```

- egész referencia

```
int &pumba=alma;
```

- egészek tömbje

```
int tomb[12];
```

- egészek tömbjének referencia (nem az első elemé)

```
int (&ref)[12] = tomb;
```

- egészre mutató mutatók tömbje

```
int *t[12];
```

- egészre mutató mutatót visszaadó függvény

```
int *fg();
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int *(*a)();
```

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

```
int (*l (int k)) (int u, int o)
```

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

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

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

Egy "a" nevű egész típusú változót.

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

Egy "b" nevű pointert, ami az "a"-ra mutat.

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

Egy "r"-re mutató egész típusú változót.

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

Egy egész típusú, 5 elemű "c" nevű tömböt.

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

Egy egész típusú referenciatömböt.

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

Egész típusú mutatók 5 elemű tömbjét.

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

Egy olyan mutatót visszaadó függvényt, amely egész típusú változóra mutat.

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

Egy olyan függvényre mutatót mutatót visszaadó függvényt, amely egész típusú változóra mutat.

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

Egy olyan függvényt, amely egy két egészet kapó függvényre mutatót mutatót ad vissza, és egy egészet kap paraméterül.

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

Egy olyan függvényre mutatót, amely egy két egészet kapó és egy egészet adó függvényre mutatót mutatót ad vissza, és értékül két egészet kap és egy egészet ad vissza.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Caesar/tm.c

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

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

Programunk azzal kezdődik, hogy megadjuk a sorok számát, majd lefoglalunk 8 byte-nyi helyet a memóriában a "tm" számára. Ezután az első byte címét printf függvény használatával kiíratjuk.

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

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

A malloc függvény segítségével 40 byte-nyi (5×8) helyet lefoglal az operációs rendszer a memóriában. A malloc függvény viszont egy void mutatót ad vissza erre a lefoglalt memóriára, ezért a típusát "double **"-ra kényszerítjük. Ha eredményül 0-t kapunk, az azt jelenti, hogy nem tudott kellő mennyiségű helyet lefoglalni, így kilépünk a programból. Ezután a tm-ben a malloc függvény által visszaadott értéket kiíratjuk.

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

```
}
```

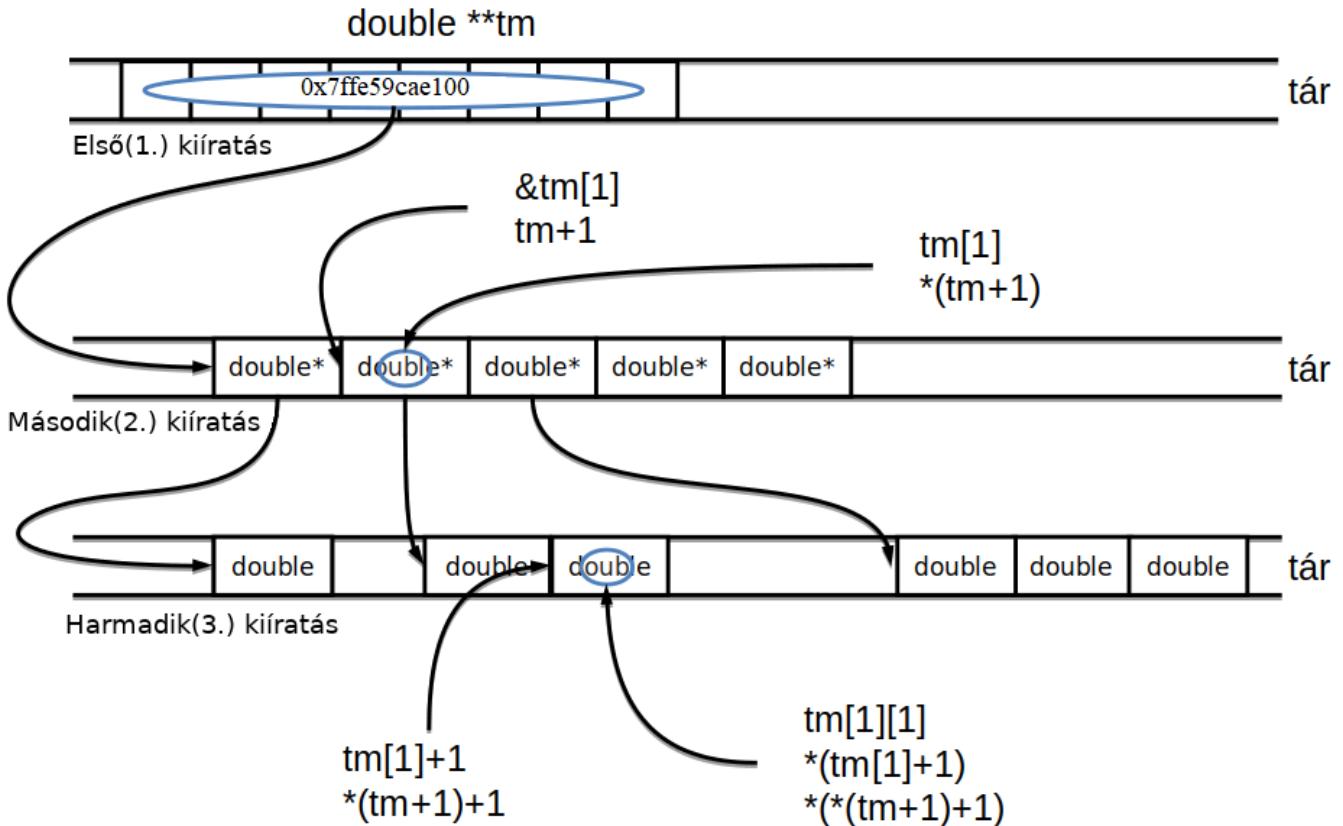
Ezután a malloc függvény használatával soronként lefoglalunk $n*8\text{byte}$ -nyi helyet, és "double*" -ra kénytelen szerítjük a visszakapott értéket. Ha nem bírt lefoglalni annyi memóriát amennyit kellett volna -1-gyel tért vissza, vagyis kilép a program.

```
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;  
*(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;  
}
```

A program több részében a háromszög mátrix elemeit határozzuk meg és íratjuk ki.

```
puskas@puskas-VirtualBox: ~
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ gcc tm.c -o tm
puskas@puskas-VirtualBox:~$ ./tm
(1.)A double** tm által a memóriában elfoglalt hely kezdeti címe:
0x7ffe59cae100
(2.)A double** tm-ben elhelyezett első double*-nak lefoglalt hely kezdeti címe:
0x562d38461670
(3.)Az első double*-ban lefoglalt első double értéknek lefoglalt hely kezdeti címe:
0x562d384616a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
puskas@puskas-VirtualBox:~$ █
```

A program kipróbálása.



A memóriakezelés szemléltetésére készült ábra.

4.2. C EXOR titkosító

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Caesar/e.c

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

Programunkat a szükséges header-ök meghívásával kezdjük.

```
#define MAX_KULCS 100
#define BUFFER_MERET 256
```

Meghatározunk két értéket, amelyet a `MAX_KULCS` és `BUFFER_MERET` helyeken fogunk használni.

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

Main programunk azzal kezdődik, hogy deklarálunk két karakter típusú tömböt, és elemeinek számát beállítjuk a fentebb definiált értékekre. Ezzel a kulcs és a beolvasott bajtok maximális számát határoztuk meg.

```
int kulcs_index = 0;
int olvasott_bajtok = 0;
```

Az olvasott bajtok számát, és a kulcs index tárolására használt változót kinullázzuk.

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

A kulcs mértét úgy kapjuk, hogy megnézzük az első parancssori argumentum hosszát, ugyanis az első parancssori argumentumban adjuk meg a kulcsot. Ezután a kulcs nevű tömbünkbe belemásoljuk az első parancssori argumentum byte-jait, ez maximum MAX_KULCS byte-ot jelenthet.

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

While ciklusunk addig fog futni, amíg a beolvasott bajtok száma el nem éri szöveg méretét, amelyet a read() függvény segítségével állapítunk meg, ugyanis ez a függvény a beolvasott byte-ok számát adja vissza. A for ciklusunkban végigmegyünk a beolvasott byte-okon és elvégezzük a kulcstól függően a titkosítást. A write() függvényünk a titkosított szöveget fogja byteenként (max "olvasott_bajtok" byte-ot fog beolvasni) beleírni a buffer tömbbe.

4.3. Java EXOR titkosító

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Caesar/ExorTitkosito.java

```
public static void ExorTitkosito(String kulcsSzoveg, java.io.InputStream ←
    bejovoCsatorna, java.io.OutputStream kimenetCsatorna) throws java.io. ←
    IOException {
```

Javaban megírt programunk kezdetén megadjuk, hogy szükségünk lesz egy kulcsra, egy bemeneti fájlra/ szöbegre és egy kimeneti fájlra.

```
while(olvasottBajtok!= -1) {  
  
    for(int i=0; i<olvasottBajtok; ++i) {  
  
        buffer[i] = (byte) (buffer[i] ^ kulcs[kulcsIndex]);  
        kulcsIndex = (kulcsIndex+1) % kulcs.length;  
  
    }  
}
```

While ciklusunk itt is addig megy, míg be nem olvastuk az összes byte-ot. Ugyanúgy betöljük az összexorozott byte-okat a buffer tömbbe. Az exor művelet itt is a kulcs segítségével történik.

```
kimenoCsatorna.write(buffer, 0, olvasottBajtok);  
olvasottBajtok=bejovoCsatorna.read(buffer);  
}  
  
}
```

Itt meghatározzuk, hogy a buffer tömb elemeit (olvasottBajtok-ig) írja bele egy kimeneti fájlba. Az olvasottBajtok-at pedig a bejovoCsatorna-l olvassuk be a buffer-be.

```
public static void main(String[] args) {  
  
    try {  
  
        ExorTitkosito(args[0], System.in, System.out);  
  
    } catch(java.io.IOException e) {  
  
        e.printStackTrace();  
  
    }  
}
```

Itt pedig megnézzük, hogy a megfelelő argumentumokat kapott-e a program, ha nem, jelezük a felhasználó felé.

4.4. C EXOR törő

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Caesar/t.c

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

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

A programunk bizonyos részek definiálásával, és a szükséges header-ök meghívásával kezdődik.

```
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;
}
```

Ez a függvény a szavak átlagos hosszát adja vissza értékkül. Ezt úgy kapjuk, hogy a szavak számának felveszünk egy változót, amit kezdetben kinullázunk. Majd a beolvastott karaktereken egyesével egy for ciklus segítségével végig megyünk, és ha szóközt találtunk, a szavak számához hozzáadunk egyet. Ezután a szavak átlagos hosszát a bekért szöveg méretének és a szavak számának hányadosa adja.

```
int
tiszta_lehet (const char *titkos, int titkos_meret)
{

    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");

}
```

Következő függvényünk megnézi, hogy az adott szövegen van-e titkosítatlan szövegrész. Ha az átlagos szóhossz 6 és 9 közé esik, és talál benne "hogy", "nem", "az", "ha" szavakat, akkor visszaad egy true értéket.

```
void
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{

    int kulcs_index = 0;

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

        titkos[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
```

```
    }  
  
}
```

Az "exor" függvényben történik a szöveg és a kulcs összexorozása, majd a kulcson lépkedünk.

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

Az "exor_tores"-ben meghívjuk a fentebb definiált "exor" függvényünket, majd a "tiszta_lehet" függvényt hívjuk meg a titkos szövegre, és ennek a függvénynek az értékét fogjuk visszaadni.

```
int  
main (void)  
{  
  
    char kulcs[KULCS_MERET];  
    char titkos[MAX_TITKOS];  
    char *p = titkos;  
    int olvasott_bajtok;  
  
    // titkos fajt berantasa  
    while ((olvasott_bajtok =  
            read (0, (void *) p,  
                  (p - titkos + OLVASAS_BUFFER <  
                   MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))  
        p += olvasott_bajtok;
```

A while ciklusunkban olvassuk be a titkos fájlt byteonként.

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

A titkos bufferben a megmaradt helyeket kinullázzuk.

```
for (int ii = '0'; ii <= '9'; ++ii)  
    for (int ji = '0'; ji <= '9'; ++ji)  
        for (int ki = '0'; ki <= '9'; ++ki)  
    for (int li = '0'; li <= '9'; ++li)  
        for (int mi = '0'; mi <= '9'; ++mi)  
            for (int ni = '0'; ni <= '9'; ++ni)  
                for (int oi = '0'; oi <= '9'; ++oi)  
        for (int pi = '0'; pi <= '9'; ++pi)  
    {
```

```
kulcs[0] = ii;
kulcs[1] = ji;
kulcs[2] = ki;
kulcs[3] = li;
kulcs[4] = mi;
kulcs[5] = ni;
kulcs[6] = oi;
kulcs[7] = pi;
```

Egymásba ágyazott for ciklusok segítségével az összes lehetséges kulcsot előállítjuk.

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

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

return 0;
}
```

A feltételes utasításunkban igénybe vesszük az "exor_tores" függvényünket, mellyel a lehetséges kulcsok mindegyikét kipróbáljuk, és megvizsgáljuk, hogy valamelyik kulccsal visszakapjuk e a tiszta szöveget, ha igen, ezt a kulcsot és a tiszta szöveget kiíratjuk.

4.5. Neurális OR, AND és EXOR kapu

```
library(neuralnet)

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

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

Programunk azzal keződik, hogy a neuralnet nevű csomagot meghívjuk, majd 3 vektort hozunk létre. Ennek elemei azért helyezkednek el ilyen sorrendben, mert az OR vektorban az a1 és a2 értékeinek logikai vagy alapján kiértékelte értéke látható. (0 vagy 0=0, 1 vagy 0=1 stb stb) Majd a következő sorban ebből a 3 vektorból adatot csinálunk.

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

A "neuralnet" függvény használatával állítjuk elő a neurális hálót, amely beállítja a súlyokat. A súlyokat matematikai műveletek segítségével határoztuk meg. Ezáltal a gép "megtanulta" a logikai vagy művelet értékét.

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

Majd a compute parancssal beadjuk neki az értékeket.

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

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

Következő lépésekben a logikai és műveletre fogjuk megtanítani a gépet. Ehhez ismét előre meghatározzuk a művelet értékeit, majd adatot képzünk belőle.

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

Ahogyan már pár sorral fentebb, ismét itt történik a "tanítás".

```
plot(nn.operand)
```

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

Itt pedig a rajzolás és az értékek átadása.

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

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

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

plot(nn.exor)

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

Ebben a részben pedig az exor műveletre szeretnénk megtanítani a gépet. Ez viszont ezzel a módszerrel nem sikerülhet. Ugyanis a súlyok értéke rendre 0.5 körüli értékek, így nem kapunk pontos megközelítést, hogy arra az értékre 0 vagy 1 értéket kell visszaadnia.

```

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=2, linear.output=FALSE, ←
  stepmax = 1e+07, threshold = 0.000001)

```

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

Ezzel a módszerrel viszont képesek vagyunk exor műveletre megtanítani a gépet. Ebben a módszerben 2 rejtett neuron segítségével tanítjuk meg a gépet az exor műveletre.

4.6. Hiba-visszaterjesztéses perceptron

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Caesar/main.cpp

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

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

A szükséges header-ök importálása után a main függvényünket egy kép létrehozásával kezdjük, melynek a méretét az első parancssori argumentum értékével tesszük egyenlővé. Ezután felveszünk egy változót, amely ennek a képnak a méretét fogja tárolni, ezt úgy kapjuk meg, hogy a kép szélességét megszorozzuk a kép magasságával. Majd létrehozunk egy perceptron-t, amelyet 3 rétegűre állítunk be, az első réteg méretét a kép valódi mérete adja, a második réteg méretét 256-ra, a harmadik réteg méretét pedig 1-re állítjuk, ezt a számot fogjuk kiszámolni.

```
double* image = new double[size];
```

Az "image" nevű tömböt a kép pixeleinek hozzuk létre. A kép pixeleinek száma megegyezik a kép méretével, ezért a tömb hossza a kép mérete lesz.

```
for(int i{0};i<png_image.get_width();++i)
    for(int j{0};j<png_image.get_height();++j)
        image[i*png_image.get_width()+j] = png_image[i][j].red;
```

Két for ciklus segítségével végigmegyünk a pixeleken, majd mindegy egyes piros pixelt beletöltünk az "image" tömbünkbe.

```
double value = (*p)(image); //A "p" mutató helyen tárolt perceptron →
                            objektumot ad vissza
```

A "value" változó értéket úgy kapjuk meg, hogy a *p perceptron függvényként meghívjuk erre az "image" tömbre. Ezzel betöltöttük a tömbünket a neurális hálóba, és egy -1 és 1 közötti értéket fogunk visszakapni súlyként.

```
std::cout << value << std::endl;

delete p;
delete [] image;
}
```

Ezután kiíratjuk a neurális háló által visszaadott súlyt a standard kimenetre, majd felszabadítjük a p perceptronról és az image tömöböt.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Mandelbrot/mandelbrot.cpp

A mandelbrot halmaz azon komplex számokból áll, melyek azon rekurzív sorozata amelynek minden elemét az előző elem négyzetének és az első elemnek az összege adja, nem a végtelenbe tart, tehát felülről határos. Ha ezt a halmazt ábrázoljuk, egy fraktálalakzatot kapunk.

```
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
#define ITER_HAT 32000
```

Programunk a szükséges header-ök és két konstans definiálásával kezdődik.

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

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

Ezt egy void típusú függvény deklarálása követi, amely egy kétdimenziós tömböt fog visszaadni, egy 600x600-as méretűt, majd megnézzük a program futtatásához hány óraütésre van szükség.

```
// 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;
```

Felvesszük a számításhoz szükséges adatokat, megadjuk a kép szélességét és magasságát, valamint deklarálunk az iterációs határ meghatározására egy változót, ami ahhoz fog kelleni, hogy meddig fussen a pár sorral lejjebb található while ciklus. Felveszünk a komplex számok miatt valós típusú változókat, külön a valós és külön a képzetes részeknek, majd az iterációk számához egy gyűjtőváltozót is deklarálunk 0 kezdőértékkel.

```
// Végigzongorázzuk a szélesség x magasság rácsot:
for (int j = 0; j < magassag; ++j)
{
    //sor = j;
    for (int k = 0; k < szelesseg; ++k)
    {
        // c = (reC, imC) a rács csomópontjainak
        // megfelelő komplex szám
        reC = a + k * dx;
        imC = d - j * dy;
        // z_0 = 0 = (reZ, imZ)
        reZ = 0;
        imZ = 0;
        iteracio = 0;
        // z_{n+1} = z_n * z_n + c iterációk
        // számítása, amíg |z_n| < 2 vagy még
        // nem értük el a 255 iterációt, ha
        // viszont elértek, akkor úgy vesszük,
        // hogy a kiinduláci c komplex számra
        // az iteráció konvergens, azaz a c a
        // Mandelbrot halmaz eleme
```

Két egymásba ágyazott for ciklussal végigmegyünk a kép pixelein, és előállítjuk a valós és imaginárius részeit a komplex számoknak. Az első elem tehát a 0 lesz. A második elemet úgy kapjuk, hogy az első elem négyzetéhez hozzáadjuk a fentebb előállított c számot.

```
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;

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

While ciklusunk addig fog futni, míg a fentebbi módon előállított komplex szám négyzete el nem éri a 4-et, és az iterációk száma meg nem haladja 32000-et. A ciklusban előállítjuk a következő komplex számot, majd az eredeti értékét felülírjük az újonan kapott komplex szám értékével, és az iterációk számát növeljük.

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

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

}

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

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

Main függvényünkben megvizsgáljuk, hogy elegendő argumentum áll-e rendelkezésünkre, ha nem, kiíratjuk a felhasználó számára, hogy hogyan lehet lefuttatni a programot.

```
int kepadat [MERET] [MERET];

mandel (kepadat);

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

Ezután felveszünk egy két dimenziós tömböt, majd meghívjuk rá a "mandel" nevű függvényünket, majd előállítjuk a képet.

```
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 -->
                                         ) );
    }
}
```

```
}
```

A pixelek rgb kódját pedig a fentebb látható módon állítjuk elő.

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

Parancssori argumentum első elemének a kép nevét kaptuk, tehát a képet ilyen néven kell előállítanunk, majd felvilágosítjuk a felhasználót, hogy az általa választott néven elkészült a kép.

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

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Mandelbrot/m_komplex.cpp

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

Meghívjuk a szükséges header-öket, amint látható, itt a complex nevű header is meghívásra került, ami lehetővé teszi a komplex számok használatát a forráskódban, így egyszerűbb dolgunk van.

```
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;
```

Ebben a programban függvények használata helyett a main-ben írunk meg minden. Az előző forráskódhoz hasonlóan itt is deklaráljuk a szelesség, magasság, iterációs határ és az a,b,c,d változót.

```
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] );
}
```

Megvizsáljuk, hogy 9 parancsosri argumentumot kaptunk e, ha igen, a "szelesseg" változóba az "atoi" függvény segítségével beletöltyük a parancssori argumentum második elemét. Ugyanezt tesszük a "magassag" és az "iteraciosHatar" változóval is, ezek rendre a 3. és 4. parancssori argumentumot kapják értékül. Ugyanez történik az a,b,c,d változókkal is, de mivel ezek double típusúak lettek deklarálva, ezek az "atof" függvény használatával kapják meg rendre az 5.,6.,7.,8. argumentumot.

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

Ellenkező esetben tájékoztatjuk a felhasználót a program futtatásának módjáról.

```
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";
```

Előállítjuk a képet, a számításhoz szükséges változókat deklaráljuk, majd felvilágosítjuk a felhasználót a számítás állapotáról.

```
// 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;
```

Végigmegyünk a pixeleken, és előállítjuk a komplex számot, ami mindenhol hozzá fog adódni a sorozat aktuális eleméhez, majd a sorozat első elemét 0 kezdőértékkel deklaráljuk komplex típusú változóként.

```
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 ) );
}
```

While ciklusunk itt is addig fut, míg ez a komplex szám abszolut értéke el nem éri a 4-et, illetve az iterációk száma el nem éri az iterációs határt. A ciklusban vesszük ennek a számnak a négyzetét, és hozzáadjuk a fentebb látható módon előállított "c" komplex számot és növeljük az iterációk számát. A kép pixeleinek rgb kódját az "iteracio" változó értéke alapján beállítjuk.

```
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;

}
```

A számítás állapotát %-ban kiírjuk a felhasználó számára. Ez az "std::flush" miatt folyamatosan frissülni fog. Ezután a parancsosri argumentum első elemeként kapott néven létrehozzuk ezt a képet, és tájékoztatjuk a felhasználót, hogy az általa kívánt kép elkészült.

5.3. Biomorfok

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

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

A biomorf algoritmus 1986-ban jött létre a Pickover által. Pickover egy programot szánt írni, amely a Julia halmaz elemeit rajzolja ki. Az algoritmusban viszont elkövetett egy hibát, így a program biomorf-hoz hasonlító alakzatokat rajzolt ki. A forráskód tehát nem sok mindenben különbözik a feljebb elemzett mandelbrot halmazt kirajzoló program forráskódjától. A program ugyanúgy végigmegy a rácson, viszont a komplex számot egy másfajta módon állította elő. Több ilyen függvény is létezik, ezek (és az ezekhez szükséges összes paraméter) megtalálható itt: https://www.emis.de/journals/TJNSA/includes/files/articles/Vol19_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf

```
int szelessseg = 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;
```

Két újabb váltózóval bővült a deklaráció, egy valós számmal. Ezek a fent említett új paraméterek, melyeket az oldalon minden függvényhez táblázatba foglalva láthatunk.

```
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] );
```

Parancssori argumentumaink száma tehát 3-mal bővült a korábbihoz képest. Ez a 3 argumentum fogja adni a reC, imC, R értékeit.

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

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

Meghatározunk a reC és imC alapján egy komplex számot.

```
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;
    }
}
```

Eddigi while ciklusunk a fentebb látható for ciklussá vált. Ez a cílus addig megy, míg el nem éri az iterációs határt. A "z_n" komplex számot a fentebb látható módon határoztuk meg. A feltételes utasításban látható a Pickover által elkövetett számítási hiba. Eddig ugyanis a komplex szám hosszára tettünk vizsgálatot, valamint hogy az iteráció eléri e az iterációs határt. Ebben a biomorfos változatban Pickover a valós számunk valós és imaginárius részére tett vizsgálatot.

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

Programunk még a pixelek színezésében tér el a mandelbrot-os programtól.

5.4. 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!

Forrás: https://gitlab.com/puskasmate/bhax/tree/master/thematic_tutorials/bhax_textbook/Mandelbrot/Nagyito

frakablak.cpp

```
#include "frakablak.h"

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

    int magassag = (int)(szelesseg * ((d-c) / (b-a)));

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

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

}

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

void FrakAblak::paintEvent(QPaintEvent*) {
    QPainter qpainter(this);
    qpainter.drawImage(0, 0, *fraktal);
    qpainter.end();
}

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

```
    255-sor[i] % 64,  
    255-sor[i] % 16 );  
  
    fraktal->setPixel(i, magassag, szin);  
}  
update();  
}
```

frakszal.h

```
#ifndef FRAKSZAL_H  
#define FRAKSZAL_H  
  
#include <QThread>  
#include <QImage>  
#include "frakablak.h"  
  
class FrakAblak;  
  
class FrakSzal : public QThread  
{  
    Q_OBJECT  
  
public:  
    FrakSzal(double a, double b, double c, double d,  
              int szelessseg, int magassag, int iteraciosHatar, FrakAblak * frakAblak);  
    ~FrakSzal();  
    void run();  
  
protected:  
    // A komplex sík vizsgált tartománya [a,b]x[c,d].  
    double a, b, c, d;  
    // A komplex sík vizsgált tartományára feszített  
    // háló szélessége és magassága.  
    int szelessseg, magassag;  
    // Max. hány lépésig vizsgáljuk a  $z_{n+1} = z_n \cdot z_n + c$  iterációt?  
    // (tk. most a nagyítási pontosság)  
    int iteraciosHatar;  
  
    FrakAblak* frakAblak;  
    int* egySor;  
};  
  
#endif // FRAKSZAL_H
```

frakszal.cpp

```
#include "frakszal.h"
```

```
FrakSzal::FrakSzal(double a, double b, double c, double d,
                     int szelessseg, int magassag, int iteraciosHatar, ←
                     FrakAblak *frakAblak)
{
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->szelessseg = szelessseg;
    this->iteraciosHatar = iteraciosHatar;
    this->frakAblak = frakAblak;
    this->magassag = magassag;

    egySor = new int[szelessseg];
}

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

// A szál kódját a Javát tanítokhoz írt Java kódomból vettetem át
// http://www.tankonyvtar.hu/informatika/javat-tanitok-2-2-080904-1
// mivel itt az algoritmust is leírtam/lerajzoltam, így meghagytam
// a kommenteket, hogy a hallgató könnyen hozzáolvashassa az "elméletet",
// ha érdekli.
void FrakSzal::run()
{
    // A [a,b]x[c,d] tartományon milyen sűrű a
    // megadott szélesség, magasság háló:
    double dx = (b-a)/szelessseg;
    double dy = (d-c)/magassag;
    double reC, imC, rez, imZ, ujrez, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság hálót:
    for(int j=0; j<magassag; ++j) {
        //sor = j;
        for(int k=0; k<szelessseg; ++k) {
            // c = (reC, imC) a háló rácspontjainak
            // megfelelő komplex szám
            reC = a+k*dx;
            imC = d-j*dy;
            // z_0 = 0 = (rez, imZ)
            rez = 0;
            imZ = 0;
            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 kiinduláci 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;  
  
}  
// ha a < 4 feltétel nem teljesült és a  
// iteráció < iterációsHatár sérülésével lépett ki, azaz  
// feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c  
// sorozat konvergens, azaz iteráció = iterációsHatár  
// ekkor az iteráció %= 256 egyenlő 255, mert az esetleges  
// nagyítások során az iteráció = valahány * 256 + 255  
  
//a színezést viszont már majd a FrakAblak osztályban lesz  
egySor[k] = iteracio;  
}  
// Ábrázolásra átadjuk a kiszámolt sort a FrakAblak-nak.  
frakAblak->vissza(j, egySor, szelesseg, iteraciosHatar);  
}  
}
```

frakablak.h

```
#ifndef FRAKABLAK_H  
#define FRAKABLAK_H  
  
#include <QMainWindow>  
#include <QImage>  
#include <QPainter>  
#include "frakszal.h"  
  
class FrakSzal;  
  
class FrakAblak : public QMainWindow  
{  
    Q_OBJECT  
  
public:  
    FrakAblak(double a = -2.0, double b = .7, double c = -1.35,  
              double d = 1.35, int szelesseg = 600,  
              int iteraciosHatar = 255, QWidget *parent = 0);  
    ~FrakAblak();
```

```
void vissza(int magassag , int * sor, int meret, int hatar) ;  
  
protected:  
    void paintEvent (QPaintEvent*);  
  
private:  
    QImage* fraktal;  
    FrakSzal* mandelbrot;  
};  
  
#endif // FRAKABLAK_H
```

main.cpp

```
#include <QApplication>  
#include "frakablak.h"  
  
int main(int argc, char *argv[]){  
    QApplication a(argc, argv);  
    // További adatokat olvashatsz le innen:  
    // http://www.tankonyvtar.hu/informatika/javat-tanitok-2-3-080904  
    FrakAblak w1,  
    w2(-.08292191725019529, -.082921917244591272,  
        -.9662079988595939, -.9662079988551173, 600, 3000),  
    w3(-.08292191724880625, -.0829219172470933,  
        -.9662079988581493, -.9662079988563615, 600, 4000),  
    w4(.14388310361318304, .14388310362702217,  
        .6523089200729396, .6523089200854384, 600, 38656);  
    w1.show();  
    w2.show();  
    w3.show();  
    w4.show();  
  
    return a.exec();  
}
```

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.

Megvalósítás C++-ban:

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

class PolarGen
{
public:
    PolarGen ()
    {
        nincsTarolt = true;
        std::srand (std::time (NULL));
    }
    ~PolarGen ()
    {
    }
    double kovetkezo ()
    {
        if (nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do
            {
                u1 = std::rand () / (RAND_MAX + 1.0);
                u2 = std::rand () / (RAND_MAX + 1.0);
                v1 = 2 * u1 - 1;
```

```
v2 = 2 * u2 - 1;
w = v1 * v1 + v2 * v2;
}
while (w > 1);

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

tarolt = r * v2;
nincsTarolt = !nincsTarolt;

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

private:
    bool nincsTarolt;
    double tarolt;
};

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

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

    return 0;
}
```

Megvalósítás Java-ban:

```
import java.util.Random;

public class PolarGen
{
    private double tarolt;
    private boolean nincsTarolt;
    private Random r;
    private int RAND_MAX;

    public PolarGen()
```

```
{  
    nincsTarolt = true;  
    r = new Random();  
    r.setSeed(20);  
    this.RAND_MAX=100;  
}  
public PolarGen(Integer RAND_MAX)  
{  
    nincsTarolt = true;  
    r = new Random();  
    r.setSeed(20);  
    this.RAND_MAX=RAND_MAX;  
}  
  
public double kovetkezo()  
{  
    if (nincsTarolt)  
    {  
  
        double u1, u2, v1, v2, w;  
        int i=0;  
        do  
        {  
            u1 = r.nextInt() / (RAND_MAX + 1.0);  
            u2 = r.nextInt() / (RAND_MAX + 1.0);  
            v1 = 2 * u1 - 1;  
            v2 = 2 * u2 - 1;  
            w = v1 * v1 + v2 * v2;  
        }  
        while (w > 1 && i++ < 40000000);  
        double r = Math.sqrt ((2 * Math.log10(w)) / w);  
        tarolt = r * v2;  
        nincsTarolt = !nincsTarolt;  
        return r * v1;  
    }  
    else  
    {  
        nincsTarolt = !nincsTarolt;  
        return tarolt;  
    }  
}
```

6.2. LZW

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

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/

bhax_textbook/Welch/binfa.c

```
//A program forrása: https://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

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

//Készítünk egy binfa struktúrát ami áll ely értékből és egy bal és jobb ←
//oldali pointerból.

typedef struct binfa
{
    int ertek;
    struct binfa *bal_nulla;
    struct binfa *jobb_egy;

//Létrehozzuk a BINFA és BINFA_PTR típusokat.
} BINFA, *BINFA_PTR;

//Meglézzük hogy ha üres a BINFA_PTR akkor kilépünk.

BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

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

//Deklaráljuk a kiir és szabadít függvényeket.

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

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

    //A gyökérre példányosítunk egy elemet ami a '/' lesz ezután ráállítjuk a ←
    //mutatót a gyökérre

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

```
//Olvassuk a bemenetet a kimentetre (standart output) pedig írunk.
while (read (0, (void *) &b, 1))
{
    write (1, &b, 1);
    //Mégnezzük 0-t kel-e tenni a fába
    if (b == '0')
    {
        //Megnézzük az adott csomópontnak van-e nullás gyereke. Ha nincs ←
        //csinálunk egyet, majd ráállítjuk a mutatót, ezt követően pedig a ←
        //gyökérre az algoritmus miatt.
        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
        {
            //Ha volt nullás eleme akkor csak ráállítjuk a mutatót.
            fa = fa->bal_nulla;
        }
    }
    //Ez megcsináljuk egyes gyerekkel is.
    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;
        }
    }
}
//Kiirathuk a gyokeret a mélyésggel együtt.
printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d", max_melyseg);
szabadit (gyoker);
}

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

```
// A kiír függvényben növeljük megmérjük a mélységet és kiírjuk a fát ←
// inorder feldolgozás szerint, tehát először az egyes elemeket, majd a ←
// gyökeret utoljára pedig a nullás elemeket.
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);
        kiir (elem->bal nulla);
        --melyseg;
    }
}
// A szabadít függvényben felszabadítjuk az adott elemet a gyermekeivel ←
// kezdve.
void
szabadit (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        szabadit (elem->jobb_egy);
        szabadit (elem->bal nulla);
        free (elem);
    }
}
```

Az LZWBInFa osztály a kapott bemeneti fájlhoz tartozó bináris fát építi fel. A fába 0-s és 1-s karaktereket rakhatunk bele. A faépítésnek megvan a maga folyamata:

Fentebb említve lett, hogy a fa (a gyökerén kívül) 0-s és 1-s karakterekből állhat, tehát a fa minden csomópontjának 2 gyermeke lehet.

A bemenetről olvassuk a karaktereket, és el kell döntenünk, hogy az 0-s vagy 1-s értékkel kerüljön a fába.

Ha a beolvasott elem 0, az a bal oldali ág lesz, viszont ha már létezik ilyen, egy ággal lejjebb ugrunk, és az ott bekért bemenetre újra megtesszük ezt a vizsgálatot.

Előfrodulhat, hogy olyan csomópontron állunk, amely bal oldali eleme 0 értékkel rendelkezik, és a beolvasott bemenet szintén 0 értéket kap, ekkor visszaugrunk a gyökérre.

Ha a beolvasott bájt nem 0, azt automatikusan 1-s értékkel rakjuk be a fába.

6.3. Fabejárás

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Welch/bejarasok.cpp

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

Preorder bejárás: Először feldolgozzuk a soron következő elemet, majd bejárjuk a fa bal oldali részfáját, utánna következik a jobb oldali részfa.

Postorder bejárás: Először bejárjuk a bal oldali részfát, majd jöhét a jobb oldali, és végül feldolgozzuk az aktuális elemet.

```
// z3a7.cpp

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

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

    ~LZWBinFa() {
        szabadit(gyoker.egyesGyermek());
        szabadit(gyoker.nullasGyermek());
    }

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

    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 setBejaras(char bejar) {
    this->bejar = bejar;
}

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

private:
char bejar;

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

Csomopont *fa;
int melyseg, atlagosszeg, atlagdb;
```

```
double szorasosszeg;

//nocopy
LZWBInFa(const LZWBInFa &);
LZWBInFa & operator=(const LZWBInFa &);

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

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

    else {
        if (elem != NULL) {
            ++melyseg;
            kiir(elem->egyesGyermek(), 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->egyesGyermek());
        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->egyesGyermek());
rmelyseg(elem->>nullasGyermek());
--melyseg;
}
}

void LZWBInFa::ratlag(Csomopont * elem)
{
    if (elem != NULL) {
        ++melyseg;
        ratlag(elem->egyesGyermek());
        ratlag(elem->>nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL
            && elem->>nullasGyermek() == NULL) {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

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

void usage(void)
{
    std::cout << "Usage: lzwtree [infile] -o [outfile] [o/r/i]" << std::endl;
}

int main(int argc, char *argv[])
{
    // 4 -> 5 mert a bejárás is rögzítjük
    if (argc != 5) {
        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 létezik..." << std::endl;
    usage();
    return -3;
}

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

// r betű: preorder bejáras
// o betű: postorder bejáras
char bejar = *((++argv)[0];
// std::cout << bejar;

unsigned char b;
LZWBinFa binFa;

binFa.setBejaras(bejar);

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) { // > karakter
        kommentben = true;
        continue;
    }

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

    if (kommentben)
        continue;

    if (b == 0x4e) // N betű
        continue;
```

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

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

kifile.close();
beFile.close();

return 0;
}
```

Az előző forráskódot kiegészítettük 2 függvényel, melyek a preorder és posztorder bejárásokhoz szükségesek. Ezután az argumentumok száma immár 2 kell hogy legyen, erre rávizsgálunk, és a megfelelő bejárást hívjuk meg. Amennyiben nem kaptunk elegendő argumentumot, vagy hibás argumentumokat kaptunk, kiíratjuk a felhasználó számára, hogy hogyan lehet futtatni a programot.

6.4. Tag a gyökér

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Welch/z3a7.cpp

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

```
#include <iostream> // mert olvassuk a std::cin, írjuk a std::cout ←
                  csatornákat
#include <cmath>   // mert vonunk gyököt a szóráshoz: std::sqrt
#include <fstream> // fájlból olvasunk, írunk majd

class LZWBinFa
{
public:

    LZWBinFa () :fa (&gyoker)
    {
    }
    ~LZWBinFa ()
    {
```

```
    szabadit (gyoker.egyesGyermek ());
    szabadit (gyoker.nullasGyermek ());
}

void operator<< (char b)
{
    // Mit kell betenni éppen, '0'-t?
    if (b == '0')
    {
        /* Van '0'-s gyermeke az aktuális csomópontnak?
        megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
        if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
        {
            // elkészítjük, azaz páldányosítunk a '0' betű akt. ←
            // parammá
            Csomopont *uj = new Csomopont ('0');
            // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
            // jegyezze már be magának, hogy nullás gyereke mostantól ←
            // van
            // küldjük is Neki a gyerek címét:
            fa->ujNullasGyermek (uj);
            // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
            fa = &gyoker;
        }
        else // ha van, arra rálépünk
        {
            // azaz a "fa" pointer már majd a szóban forgó gyermekre ←
            // mutat:
            fa = fa->nullasGyermek ();
        }
    }
    // Mit kell betenni éppen, vagy '1'-et?
    else
    {
        if (!fa->egyesGyermek ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyesGyermek (uj);
            fa = &gyoker;
        }
        else
        {
            fa = fa->egyesGyermek ();
        }
    }
}

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 *egyesGyermek () const
    {
        return jobbEgy;
    }

    void ujNullasGyermek (Csmopont * gy)
    {
        balNulla = gy;
    }

    void ujEgyesGyermek (Csmopont * gy)
```

```
{  
    jobbEgy = gy;  
}  
  
char getBetu () const  
{  
    return betu;  
}  
  
private:  
  
    char betu;  
  
    Csomopont *balNulla;  
    Csomopont *jobbEgy;  
  
    Csomopont (const Csomopont &); //másoló konstruktor  
    Csomopont & operator= (const Csomopont &);  
};  
  
Csomopont *fa;  
  
int melyseg, atlagosszeg, atlagdb;  
double szorasosszeg;  
  
LZWBinFa (const LZWBinFa &);  
LZWBinFa & operator= (const LZWBinFa &);  
  
void kiir (Csmopont * elem, std::ostream & os)  
{  
  
    if (elem != NULL)  
    {  
        ++melyseg;  
        kiir (elem->egyesGyermek (), os);  
        // ez a postorder bejáráshoz képest  
        // 1-el nagyobb mélység, ezért -1  
        for (int i = 0; i < melyseg; ++i)  
            os << "---";  
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;  
        kiir (elem->>nullasGyermek (), os);  
        --melyseg;  
    }  
}  
void szabadit (Csmopont * elem)  
{
```

```
if (elem != NULL)
{
    szabadit (elem->egyesGyermek ());
    szabadit (elem->>nullasGyermek ());

    delete elem;
}
}

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

void rmelyseg (Csmopont * elem);
void ratlag (Csmopont * elem);
void rszoras (Csmopont * 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->egyesGyermek ());
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        rmelyseg (elem->>nullasGyermek ());
        --melyseg;
    }
}

void
LZWBinFa::ratlag (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyesGyermek ());
        ratlag (elem->>nullasGyermek ());
        --melyseg;
        if (elem->egyesGyermek () == NULL && elem->>nullasGyermek () == NULL -->
            )
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}

void
LZWBinFa::rszoras (Csomopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyesGyermek ());
        rszoras (elem->>nullasGyermek ());
        --melyseg;
        if (elem->egyesGyermek () == 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[])  
{  
  
    // A kiírás szerint ./lzwtree in_file -o out_file alakra kell mennie, ↵  
    // ez 4 db arg:  
    if (argc != 4)  
    {  
        // Ha nem annyit kapott a program, akkor felhomályosítjuk erről a ↵  
        // júzetről:  
        usage ();  
        // És jelezzük az operációs rendszer felé, hogy valami gáz volt...  
        return -1;  
    }  
  
    // "Megjegyezzük" a bemenő fájl nevét  
    char *inFile = *++argv;  
  
    // A -o kapcsoló jön?  
    if (*(*++argv) + 1) != 'o'  
    {  
        usage ();  
        return -2;  
    }  
  
    std::fstream beFile (inFile, std::ios_base::in);  
  
    if (!beFile)  
    {  
        std::cout << inFile << " nem létezik..." << std::endl;  
        usage ();  
        return -3;  
    }
```

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

unsigned char b;      // ide olvassuk majd a bejövő fájl bájtjait
LZWBinFa binFa;     // s nyomjuk majd be az LZW fa objektumunkba

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)
    {
        // > karakter
        kommentben = true;
        continue;
    }

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

    if (kommentben)
        continue;

    if (b == 0x4e)      // N betű
        continue;

for (int i = 0; i < 8; ++i)
{
    if (b & 0x80)
        // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW ←
        // fa objektumunkba
        binFa << '1';
    else
        // különben meg a '0' betűt:
        binFa << '0';
    b <<= 1;
}

}
```

```
kiFile << binFa;

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

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

return 0;
}
```

6.5. Mutató a gyökér

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

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Welch/binfa_mutato_gyoker.cpp

```
#include <iostream>    // mert olvassuk a std::cin, írjuk a std::cout    ←
    csatornákat
#include <cmath>      // mert vonunk gyököt a szóráshoz: std::sqrt
#include <fstream>    // fájlból olvasunk, írunk majd

class LZWBinFa
{
public:

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

    void operator<< (char b)
    {
        // Mit kell betenni éppen, '0'-t?
        if (b == '0')
```

```
{  
    /* Van '0'-s gyermeké az aktuális csomópontnak?  
    megkérdezzük Tőle, a "fa" mutató éppen reá mutat */  
    if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk  
    {  
        // elkészítjük, azaz páldányosítunk a '0' betű akt. ↵  
        // parammal  
        Csomopont *uj = new Csomopont ('0');  
        // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy  
        // jegyezze már be magának, hogy nullás gyereke mostantól ↵  
        // van  
        // küldjük is Neki a gyerek címét:  
        fa->ujNullasGyermek (uj);  
        // és visszaállunk a gyökérre (mert ezt diktálja az alg.)  
        fa = gyoker;  
    }  
    else // ha van, arra rálépünk  
    {  
        // azaz a "fa" pointer már majd a szóban forgó gyermekre ↵  
        // mutat:  
        fa = fa->nullasGyermek ();  
    }  
}  
// Mit kell betenni éppen, vagy '1'-et?  
else  
{  
    if (!fa->egyesGyermek ())  
    {  
        Csomopont *uj = new Csomopont ('1');  
        fa->ujEgyesGyermek (uj);  
        fa = gyoker;  
    }  
    else  
{  
        fa = fa->egyesGyermek ();  
    }  
}  
}  
  
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 *egyesGyermek () const
    {
        return jobbEgy;
    }

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

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

    char getBetu () const
    {
        return betu;
    }
```

```
private:

    char betu;

    Csomopont *balNulla;
    Csomopont *jobbEgy;

    Csomopont (const Csomopont &); //másoló konstruktor
    Csomopont & operator= (const Csomopont &);

};

Csomopont *fa;

int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;

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

void kiir (Csmopont * elem, std::ostream & os)
{
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        for (int i = 0; i < melyseg; ++i)
            os << "----";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->>nullasGyermek (), os);
        --melyseg;
    }
}

void szabadit (Csmopont * elem)
{
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek ());
        szabadit (elem->>nullasGyermek ());

        delete elem;
    }
}
```

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

    void rmelyseg (Csmopont * elem);
    void ratlag (Csmopont * elem);
    void rszoras (Csmopont * 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 (Csmopont * elem)
{
    if (elem != NULL)
```

```
{  
    ++melyseg;  
    if (melyseg > maxMelyseg)  
        maxMelyseg = melyseg;  
    rmelyseg (elem->egyesGyermek ());  
    // ez a postorder bejáráshoz képest  
    // 1-el nagyobb mélység, ezért -1  
    rmelyseg (elem->>nullasGyermek ());  
    --melyseg;  
}  
}  
  
void  
LZWBinFa::ratlag (Csomopont * elem)  
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        ratlag (elem->egyesGyermek ());  
        ratlag (elem->>nullasGyermek ());  
        --melyseg;  
        if (elem->egyesGyermek () == NULL && elem->>nullasGyermek () == NULL ↔  
            )  
        {  
            ++atlagdb;  
            atlagosszeg += melyseg;  
        }  
    }  
}  
  
void  
LZWBinFa::rszoras (Csomopont * elem)  
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        rszoras (elem->egyesGyermek ());  
        rszoras (elem->>nullasGyermek ());  
        --melyseg;  
        if (elem->egyesGyermek () == 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[])
{

    // a kiírás szerint ./lzwtree in_file -o out_file alakra kell mennie, ←
    // ez 4 db arg:
    if (argc != 4)
    {
        // ha nem annyit kapott a program, akkor felhomályosítjuk erről a ←
        // júzetről:
        usage ();
        // és jelezzük az operációs rendszer felé, hogy valami gáz volt...
        return -1;
    }

    // "Megjegyezzük" a bemenő fájl nevét
    char *inFile = *++argv;

    // a -o kapcsoló jön?
    if (*((++argv) + 1) != 'o')
    {
        usage ();
        return -2;
    }

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

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

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

    unsigned char b;      // ide olvassuk majd a bejövő fájl bájtjait
    LZWBinFa binFa;      // s nyomjuk majd be az LZW fa objektumunkba

    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)
    {      // > karakter
        kommentben = true;
        continue;
    }

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

    if (kommentben)
        continue;

    if (b == 0x4e)      // N betű
        continue;

    for (int i = 0; i < 8; ++i)
    {

        if (b & 0x80)
            // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW ←
            // fa objektumunkba
            binFa << '1';
        else
            // különben meg a '0' betűt:
            binFa << '0';
        b <<= 1;
    }

}

kiFile << binFa;

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

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

    return 0;
}
```

Ebben a programban is a már meglévő állományt használjuk alapként. Itt annyi lesz a különbség, hogy az itteni fánk gyökere mutató legyen. Nem kell elvégeznünk sok módosítást. Az LZWBInFa osztály deklarálásában a "gyoker" számára helyet foglalunk. Ebből mutatót kapunk. Ezt a mutatót kell behelyettesítenünk olyan helyekre, ahol alapból erre a változóra hivatkoztunk. Egyszóval pár helyen el kell távolítanunk a "&" referencia operátort.

6.6. Mozgató szemantika

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

Forrás: https://gitlab.com/puskasmate/bhax/blob/master/thematic_tutorials/bhax_textbook/Welch/binfa_mozgato.cpp

A mutató a gyökér feladatban használt programon fogunk némi módosítást elvégezni, ezeket szeretném külön kiemelni. Mozgató, illetve mozgató értékadó konstruktort fogunk készíteni. A konstruktorban az alábbi módon kinullázzuk a gyökeret, majd az eredeti fa "*this" mutatóját az új fára állítjuk rá. A move függvény segítségével készítünk egy rvalue típusú referenciát.

```
LZWBInFa ( LZWBInFa&& regi) {
    std::cout << "LZWBInFa move ctor" << std::endl;

    gyoker = nullptr;
    *this = std::move(regi);

}
```

Az egyenlőség jel operátort az alábbi módon terheljük túl:

```
LZWBInFa& operator = (LZWBInFa&& regi){
    std::swap(gyoker, regi.gyoker);
    std::swap(fa, regi.fa);
    return *this;

}
```

Ezzel elérjük, hogy az egyenlőség jel használatakor a régi és új gyökerek kicserélődjenek a swap függvény segítségével. A this mutató az új fát adja vissza. Ezzel megtörténik a mozgató értékadás.

```
LZWBInFa binFa2 = std::move(binFa);
```

A programban itt történik az eredeti fa belemozgatása az új fába.

```
kiFile << "A másolt fa: \n";
kiFile << binFa2;

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

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

return 0;
}
```

Programunk azzal végződik, hogy kiíratjuk az új fát.

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

Megoldás forrása: https://gitlab.com/puskasmate/bhax/tree/master/thematic_tutorials/bhax_textbook/Conway/Hangya

A hangyszimulációs program tulajdonképpen mozgó pontokat fog ábrázolni, amely egy hangyacsoporthoz hasonlít. A projekteben 6 forráskód szerepel.

ant.h

```
class Ant
{
public:
    int x;
    int y;
    int dir;

    Ant(int x, int y) : x(x), y(y) {
        dir = qrand() % 8;
    }

};

typedef std::vector<Ant> Ants;

#endif
```

Az "ant.h" header-ben a hangya objektumok vannak definiálva. A hangyák viselkedése került lefordításra programozói nyelvre. A hangyáknak van pozíciójuk, ezt mi adjuk meg, illetve van irányuk, amit maguknak választanak ki. Valamint, hogy az ilyen típusú objektumokat "Ant" néven tároljuk.

antthread.h

```
#ifndef ANTHREAD_H
#define ANTHREAD_H

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

class AntThread : public QThread
{
    Q_OBJECT

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

    ~AntThread();

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

    void pause()
    {
        paused = !paused;
    }

    bool isRunning()
    {
        return running;
    }

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

```
int height;
int gridIdx;
int delay;

void timeDevel();

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

signals:
    void step ( const int &);

};

#endif
```

Ebben a header-ben deklaráljuk az AntThread osztályt. Ez az osztály rendelkezni fog mindenkel, amivel a QThread rendelkezik. Ez az osztály felelős a hangyák számításaiért. Deklarálunk benne egy run(), egy finish(), egy pause() és egy isRunning() függvényt. A finish a running értékét állítja hamisra, a pause a paused értékét negálja, az isRunning pedig visszaadja a running értékét. A private után deklaráljuk az összes változót, amelyekre a AntThread-nek szüksége van.

antwin.h

```
#ifndef ANTWIN_H
#define ANTWIN_H

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

class AntWin : public QMainWindow
{
    Q_OBJECT

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

    AntThread* antThread;
```

```
void closeEvent ( QCloseEvent *event ) {

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

void keyPressEvent ( QKeyEvent *event )
{

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

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

private:

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

public slots :
    void step ( const int &);

};

#endif
```

Ez a header is osztályok meghívásával keződik. A QMainWindow osztály fogja a szimuláció számára létrehozni az ablakot. A QPainter miatt a kirajzolás valósulhat meg, a QString a karakterláncokat fogja kezelní, a QCloseEvent pedig a kilépésért felelős, valamint meghívjuk a fentebb látható 2 headert. Átadjuk az AntWin osztályt, és a konstruktor illetve a destruktur mellett egy paintEvent nevű függvényt is deklarálunk. A keyPressEvent függvényt az ablak bezárása és a szimuláció szüneteltetése miatt szükséges deklarálnunk. A Q valamint az Escape billentyű megnyomásával bezárhatjuk az ablakot, a P billentyű egnyomásával pe-

dig szüneteltethetjük a szimulációt. A private részben a szokásos változókat deklaráljuk. Végezetül egy új slotot deklarálunk step néven.

antwin.cpp

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

A szükséges header-ök meghívásával kezdünk.

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

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

    cellWidth = 6;
    cellHeight = 6;

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

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

    gridIdx = 0;
    grid = grids[gridIdx];

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

    ants = new Ants();

    antThread = new QThread ( this );
    antThread->start();
    connect ( antThread, SIGNAL ( finished() ), this, SLOT ( step() ) );
    connect ( antThread, SIGNAL ( finished() ), antThread, SLOT ( deleteLater() ) );
}
```

```
    connect ( antThread, SIGNAL ( step ( int ) ),
              this, SLOT ( step ( int ) ) );
    antThread->start ();
}
```

A konstruktorban megadjuk az ablak nevét, méreteit, és beállítjuk, hogy mekkora legyen egy cella. A grids tömbben beállítjuk a rácsokat. Létrehozunk egy AntThread-et antThred néven. A connect függvényel összekötjük a step slottal az előbb létrehozott antThread-et, majd a start függvény segítségével elindítjuk a folyamatot.

```

void AntWin::paintEvent ( QPaintEvent* )
{
    QPainter qpainter ( this );
    grid = grids[gridIdx];

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

            double rel = 255.0/max;

            qpainter.fillRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight,
                                QColor ( 255 - grid[i][j]*rel,
                                         255,
                                         255 - grid[i][j]*rel ) );

            if ( grid[i][j] != min )
            {
                qpainter.setPen (
                    QPen (
                        QColor ( 255 - grid[i][j]*rel,
                                 255 - grid[i][j]*rel, 255 ),
                        1 )
                );
                qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                    cellWidth, cellHeight );
            }
        }
    }

    qpainter.setPen (
        QPen (
            QColor ( 0,0,0 ), 1 );
}

```

```
        qpainter.drawRect ( j*cellWidth, i*cellHeight,
                            cellWidth, cellHeight );

    }

}

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

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

}

qpainter.end();
}
```

A paintEvent függvény kiszínezi az ablakot. A hangyák színe a fekete lesz.

```
AntWin::~AntWin()
{
    delete antThread;

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

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

    delete ants;
}
```

Fentebb látható a destruktur, amely törli az antThread-et és a rácsokat, valamint a hangyákat.

```
void AntWin::step ( const int &gridIdx )
{
    this->gridIdx = gridIdx;
    update();
}
```

A step függvény pedig folyamatosan átállítja a gridIdx aktuális értékét, és az update függvény segítségével az ablakot is minden frissíti.

antthread.cpp

```
AntThread::AntThread ( Ants* ants, int*** grids,
                      int width, int height,
```

```
        int delay, int numAnts,
        int pheromone, int nbrPheromone,
        int evaporation,
        int min, int max, int cellAntMax)
{
    this->ants = ants;
    this->grids = grids;
    this->width = width;
    this->height = height;
    this->delay = delay;
    this->pheromone = pheromone;
    this->evaporation = evaporation;
    this->min = min;
    this->max = max;
    this->cellAntMax = cellAntMax;
    this->nbrPheromone = nbrPheromone;

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

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

    qsrand ( QDateTime::currentMSecsSinceEpoch() );
}

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

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

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

    ants->push_back ( h );
}

gridIdx = 0;
}
```

A header-ök meghívása után fentebb látható a konstruktor. Itt értékkadás történik, valamint a hangyák szétosztása a rácsokon.

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

```
int ifrom, ito;
int jfrom, jto;

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

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

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

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

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

    }

return sum;
}
```

A sumNbhs (sumNeighborhoods) ahogy a nevében van, összeszámolja, hogy egy adott cellának hány szomszédja van.

```
int AntThread::newDir ( int sor, int oszlop, int vsor, int voszlop )

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

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

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

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

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

        return -1;
    }
}
```

A newDir függvény egy hangya számára egy új irány kiosztásáért felelős.

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

```
{  
  
    switch ( dir ) {  
        case 0:  
            ifrom = -1;  
            ito = 0;  
            jfrom = -1;  
            jto = 2;  
            break;  
        case 1:  
            ifrom = -1;  
            ito = 1;  
            jfrom = 0;  
            jto = 2;  
            break;  
        case 2:  
            ifrom = -1;  
            ito = 2;  
            jfrom = 1;  
            jto = 2;  
            break;  
        case 3:  
            ifrom =  
                0;  
            ito = 2;  
            jfrom = 0;  
            jto = 2;  
            break;  
        case 4:  
            ifrom = 1;  
            ito = 2;  
            jfrom = -1;  
            jto = 2;  
            break;  
        case 5:  
            ifrom = 0;  
            ito = 2;  
            jfrom = -1;  
            jto = 1;  
            break;  
        case 6:  
            ifrom = -1;  
            ito = 2;  
            jfrom = -1;  
            jto = 0;  
            break;  
        case 7:  
            ifrom = -1;  
            ito = 1;  
            jfrom = -1;
```

```
    jto = 1;
    break;

}

}
```

A detDirs megvizsgálja egy hangya mozgásának irányát, és ez alapján a lehetséges útvonalakat állapítja meg.

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

    int ifrom, ito;
    int jfrom, jto;

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

    double osszes = sumNbhs ( racs, sor, oszlop, dir );
    double random = ( double ) ( qrand() %1000000 ) / ( double ) 1000000.0;
    double gvalseg = 0.0;

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

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

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

```
double kedvezo = (racs[s][o]+1)*(racs[s][o]+1)*(racs[s][o ←
] +1);

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

if ( gvalseg >= random ) {

    vsor = s;
    voszlop = o;

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

}

}

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

return dir;
}
```

A moveAnts függvény, mint azt a neve is mutatja a hangyák mozgatásáért felelős.

```
void AntThread::timeDevel()
{

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

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

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

    for ( Ant &h: *ants )
    {

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

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

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

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

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

    }

}

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

A timeDevel függvény az összes hangya mozgásáért felelős, a gridIdx változó értékét változtatja folyamatosan. A mozgatás 2 rácson történik, az egyik rács a régi, az új rács pedig a régi alapján jön létre, ugyanis az új helyzeteket itt tároljuk a régiek felhasználásával.

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

    for ( int i=-1; i<2; ++i )
        for ( int j=-1; j<2; ++j )
            if ( ! ( ( i==0 ) && ( j==0 ) ) )
            {
                int o = oszlop + j;
                {
                    if ( o < 0 ) {
                        o = width-1;
                    } else if ( o >= width ) {
                        o = 0;
                    }
                }
                int s = sor + i;
                {
                    if ( s < 0 ) {
                        s = height-1;
                    } else if ( s >= height ) {
                        s = 0;
                    }
                }

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

```
    }

}

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

}
```

A setPheromone függvény beállítja a feromonokat az aktuális cellán.

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

        QThread::msleep ( delay );

        if ( !paused ) {
            timeDevel();
        }

        emit step ( gridIdx );
    }
}
```

A run függvény a folyamat futásáért felelős. A timeDevel függvény is meghívásra kerül benne, így felfüggesztheti egyszerre az összes hangya mozgását.

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

    delete [] numAntsinCells;
}
```

A destrukturban a hangyákat tároló tömb elemeit egyesével töröljük, amint a hangyákat kitöröltük, eltávolítjuk a hangyákat tároló tömböt is.

main.cpp

```
#include <QApplication>
#include <QDesktopWidget>
#include <QDebug>
```

```
#include <QDateTime>
#include <QCommandLineOption>
#include <QCommandLineParser>

#include "antwin.h"

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

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

    QApplication a ( argc, argv );

    QCommandLineOption szeles_opt ( {"w","szelesseg"}, "Oszlopok (cellakban ←
        ) szama.", "szelesseg", "200" );
    QCommandLineOption magas_opt ( {"m","magassag"}, "Sorok (cellakban) ←
        szama.", "magassag", "150" );
    QCommandLineOption hangyaszam_opt ( {"n","hangyaszam"}, "Hangyak szama ←
        .", "hangyaszam", "100" );
    QCommandLineOption sebesseg_opt ( {"t","sebesseg"}, "2 lepes kozotti ←
        ido (millisec-ben).", "sebesseg", "100" );
    QCommandLineOption parolgas_opt ( {"p","parolgas"}, "A parolgas erteke ←
        .", "parolgas", "8" );
    QCommandLineOption feromon_opt ( {"f","feromon"}, "A hagyott nyom ←
        erteke.", "feromon", "11" );
    QCommandLineOption szomszed_opt ( {"s","szomszed"}, "A hagyott nyom ←
        erteke a szomszedokban.", "szomszed", "3" );
    QCommandLineOption alapertek_opt ( {"d","alapertek"}, "Indulo ertek a ←
        cellakban.", "alapertek", "1" );
    QCommandLineOption maxcella_opt ( {"a","maxcella"}, "Cella max erteke ←
        .", "maxcella", "50" );
    QCommandLineOption mincella_opt ( {"i","mincella"}, "Cella min erteke ←
        .", "mincella", "2" );
    QCommandLineOption cellamerete_opt ( {"c","cellameret"}, "Hany hangya ←
        fer egy cellaba.", "cellameret", "4" );
    QCommandLineParser parser;

    parser.addHelpOption();
    parser.addVersionOption();
    parser.addOption ( szeles_opt );
    parser.addOption ( magas_opt );
    parser.addOption ( hangyaszam_opt );
    parser.addOption ( sebesseg_opt );
    parser.addOption ( parolgas_opt );
    parser.addOption ( feromon_opt );
```

```
parser.addOption ( szomszed_opt );
parser.addOption ( alapertek_opt );
parser.addOption ( maxcella_opt );
parser.addOption ( mincella_opt );
parser.addOption ( cellamerete_opt );

parser.process ( a );

QString szeles = parser.value ( szeles_opt );
QString magas = parser.value ( magas_opt );
QString n = parser.value ( hangyaszam_opt );
QString t = parser.value ( sebesseg_opt );
QString parolgas = parser.value ( parolgas_opt );
QString feromon = parser.value ( feromon_opt );
QString szomszed = parser.value ( szomszed_opt );
QString alapertek = parser.value ( alapertek_opt );
QString maxcella = parser.value ( maxcella_opt );
QString mincella = parser.value ( mincella_opt );
QString cellameret = parser.value ( cellamerete_opt );

qsrand ( QDateTime::currentMSecsSinceEpoch() );

AntWin w ( szeles.toInt(), magas.toInt(), t.toInt(), n.toInt(), feromon.toInt(),
           szomszed.toInt(), parolgas.toInt(),
           alapertek.toInt(), mincella.toInt(), maxcella.toInt(),
           cellameret.toInt() );

w.show();

return a.exec();
}
```

A szükséges header-ök meghívása, valamint a megfelelő osztályok hozzáadása után "a" néven létrehozunk egy QApplication példányt. Ez lesz a konstruktur, melynek paraméterei az argc és argv lesznek. Ezután a lehetséges kapcsolókat adjuk meg. Ezután az "a" alapján egy parser hozunk létre. Ehhez ismét hozzáadjuk a lehetséges kapcsolókat, valamint egy help és egy version lehetőséget. Ezután a process függvény segít feldolgozni az "a" adatait, majd eltárolja ezeket QString-ekbe. Véletlenszámok előállításával egy AntWin objektumot állítunk elő, melyre "w" néven tudunk hivatkozni. A show függvénnyel megmutatjuk a felhasználónak az ablakot. Láthatjuk, hogy a programunk az "a"-nak az exec függvény által módosított értékét adja vissza, amely a q lenyomásakor tér vissza.

7.2. Qt C++ életjáték

Most Qt C++-ban!

Forrás: https://gitlab.com/puskasmate/bhax/tree/master/thematic_tutorials/bhax_textbook/Conway/Hangya2/sejtszal.h

```
#ifndef SEJTSZAL_H
#define SEJTSZAL_H

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

class SejtAblak;

class SejtSzal : public QThread
{
    Q_OBJECT

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

protected:
    bool ***racsok;
    int szelesseg, magassag;
    // Megmutatja melyik rács az aktuális: [rácsIndex][][][]
    int racsIndex;
    // A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
    // közötti valós idő.
    int varakozas;
    void idoFejlodes();
    int szomszedokSzama(bool **racs,
                         int sor, int oszlop, bool allapot);
    SejtAblak* sejtAblak;

};

#endif // SEJTSZAL_H
```

A "sejtszal.h" header-ben a SejtAblak osztályt fogjuk deklární. Ez az osztály ismét a QThread adattípusúval fog rendelkezni. A public részben deklarájuk a konstruktort valamint a destruktort, továbbá a run() függvényt. A protected részben pedig a konstruktur által kért változókat deklarájuk. Egy 3 dimenziós rácsok tömböt, a szélesség és magasság értékét, a racsIndex változót, mely az aktuális rácsot mutatja meg, egy varakozás változót, egy időfejlődés függvényt, valamint a szomszédok számának meghatározására egy függvényt, és egy sejtablak típusú pointert.

sejtablak.h

```
#ifndef SEJTABLAK_H
#define SEJTABLAK_H

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

```
class SejtSzal;

class SejtAblak : public QMainWindow
{
    Q_OBJECT

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

    ~SejtAblak();
    // Egy sejt lehet élő
    static const bool ELO = true;
    // vagy halott
    static const bool HALOTT = false;
    void vissza(int racsIndex);

protected:
    // Két rácsot használunk majd, az egyik a sejttér állapotát
    // a t_n, a másik a t_n+1 időpillanatban jellemzi.
    bool ***racsok;
    // Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
    // [2][][]-ból az első dimenziót használni, mert vagy az egyikre
    // állítjuk, vagy a másikra.
    bool **racs;
    // Megmutatja melyik rács az aktuális: [räcsIndex][][][]
    int racsIndex;
    // Pixelben egy cella adatai.
    int cellaSzelesseg;
    int cellaMagassag;
    // A sejttér nagysága, azaz hányszor hány cella van?
    int szelesseg;
    int magassag;
    void paintEvent(QPaintEvent* );
    void siklo(bool **racs, int x, int y);
    void sikloKilovo(bool **racs, int x, int y);

private:
    SejtSzal* eletjatek;

};

#endif // SEJTABLAK_H
```

A sejtszal.h mellett további osztályokat hívunk meg. A sejtszal segítségével deklaráljuk a sejtablak nevű osztályt. Egy sejt lehet élő, vagy halott, ezen állapot eldöntését egy bool típusú változóban tároljuk, és deklarálunk egy "vissza" nevű void típusú függvényt. A protected rész magyarázata fentebb található kommentként sorról sorra. Az utolsó 3 függvény 3 rajzoló függvény. A private részben egy SejtSzal típusú pointert deklarálunk.

sejtablak.cpp

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

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

    cellaSzelesseg = 6;
    cellaMagassag = 6;

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

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

    racsIndex = 0;
    racs = racsok[racsIndex];

    // A kiinduló racs minden cellája HALOTT
    for(int i=0; i<magassag; ++i)
        for(int j=0; j<szelesseg; ++j)
            racs[i][j] = HALOTT;
    // A kiinduló racsra "ELölényeket" helyezünk
    //siklo(racs, 2, 2);

    sikloKilovo(racs, 5, 60);

    eletjatek = new SejtSzal(racsok, szelesseg, magassag, 120, this);

    eletjatek->start();
}
```

Itt látható a konstruktur. Itt történik az ablak nevének és méreteinek, valamint a cellák méreteinek beállítása. Létrehozunk két rácsot, fel is töltjük ezeket halott sejtekkel, majd a rajzoló függvények segítségével rajzolhatunk az ablakba. Egy SejtSzal típusú objektumot is létrehozunk a fentebb látható paraméterekkel, majd a start függvény meghívásával elindul a folyamat.

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

    // Az aktuális
```

```
bool **racs = racsok[racsIndex];
// racsot rajzoljuk ki:
for(int i=0; i<magassag; ++i) { // végig lépked a sorokon
    for(int j=0; j<szelesseg; ++j) { // s az oszlopok
        // Sejt cella kirajzolása
        if(racs[i][j] == ELO)
            qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                cellaSzelesseg, cellaMagassag, Qt::black);
        else
            qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                cellaSzelesseg, cellaMagassag, Qt::white);
        qpainter.setPen(QPen(Qt::gray, 1));

        qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
            cellaSzelesseg, cellaMagassag);
    }
}

qpainter.end();
}
```

A paintEvent függvény minden sejtet kirajzol, ehhez végigmegy a rácson, az élő sejteket feketére, a halott sejteket fehérrre festi. A cella szegélyeit szürkére rajzolja.

```
SejtAblak::~SejtAblak()
{
    delete eletjatek;

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

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

A destruktur törli az "eletjatek"-ot, a rácsok tartalmát, majd a rácsokat, végül a rácsokat eltároló tömböket is.

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

A vissza függvény paraméterül a racsIndexet kapja, majd ennek a változtatásával az update függvényel frissíti az ablakot.

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

A siklót állítja elő a rácson az x és y segítségével.

```
void SejtAblak::sikloKilovo(bool **racs, int x, int y) {  
  
    racs[y+ 6][x+ 0] = ELO;  
    racs[y+ 6][x+ 1] = ELO;  
    racs[y+ 7][x+ 0] = ELO;  
    racs[y+ 7][x+ 1] = ELO;  
  
    racs[y+ 3][x+ 13] = ELO;  
  
    racs[y+ 4][x+ 12] = ELO;  
    racs[y+ 4][x+ 14] = ELO;  
  
    racs[y+ 5][x+ 11] = ELO;  
    racs[y+ 5][x+ 15] = ELO;  
    racs[y+ 5][x+ 16] = ELO;  
    racs[y+ 5][x+ 25] = ELO;  
  
    racs[y+ 6][x+ 11] = ELO;  
    racs[y+ 6][x+ 15] = ELO;  
    racs[y+ 6][x+ 16] = ELO;  
    racs[y+ 6][x+ 22] = ELO;  
    racs[y+ 6][x+ 23] = ELO;  
    racs[y+ 6][x+ 24] = ELO;  
    racs[y+ 6][x+ 25] = ELO;  
  
    racs[y+ 7][x+ 11] = ELO;  
    racs[y+ 7][x+ 15] = ELO;  
    racs[y+ 7][x+ 16] = ELO;  
    racs[y+ 7][x+ 21] = ELO;  
    racs[y+ 7][x+ 22] = ELO;  
    racs[y+ 7][x+ 23] = ELO;  
    racs[y+ 7][x+ 24] = ELO;  
  
    racs[y+ 8][x+ 12] = ELO;  
    racs[y+ 8][x+ 14] = ELO;  
    racs[y+ 8][x+ 21] = ELO;  
    racs[y+ 8][x+ 24] = ELO;  
    racs[y+ 8][x+ 34] = ELO;
```

```
    racs[y+ 8][x+ 35] = ELO;

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

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

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

}
```

Ez pedig a siklokilovo-t valósítja meg.

sejtszal.cpp

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

    racsIndex = 0;
}
```

A konstruktorban megtörténik az értékkadás.

```
/***
 * Az kérdezett állapotban lévő nyolcsomiszédok száma.
 *
 * @param rács a sejttér rács
 * @param sor a rács vizsgált sora
 * @param oszlop a rács vizsgált oszlopa
 * @param állapor a nyolcsomiszédok vizsgált állapota
 * @return int a kérdezett állapotbeli nyolcsomiszédok száma.
 */
int SejtSzal::szomszedokSzama(bool ***racs,
                                int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
    // A nyolcsomiszédok végigzongorázása:
```

```
for(int i=-1; i<2; ++i)
    for(int j=-1; j<2; ++j)
        // A vizsgált sejtet magát kihagyva:
        if(!((i==0) && (j==0))) {
            // A sejttérből szélénk szomszédai
            // a szembe oldalakon ("periódikus határfeltétel")
            int o = oszlop + j;
            if(o < 0)
                o = szelesség-1;
            else if(o >= szelesség)
                o = 0;

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

            if(racs[s][o] == állapot)
                ++állapotSzomszed;
        }
    }

    return állapotSzomszed;
}
```

A szomszedokSzama a szomszédságból azon elemek számát adja vissza, amelyet megadtunk.

```
/***
 * A sejttér időbeli fejlődése a John H. Conway féle
 * életjáték sejtautomata szabályai alapján történik.
 * A szabályok részletes ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 171. oldal.)
 */
void SejtSzal::idoFejlodes() {

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

    for(int i=0; i<magasság; ++i) { // sorok
        for(int j=0; j<szelesség; ++j) { // oszlopok

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

            if(racsElotte[i][j] == SejtAblak::ELO) {
                /* Elő élő marad, ha kettő vagy három elő
                szomszedja van, különben halott lesz. */
                if(elok==2 || elok==3)
                    racsUtana[i][j] = SejtAblak::ELO;
                else
                    racsUtana[i][j] = SejtAblak::HALOTT;
            }
        }
    }
}
```

```
    } else {
        /* Halott halott marad, ha három élő
        szomszedja van, különben élő lesz. */
        if(elok==3)
            racsUtana[i][j] = SejtAblak::ELO;
        else
            racsUtana[i][j] = SejtAblak::HALOTT;
    }
}
racsIndex = (racsIndex+1)%2;
}
```

Az idoFejlodes függvény a következő rácsba tölti bele a következő időbeli állapotot, ehhez az egyel fentebbi függvényt is meghívja. Egy sejt akkor marad élő, ha 2 vagy 3 élő szomszédja van. Egy halott sejt akkor marad halott, ha 3 szomszédja van, egyébként élő lesz. Ezeket a vizsgálatokat elvégzi az összes sejtre, majd a racsIndex értékét frissítjük.

```
/** A sejttér időbeli fejlődése. */
void SejtSzal::run()
{
    while(true) {
        QThread::msleep(varakozas);
        idoFejlodes();
        sejtAblak->vissza(racsIndex);
    }
}
```

A run függvényben egy végtelen ciklusban az msleep segítségével a várakozást altatjuk, meghívjuk az időfejlődés függvényt, és a vissza függvény segítségével az újra rajzolandó rácsot kapjuk vissza, ígyfrissül az ablak.

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

A destruktur üresen található.

main.cpp

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

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

```
    return a.exec();
}
```

Hozzáadjuk a szükséges header-öket, majd egy QApplication típusú, "a" nevű változót hozunk létre, átadjuk neki a parancssori argumentumokat. "w" néven deklarálunk egy SejtAblak típusú változót, melynek méreteit megadjuk. A show függvényel az előbb elindított életjátékot mutatjuk meg a felhasználónak, a visszatérési érték az előző programhoz hasonlóan az "a"-nak az exec függvény által módosított értéke lesz, ami az ablak bezárásakor lesz 0.

7.3. BrainB Benchmark

Megoldás videó:

Megoldás forrása: https://gitlab.com/puskasmate/bhax/tree/master/thematic_tutorial_bhax_textbook/Conway/Brain_B

A feladatunk annyi, hogy a kurzort a Samu Entropy nevű karakteren kell tartani. Ahogy tellik az idő, egyre több karakter jelenik meg, így nincs egyszerű dolgunk.

BrainBWin.h

```
#ifndef BrainBWin_H
#define BrainBWin_H

#include <QKeyEvent>
#include <QMainWindow>
#include <QPixmap>
#include <QPainter>
#include <QFont>
#include <QFile>
#include <QString>
#include <QCLOSEEvent>
#include <QDate>
#include <QDir>
#include <QDateTime>
#include "BrainBThread.h"

enum playerstate {
    lost,
    found
};

class BrainBWin : public QMainWindow
{
    Q_OBJECT

    BrainBThread *brainBThread;
    QPixmap pixmap;
    Heroes *heroes;
```

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

int xs, ys;

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

QString statDir;

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

    void closeEvent ( QCloseEvent *e ) {

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

    }

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

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

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

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

    if ( vect.size() > 1 ) {

        double accum = 0.0;

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

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

}

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

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

}

bool save ( int t ) {

    bool ret = false;

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

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

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

        textStremam << appName + " " + appVersion << "\n";
        textStremam << "time      : " << brainBThread->getT() << "\n";
        textStremam << "bps       : " << brainBThread->get_bps() << "\n" <<
```

```
    n";
textStremam << "noc      : " << brainBThread->nofHeroes() << ←
    "\n";
textStremam << "nop      : " << brainBThread->get_nofPaused() <<
    "\n";

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

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

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

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

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

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

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

        tfile.close();
    }
    return ret;
}

public slots :

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

#endif // BrainBWin
```

Tartalmaz egy konstruktort, egy destruktort, olyan függvényeket, amelyek az inputról beérkező jeleket figyelik. A kirajzolásért felelős függvények is itt vannak, illetve az ablak frissítéséért felelős függvények, valamint egy mentésért felelős függvény is definiáltunk.

BrainBWin.cpp

```
#include "BrainBWin.h"

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

BrainBWin::BrainBWin ( int w, int h, QWidget *parent ) : QMainWindow ( ←
    parent )
{
    //    setWindowTitle(appName + " " + appVersion);
    //    setFixedSize(QSize(w, h));

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

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

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

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

void BrainBWin::endAndStats ( const int &t )
{
    qDebug() << "\n\n\n";
    qDebug() << "Thank you for using " + appName;
    qDebug() << "The result can be found in the directory " + statDir;
    qDebug() << "\n\n\n";

    save ( t );
    close();
}

void BrainBWin::updateHeroes ( const QImage &image, const int &x, const int &y )
{
    if ( start && !brainBThread->get_paused() ) {

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

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

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

            firstLost = true;

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

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

```
        if ( noffound > 12 ) {

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

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

            brainBThread->incComp();
        }

    }

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

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

    QPainter qpainter ( this );

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

    qpainter.drawPixmap ( xs, ys, pixmap );

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

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

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

```
if ( brainBThread->get_paused() ) {
    QString pausedstr = "PAUSED (" + QString::number ( ←
        brainBThread->get_nofPaused() ) + ")";
    qpainter.drawText ( 210, 40, pausedstr );
}

qpainter.end();
}

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

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

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

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

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

BrainBWin::~BrainBWin()
{
```

```
}
```

Ebben a programban magyarázzuk meg a fentebbi header-ben deklarált dolgokat.

BrainBThread.h

```
#ifndef BrainBThread_H
#define BrainBThread_H

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

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

class Hero
{

public:
    int x;
    int y;
    int color;
    int agility;
    int cond {0};
    std::string name;

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

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

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

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

};

class BrainBThread : public QThread
{
    Q_OBJECT

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

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

    Heroes heroes;
    int heroRectSize {40};

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

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

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

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

public:
```

```
BrainBThread ( int w = 256, int h = 256 );
~BrainBThread();

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

    return delay;

}

void setDelay ( int delay ) {

    if ( delay > 0 ) {
        delay = delay;
    }

}

void devel() {

    for ( Hero & hero : heroes ) {

        hero.move ( w, h, ( h<w ) ?h/10:w/10 );

    }

}

int nofHeroes () {

    return heroes.size();

}

std::vector<int> &lostV () {

    return lostBPS;

}

std::vector<int> &foundV () {

    return foundBPS;

}

double meanLost () {

    return mean ( lostBPS );
```

```
}

double varLost ( double mean ) {

    return var ( lostBPS, mean );

}

double meanFound () {

    return mean ( foundBPS );

}

double varFound ( double mean ) {

    return var ( foundBPS, mean );

}

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

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

}

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

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

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

int get_bps() const {

    return bps;

}

int get_w() const {

    return w;

}
```

```
bool get_paused() const {
    return paused;
}

int get_nofPaused() const {
    return nofPaused;
}

void decComp() {
    lostBPS.push_back( bps );
    if ( heroes.size() > 1 ) {
        heroes.pop_back();
    }

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

void incComp() {
    foundBPS.push_back( bps );
    if ( heroes.size() > 300 ) {
        return;
    }

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

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

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

double ry = 200.0;
if(heroes[0].y - 200 < 0)
    ry = heroes[0].y;
else if(heroes[0].y + 200 > h)
    ry = h - heroes[0].y;

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

heroes.push_back ( other );

for ( Hero & hero : heroes ) {

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

void draw () {

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

    for ( Hero & hero : heroes ) {

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

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

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

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

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

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

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

}

cv::Mat comp;

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

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

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

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

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

prev = focus;

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

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

}

long getT() const {

    return time;

}

void finish () {

    time = endTime;

}

signals:
```

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

};

#endif // BrainBThread_H
```

8. fejezet

Helló, Gutenberg!

8.1. Programozási alapfogalmak

[?]

Juhász István: Magas Szintű Programozási Nyelvek 1

A programozási nyelveknek 3 fajtája van:

- gépi nyelv
- assembly szintű nyelv
- magas szintű nyelv

A magas szintű nyelven megírt programokat forrásprogramnak nevezzük. Ezeket a forráskódokat szabályok határozzák meg. Szabályok alatt a szintaktikai, azaz a nyelvtani szabályok összessége, valamint a szemantikai, vagyis a tartalmi és értelmezési szabályok, együttese adja.

A processzorok csak a saját gépi nyelükön megírt programokat képesek végrehajtani. Ehhez viszont a meglévő forrásszövegből futtatható állományt, programot kell készítenünk. Ezt kétféleképpen tehetjük meg.

Fordítóprogramos módszer:

Szükségünk lesz egy speciális programra, amely a magas nyelven megírt forrásszövegből képes gépi kódú tárgyprogramot előállítani. A fordítóprogram a forrásszöveget egy egységekét kezeli, és 4 lépést hajt végre: lexikális elemzés szintaktikai elemzés szemantikai elemzés kódgenerálás

A lexikális elemzés felosztja a forrásszöveget lexikális egységekre, a szintaktikai elemzés során megnézi, hogy a forrásszöveg eleget tesz-e az adott nyelv szintaktikai szabályainak. Amennyiben szintaktikailag nem helyes a forrásszöveg, nem képes a gép számára értelmezhető szöveget generálni. Amennyiben helyes, a szövegből tárgyprogramot készít, ami gépi nyelvű, de még nem futtatható, így ebből egy szerkesztő segítségével állíthatunk elő futtatható állományt. Ezt a betöltő elhelyezi a tárban, és a futtató rendszer lesz felelős a futó program működéséért.

A fordítóprogramok valamilyen tetszőleges nyelvről egy tetszőleges nyelvre fordítanak. Létezik olyan magas szintű nyelv, amely nem nyelvi elemeket is tartalmaz, ekkor előfordító segítségével egy adott nyelvű forrásprogramot kell generálni, amit a nyelv fordítója képes feldolgozni. Ilyen nyelvre egy példa a C nyelv.

Interpreteres módszer:

Az első három lépés itt is megtalálható, viszont ezen módszert alkalmazva nem készül tárgyprogram. A forrásprogramot sorról (utasításról) sorra értelmezi és végrehajtja .Az eredményt valamilyen gépi rutin lefutása után kapjuk.

A programnyelvek lehetnek fordítóprogramos, vagy interpreteresek, vagy pedig minden módszert alkalmazzák.

A hivatkozási nyelvek alatt a program saját szabványát értjük. A hivatkozási nyelvben pontosan definiálva vannak a szintaktikai és szemantikai szabályok. A hivatkozási nyelv mellett léteznek implementációk, ezek vagy fordítóprogramok vagy interpreterek. Az implementációkból sok van, viszont ezek egymással illetve a hivatkozási nyelvvel nem kompatibilisek , így a hordozhatóság problémája még 50 év után is fennáll. Manapság a programok írásához integrált fejlesztői környezeteket (IDE) is használhatunk, melyek megkönnyítik a munkánkat.

A programnyelveket 3 féle csoportra bonthatjuk:

Imperatív nyelvek:

- Algoritmikus nyelvek: algoritmust kódolunk
- utasítások sorozata adja a programot
- Legfőbb programozói eszköz a változó
- Szorosan kötődnek a Neumann-architektúrához
- Alcsoporthjai: Eljárásorientált nyelvek, Objektumorientált nyelvek

Deklaratív nyelvek:

- Nem algoritmikus nyelvek
- Nem kötődnek szorosan a Neumann-architektúrához
- A programozó csak a problémát adja meg
- Nincs lehetőség memóriaműveletekre
- Alcsoporthjai: funkcionális nyelvek, logikai nyelvek

Máselvű nyelvek

Adattípusok

Az adattípusok ún. absztrakt programozási eszközök, melyek más programozási eszköz komponenseiként jelennek meg. Névvel, azaz azonosítóval rendelkeznek.

Egy adattípust 3 dolog is meghatároz:

- tartomány
- műveletek
- reprezentáció

Minden nyelv rendelkezik beépített típusokkal, viszont saját típus definiálására is van lehetőség, ehhez meg kell adnunk a tartományát és műveleteit. Azonban nem minden nyelvben lehet saját műveleteket és operátorokat megadni ezekhez a típusokhoz.

Az adattípusokat két csoportra tudjuk bontani.

Egyszerű (skalár) adattípus

Atomi értékeket tartalmaz, minden értéke egyedi.

Ide tartoznak például az egészek ábrázolására használt típusok, melyek fixpontos ábrázolásúak, vagy a valós típusok, melyek lebegőpontos ábrázolásúak.

Ezeket numerikus típusoknak hívjuk, és ezeken numerikus valamint hasonlító műveleteket hajthatunk végreh.

A karakteres típus tartományába az elemi karakterek, a sztring típus elemei pedig a karaktersorozatok. Ezeket karakteresen ábrázoljuk, szöveges és hasonlító műveleteket lehet velük végezni.

Néhány nyelvben megjelenik a logikai típus, mely tartománya igaz és hamis értékekből áll, logikai és hasonlító műveletek elvégzésére alkalmas.

Létezik ún. felsorolásos típus, de ezt magunknak kell létrehozni, néhány nyelv pedig értelmezi a sorszámozott típust is.

Összetett típusok

Két legfontosabb típusa a tömb és a rekord. Utóbbit csak a FORTRAN nyelv nem ismeri.

A tömb statikus és homogén összetett típus, ami azt jelenti, hogy olyan értékcsoporthat foglal magba, amelyben az elemek száma és típusa azonos. Egy tömböt dimenzióinak száma, indexkészletének tartomány és típusa, valamint elemeinek típusa határoz meg.

Egyes nyelvek a többdimenziós tömböket nem ismerik. A többdimenziós tömbök lehetnek sor- vagy oszlopfolytonosak.

A tömb nevével az összes elemére együtt tudunk hivatkozni. Csak bizonyos elemekre indexek megadásával tudunk hivatkozni.

A rekord típus minden esetben heterogén, olyan értékcsoporthat foglal magába, melyek elemei lehetnek különböző típusúak is. Az értékcsoporthat belül mezők találhatóak. Ezeknek neve és saját típusa van, hivatkozni rájuk a következőképp tudunk: eszközneve.mezőnév

Mutató típus

Elemei tárcímek, az indirekt címzés valósítható meg velük, értékük tárbeli címek.

A nevesített konstans

3 komponense van: név, típus, érték.

Gyakorlatban a nevével jelenik meg, és mindig az értékét jelenti, értéke nem változtatható meg a futás során. Sokszor előforduló értékeknél szoktuk használni.

#define név literál alakban lehet bevezetni.

A változó

4 komponense van: név, attribútumok, cím és érték. A név azonosítót, az attribútumok viselkedési jellemzőket határoznak meg. Utóbbi komponenst deklarációval állítjuk be, mely történhet explicit, implicit és automatikus módon.

Címeket is több módon oszthatunk ki.

Statikusan: futás előtt eldől a változó címe

Dinamikusan: futtató rendszer végzi a hozzárendelést

Programozó által vezérelt módon: mi rendeljük a változóhoz a címkomponenst.

Mindhárom esetben kell lennie egy olyan eszköznek, mellyel a programozó megszüntetheti a címköponenst.

Több címhozzárendelést ismernek általában a nyelvek, ezek a statikus, dinamikus, és a programozó által vezérelt címkiosztások.

A változók megadása két féleképpen történhet. Kérhetünk be a felhasználótól (input módon), valamint adhatunk nekik kezdőértéket.

Kifejezések

Arra valók, hogy a programban ismert értékekből új értéket határozzunk meg. 2 komponense van: érték és típus.

Operandusokból, -lehet literál, nevesített konstans, változó vagy függvényhívás- operátorokból (műveleti jelek) és kerek zárójelekből -műveletek végrehajtási sorrendjét befolyásolják- áll.

A kifejezéseket a használt operandusok száma alapján csoportosíthatjuk.

Alakjai: prefix (* 3 5), infix (3 * 5), postfix (3 5 *) lehet. Az eljárásorientált nyelvek az infix alakot használják.

A műveletek feldolgozási sorrendje is alakfüggő. Néhány nyelvben megjelennek ún. rövidzár operátorok, melyeknél nem szükséges az egész kifejezést kiértékelni, hogy megkapjuk az eredményt. (pl. és operátor). Továbbá nyelvenként változik még, hogy egy kifejezésnél csak azonos típusú vagy különböző típusú operandusokkal végezünk műveleteket. Utóbbi eset típuskonverzió segítségével jöhet létre.

Utasítások

A programozási nyelvek egyik legfontosabb elemei. Olyan egységek, melyek algoritmusok lépéseiit adják meg. Két csoportra bontjuk: deklarációs és végrehajtható utasítások.

A deklarációs utasítások mögött nem áll tárgykód, ezek a fordítóprogramnak szólnak. A végrehajtható utasításokból generálódik a tárgykód.

Végrehajtható utasítások csoportjai: értékkopírozó-, üres-, ugró-, elágaztató-, ciklusszervező-, hívó-, vezérlésátadó-, input/output- és egyéb utasítások.

Értékkopírozó utasítás: beállítja, módosítja a változók értékkomponensét.

Üres utasítás: hatására a processzor egy üres gépi utasítást hajt végre.

Ugró utasítás: a program egy adott pontjáról egy másik végrehajtó utasításra adhatjuk át a vezérlést.(GOTO)

Elágaztató utasítások

Kétirányú (feltételes): a program egy adott pontján 2 tevékenység közül választhatunk. (IF feltétel THEN tevékenység [ELSE tevékenység])

Többirányú: a program egy adott pontján egymást kizáró tevékenységek közül egyet hajt végre.

Ciklusszervező utasítások

Azt teszik lehetővé, hogy egy utasítást akárhányszor megismételhessünk.

3 részből áll: fej, mag, vég

Feltételes ciklus: az ismétlődést a feltételigaz vagy hamis értéke határozza meg. Létezik kezdőfeltételes (while), végfeltételes (do while).

Előírt lépésszámú ciklus: az ismétlődésre vonatkozó információk a ciklusfejben vannak.

Felsorolásos ciklus: az előző típus általánosítása.

Végtelen ciklus: nincs ismétlődésre vonatkozó információ sem a fejben, sem a végben.

Összetett ciklus: az előző 4 kombinációja. Bonyolult ciklusokat valósíthatunk meg velük.

Vezérlő utasítások

A C-ben 3 féle létezik: continue, break, return.

CONTINUE;

ciklus magjában alkalmazható, a ciklus magjában a többi utasítást nem hajtja végre.

BREAK;

A ciklust szabályosan befejezeti.

RETURN [kifejezés];

Szabályosan befejezeti a függvényt, majd a vezérlés átkerül a hívóhoz.

A program szövege programegységekre bontható.

Alprogramok

Az alprogram a meghatározója a programnak. Bemeneti adatcsoportot képez le kimeneti adatcsoportra. Az alprogram az újrafelhasználás eszköze. Akkor alkalmazhatjuk, ha a programrész megismétlődik. Ekkor ezt a részt kiemelhetjük, elég egyszer megírni, és az adott részeken csak hivatkozni kell rá.

4 komponensből áll: név, formális paraméterlista, törzs, környezet.

A név egy azonosító, a formális paraméter lista pedig a specifikáció része, itt szerepelnek az azonosítók, illetve megadható a paraméter futása közbeni viselkedést szabályozó információk. Lehet üres, ez az ún. paraméter nélküli alprogram. A törzsben a deklarációs és végrehajtó utasítások szerepelnek. Az itt szereplő változók lokális változók. Egy alprogram környezetét a globális változó együttese adja.

Az alprogramok két fajtáját különböztetjük meg: eljárás és függvény.

Az eljárásban valamilyen tevékenység hajtódik végre, híváskor ezen eredményt használjuk fel.

A függvény feladata pedig az, hogy egyetlen értéket határozzon meg, ez lehet tetszőleges típusú.

Előfordul, hogy a függvény megváltoztatja a paramétereit, vagy a környezetét, Ezt mellékhatásnak nevezzük.

Függvényt meghívni csak kifejezésben lehet, alakja: függvénynév(aktuális paraméter lista)

Hívási lánc, rekurzió

Egy programegység meghívhat egy másik programegységet, amely egy másikat, és így tovább. Ez a hívási lánc, melynek első tagja mindig a főprogram. Futás közben dinamikusan épül fel és bomlik le.

Ha aktív alprogramot hívunk meg, rekurzióról beszélünk. Ez kétféle lehet.

Közvetlen: egy alprogram önmagát hívja meg

Közvetett: korábban szereplő alprogramot hívunk meg.

Néhány nyelvben alprogramot meghívni lehet a törzsben is, ún. másodlagos belépési pontokat kialakítva. Így vagy a fejben megadott névvel, vagy a másodlagos belépési pont nevével lehet hivatkozni. Amennyiben a fejben keresztül lépünk be, az egész törzs végrehajtódik, másodlagos belépési ponton történő belépés során a törzs egy része hajtódik csak végre.

Paraméterkiértékelés

Az a folyamat, mikor egy alprogram hívásánál a formális- és aktuális paraméterek egymáshoz rendelődnek. Mindig a formális paraméter lista az elsődleges, melyet az alprogram specifikációja tartalmaz, és egy van belőle, tehát ez a meghatározó.

Paraméterátadás

Kommunikáció, mely az alprogram és más programegységek között zajlik. Mindig van egy hívó és egy hívott programegység.

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

Érték szerinti: a címkomponensek értéke átkerül a hívott program területén lefoglalt címkomponensre.

Cím szerinti: meghatározódik az aktuális paraméter címe, ez átadódik a hívott alprogramnak.

Érték-eredmény szerinti: meghatározódik az aktuális paraméter értéke és címe, majd ezek átkerülnek a hívóhoz.

Név szerinti: a formális paraméter nevének minden előfordulását átírja a program.

Szöveg szerinti: a név szerinti paraméterátadás egyik fajtája, akkor írja felül a paramétert, ha a neve először fordul elő.

Blokk

Olyan programegység, mely másik programegység belsejében helyezkedik el. Van kezdet, törzse, vége, bárhol elhelyezhető, ahol végrehajtó utasítás állhat.

INPUT/OUTPUT

Ezen a területen a programnyelvek nagyban eltérnek egymástól. Egyes nyelvekben nem is lehet megvalósítani, az implementációra bízzák a megoldást.

A perifériákkal történi kommunikációért felelős. Középpontban egy állomány áll.

Egy állomány funkció szerint lehet:

input állomány: csak olvasni lehet belőle

output állomány: csak írni lehet bele

input-output állomány: olvasni és írni is lehet.

Ha állományokkal akarunk dolgozni, végre kell hajtanunk pár lépést:

Deklaráció: a logikai állományt deklarálni kell, el kell látni megfelelő névvel és attribútumokkal.

Összerendelés: a logikai állománynak megfeleltetünk egy fizikai állományt.

Állomány megnyitása: csak akkor tudunk vele dolgozni, ha megnyitjuk. Ekkor operációs rendszer rutinok futnak le. Ekkor a program futása folyamán az állományt más-más funkcióra is megnyithatjuk.

Feldolgozás: ha megnyitottuk, írhatunk bele és olvashatunk belőle.

Lezáras: ismét op.rendszer rutinok futnak le. Ilyenkor történik meg a könyvtárak aktualizálása. O, I/O állományokat le kell zárnai, Input állományokat pedig illik lezárnai.

8.2. Programozás

[BMECPP]

A C++ a C nyelv továbbfejlesztése, a C nyelv „veszélyes” elemeit cseréli le, összességében letisztultabb, átláthatóbb kódot kapunk. Szintaktikája eltér a C nyelvtől, külön fordítóval fordítjuk.

C nyelvben például ha egy függvényt paraméter nélkül deklarálunk, akkor az a függvény tetszőleges számú paraméterrel hívható, míg C++-ban ez egy void paraméterrel rendelkező függvény lesz. Ez annyit jelent, hogy a függvény nem rendelkezik paraméterrel. Amennyiben nem adunk meg visszatérési típust, a C nyelvben int típusú változót ad vissza, míg C++-ban hibát fog jelezni a fordító.

A C++ nyelvben kétféle main függvény létezik. Az egyik verzióban előjön az argc, argv. Az előbbi a parancssori argumentumok számát, utóbbi a paraméterül kapott parancssori argumentumokat tárolja.

C++ nyelvben megjelenik a bool típus, ami egy igaz/hamis értékkel rendelkező változó.

A C nyelvvel ellentétben itt megtalálható beépített több-bájtos stringek, itt nem szükséges semmit sem includálni.

A változódeklarációk már mindenhol szerepelhetnek, deklarálhatunk változókat közvetlenül a használatuk előtt.

C++-ban lehetőségünk nyílik azonos nevű függvények deklarálására, ha az argumentumlistájuk alapján elkülöníthetőek. A fordító névfordítás technikájával különbözteti meg ezeket a függvényeket.

A függvények paramétereinek alapértelmezett értékeit is adhatunk, ez akkor jó, ha nem kapnak értéket. A C++-ban referencia szerint is adhatunk át paramétereket, mivel a referencia típus is megjelenik ebben a nyelvben. Referenciát úgy deklarálhatunk, az azonosítója elé egy & jelet illesztünk. Amennyiben a referencia értékét változtatjuk, úgy változik az eredeti változó értéke is.

Egységbe zárás

Az objektumorientált programozás egyik alapelve, jelentése egyszerű, bármiből lehet adatstruktúrát készíteni, és megadni a tulajdonságait. Ezeket másnéven osztályoknak hívjuk. Ezek rendelkeznek példányokkal, ezeket nevezzük objektumoknak, az objektumokon belül megjelenik az adatrejtés fogalma is, ami annyit jelent, hogy bizonyos elemek nem férhetnek hozzá adott elemekhez. Képbe jön az öröklődés fogalma is, mely során speciális osztályok kerülnek létrehozásra, amelyek rendelkeznek az alap osztály tulajdonságával. Ezzel új objektumot hozunk létre. Ha példányosítunk egy adott osztályt, memória foglalás történik. Az osztályon belüli változók, illetve függvények a tagváltozók és tagfüggvények, utóbbi a struktúrán belül és kívül is deklarálhatjuk, viszont a tagfüggvények csak egy előfordulásánál történik memória foglalás. A private: kulcsszó után történik az adatrejtés, ekkor az itt deklarált változók és függvények csak ebben a részben lesznek láthatóak, a program többi részével nem állnak kapcsolatban, így nem lesznek használhatók a private-n kívül. Egy osztály példányosításakor inicializálhatjuk az objektumot, ezt konstruktur segítségével tehetjük. Ez egy függvény, amely az objektum példányosításakor minden esetben meghívódik magától. A fordítóprogram is képes ellátni konstruktort a programot, amennyiben mi nem írnunk külön, viszont nem fog történni benne semmi. Saját kezűleg több konstruktort is megírhatunk. A konstruktornak létezik párja, ezt destruktornak hívjuk. A destruktort is képes a fordítóprogram megírni alapértelmezetten, viszont ebben sem történik ilyenkor semmi. Ez a függvény az objektum megszűnése után hívódik meg, memóriát szabadít fel.

C++-ban megjelenik a new és a delete függvény. Előbbi memória foglalásra, utóbbi memória felszabadításra képes. A new függvényben a konstruktur paramétereit meg kell adnunk.

A konstruktornak létezik egy tipikus példája, a másoló konstruktor. Ez egy már meglévő objektumot másol bele egy új objektumba. Egy apró módosítással elérhető, hogy a private: részben szereplő változókat és függvényeket elérjük a private: részen kívül. Ezt úgy tehetjük meg, hogy a függvény/változó elé beszűrjuk a friend szót, viszont csak korlátozottan férhetünk a védet elemekhez hozzá, és ritkán használjuk. Az osztályokon beszélünk statikus változókról, ezek az adott osztályhoz tartoznak, és akkor is használhatók, ha még az osztályból még egy példány sem került példányosításra. Léteznek statikus tagfüggvények is, melyeket objektum nélkül is lehet használni, viszont csak a statikus részeket éri el. Az osztályokon belül adhatunk meg típusdefiníciókat.

A C++ nyelvben a matematikai műveleteket operátorok használatával végezhetünk. Ezek az operátorok be vannak építve, viszont túlterheléssel funkciójukat megváltoztathatjuk. Megjelent néhány új operátor is, pl a pointer-tag operátor. Az operátorokra mint függvényekre kell tekintenünk.

C++-ban bekérésre és kiíratásra adatfolyamok állnak rendelkezésünkre, melyeket a << és a >> operátorokkal tudunk használni. Kiíratásra a cout adatfolyamot használjuk, majd utána << operátort használunk. A bekérésre a cin jelent megoldást, a >> operátor használatával. A beolvasási limitet meg lehet változtatni az ignore függvényvel. Az adatfolyamok rendszerhívásokat használnak, ezért buffereket hoznak létre ezekhez a rendszerhívásokhoz. Ezeket a buffereket a flush függvényvel lehet kiüríteni, de akár hibaüzenetet is küldhetünk, ekkor a beolvasáskor nem történik semmi. A stringek olvasásánál kicsit más a helyzet. Amennyiben szóközzel elválasztott karakterláncokat tartalmaz, a getline függvényel tudjuk ezeket beolvasni. Az adatfolyamok tagfüggvényekkel rendelkeznek. Ezeket speciális objektumokkal, ún. manipulátorokkal tudjuk megváltoztatni, egy egyszerű példa ezekre a speciális objektumokra az endl, mely megtöri az adott sort. Az állományból olvasás, állományba írás is adatfolyamokon keresztül működik, ilyenkor az ifstream és ofstream adatfolyamot használjuk, melyeket meg kell nyitni és be is kell zárni, ez történhet konstruktor-destruktur párossal is, viszont létezik ezek megvalósítására open és close függvény.

A hibakezelés kivételkezeléssel történik. Amint egy kivételt érzékel a program, a vezérlés a kivételkezelőre ugrik. Ennek megvalósítása a try-catch blokkal történik. A try blokkban található utasítások minden végrehajtónak, ha nem talált hibát. Amennyiben viszont találunk hibát, azt eldobjuk a throw függvényel, és az elkapást a catch függvényre bízzuk. Ezeket a blokkokat egymásba is ágyazhatjuk, ezzel megvalósítva a különböző szinteken történő hibakezelést. A throw rendelkezhet üres paraméterlistával is, ekkor a kivétel újra dobható lesz. A 190. oldalon található egy program, ez a program egy szám reciprokát adja vissza. Ebben a programban alkalmazzuk a try catch blokkot. A könyvben a 197. oldalon található még egy példa, melyben az ún. verem visszacsévélése folyamat található meg. Ez gyakorlatban annyit jelent, hogy eldobjuk a kivételt, utána az elkapásig egy hívási láncban haladunk felfelé, és a lokális függvények változóit felszabadítjuk.

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.

9. fejezet

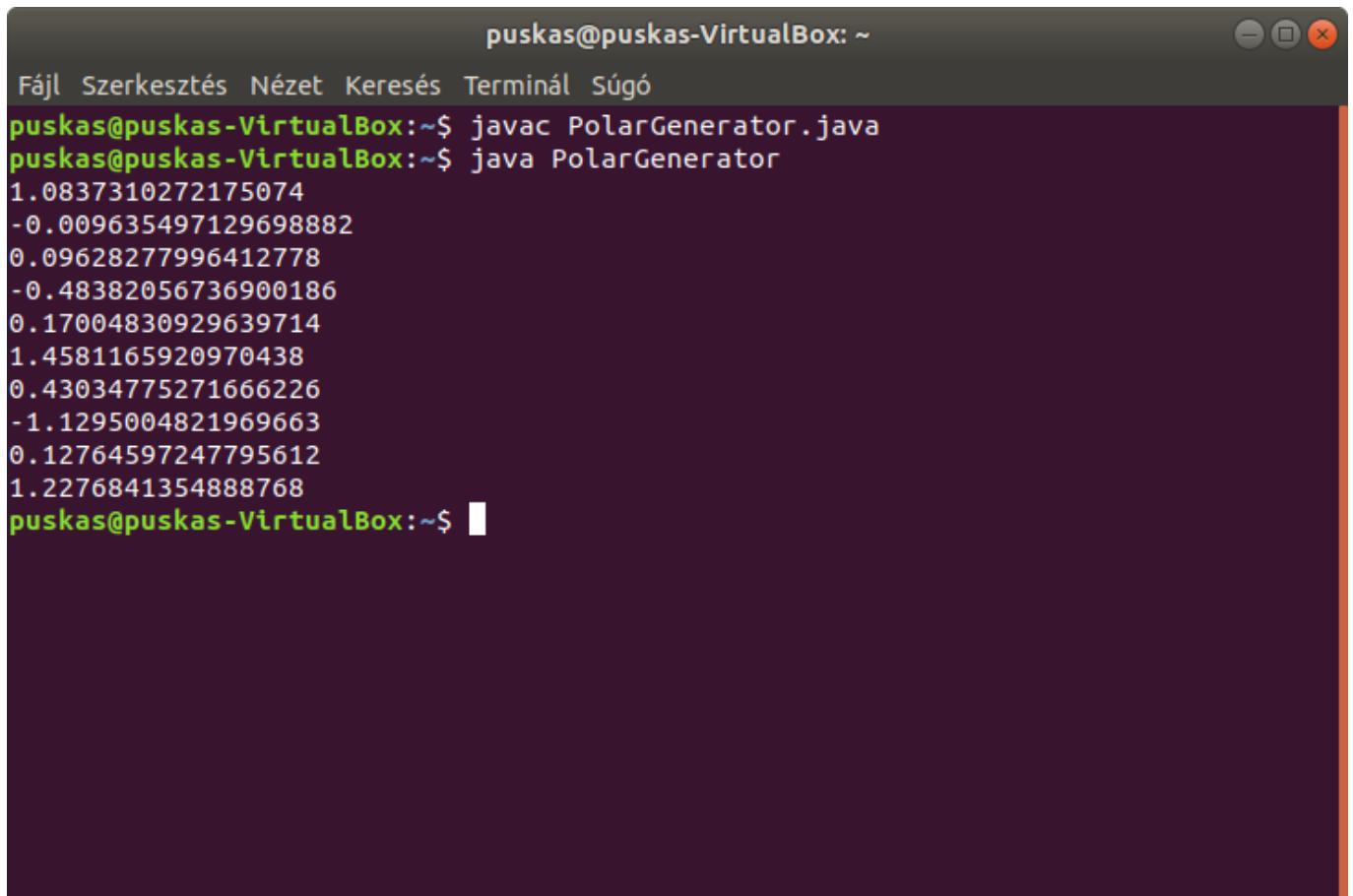
Helló, Arroway!

9.1. OO szemlélet

Megoldás forrása: <https://gitlab.com/puskasmate/bhax/blob/master/Prog2/00/PolarGenerator.java>

Az objektumorientált programozási nyelvekben osztályokat hozunk létre, ezeknek az osztályoknak az elemeit pedig objektumoknak nevezzük. minden objektum rendelkezik az osztály összes tulajdonságával. Ez azért kényelmes, mivel elég csak az osztály létrehozásakor leírni a tulajdonságait, amivel az összes létrehozott objektum rendelkezni fog.

A program első részében létrehozunk egy PolarGenerator osztályt, amely két változóval és egy függvénytel fog rendelkezni. A nincsTarolt változó értéke fogja meghatározni, hogy van e változó eltárolva. A függvény megvizsgálja, hogy van e változó eltárolva, ha nincs, generál 2 random számot, az egyiket eltárolja, a másikat pedig visszaadja, ha volt változó eltárolva akkor pedig az eltárolt változót adja vissza. A feltételes utasítás minden ágában negálja a nincsTarolt változó értékét.



A screenshot of a terminal window titled "puskas@puskas-VirtualBox: ~". The window shows the command "javac PolarGenerator.java" being run, followed by the output of the "PolarGenerator" class. The output consists of several random double values generated by the class's method.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ javac PolarGenerator.java
puskas@puskas-VirtualBox:~$ java PolarGenerator
1.0837310272175074
-0.009635497129698882
0.09628277996412778
-0.48382056736900186
0.17004830929639714
1.4581165920970438
0.43034775271666226
-1.1295004821969663
0.12764597247795612
1.2276841354888768
puskas@puskas-VirtualBox:~$
```

Forráskód:

```
public class PolarGenerator {

    boolean nincsTarolt = true;
    double tarolt;

    public PolarGenerator() {
        nincsTarolt = true;
    }

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

```
    } else {
        nincsTarolt = !nincsTarolt;
        return tarolt;
    }
}

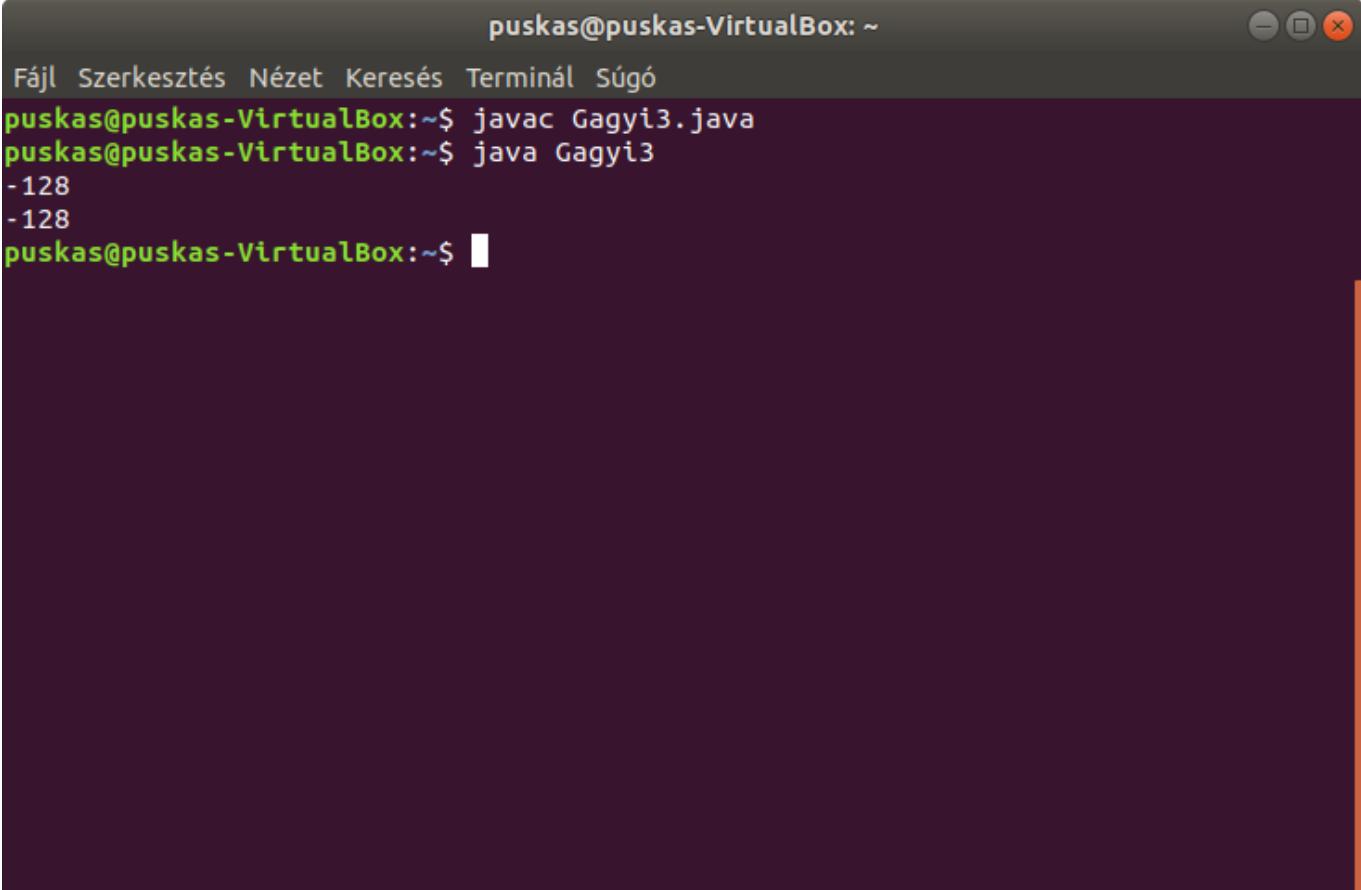
public static void main (String[] args){
    PolarGenerator g= new PolarGenerator();
    for (int i=0;i < 10;i++){
        System.out.println(g.kovetkezo());
    }
}
```

9.2. "Gagyi"

Megoldás forrása: <https://gitlab.com/puskasmate/bhax/tree/master/Prog2/Gagyi>

Feladatunk a "while ($x \leq t \& \& x \geq t \& \& t \neq x$)" tesztkérdésen alapul. Itt egyik esetben x és t objektum által hordozott érték, a másik esetben pedig az objektum címe. Az Integer típus -128-127 tartományba eső számokat egy bufferbe tölti be, és ebben a bufferben lévő számok azonos memóriacímet kapnak. Ezen intervallumba eső értékek használatakor tehát ugyanaz a memóriacím kerül betöltésre a bufferból, így ha -128-tól kisebb vagy 127-től nagyobb értéket adunk meg, be tudunk lépni a ciklusba, mivel a két szám memóriacíme nem fog megegyezni, de a két szám mégis megegyezik.

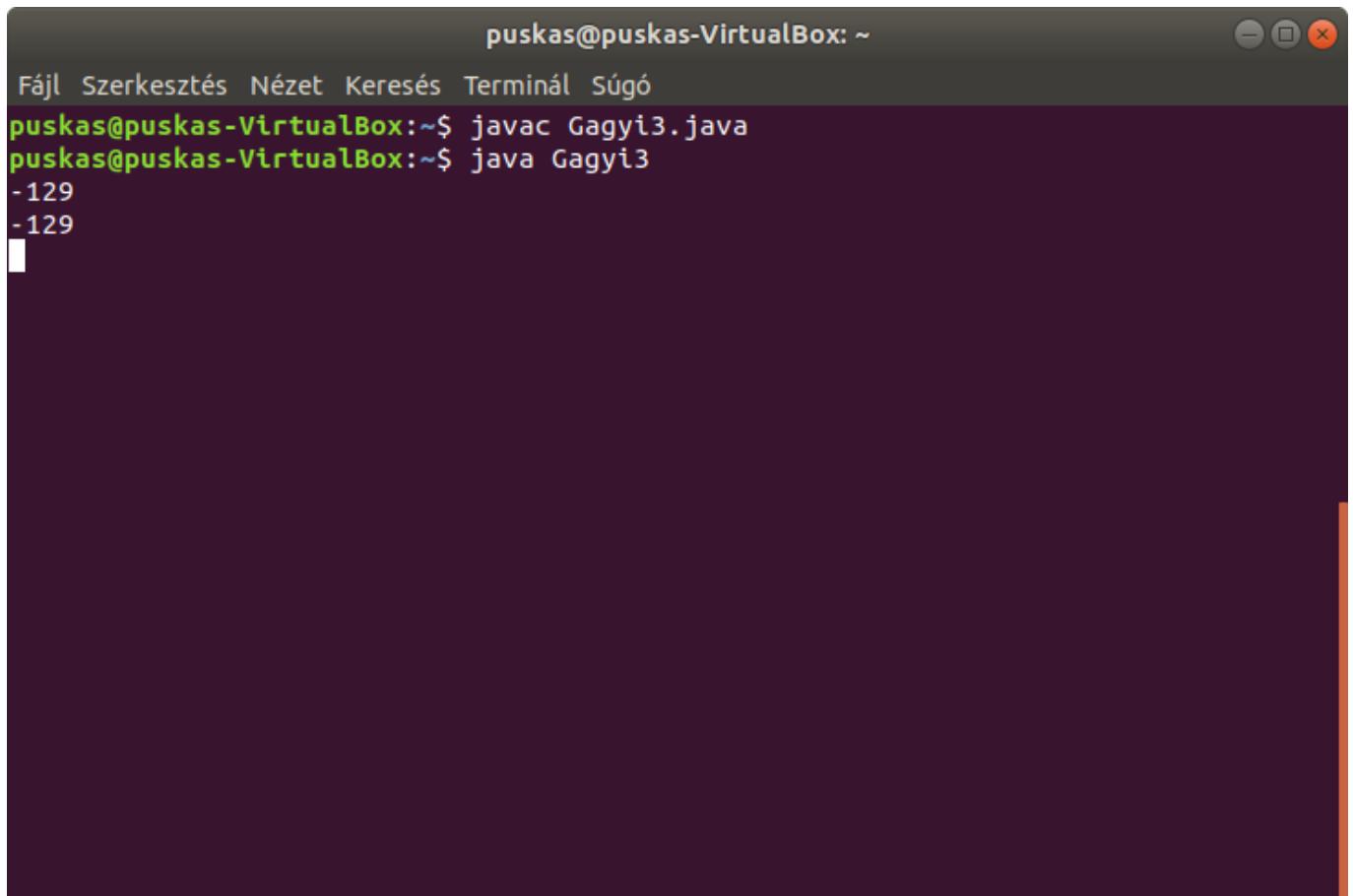
-128 esetén tehát nem lép be a ciklusba



A screenshot of a terminal window titled "puskas@puskas-VirtualBox: ~". The window has standard Linux-style window controls (minimize, maximize, close) in the top right corner. The terminal's background is dark, and the text is white. The user has run the command "javac Gagyi3.java" followed by "java Gagyi3". The output shows two lines of "-128" followed by a prompt at the bottom.

```
puskas@puskas-VirtualBox:~$ javac Gagyi3.java
puskas@puskas-VirtualBox:~$ java Gagyi3
-128
-128
puskas@puskas-VirtualBox:~$
```

-129 esetén pedig belépünk a ciklusba.



A screenshot of a terminal window titled "puskas@puskas-VirtualBox: ~". The window has a dark purple background. The terminal output is as follows:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ javac Gagyi3.java
puskas@puskas-VirtualBox:~$ java Gagyi3
-129
-129
```

9.3. Yoda

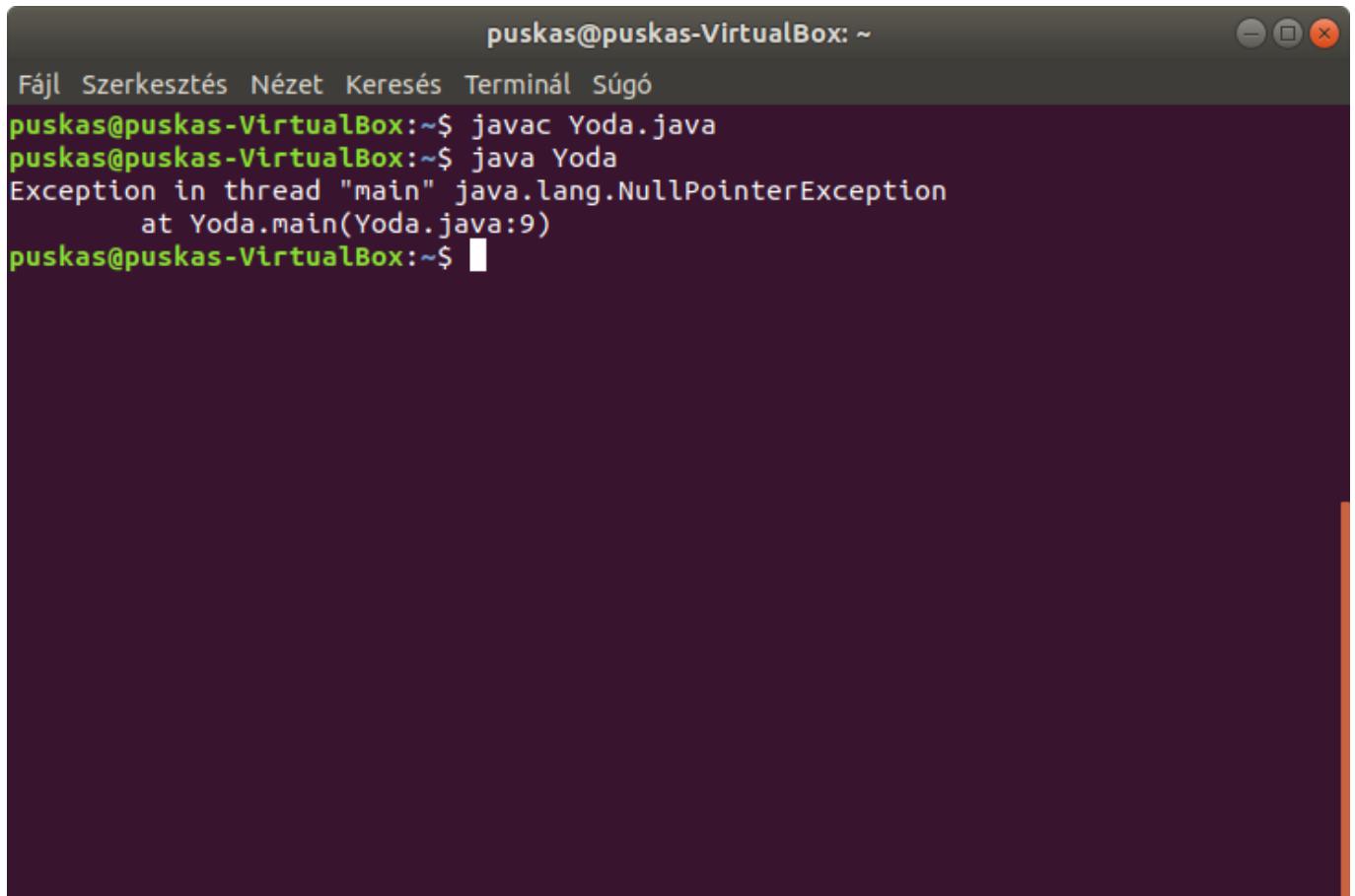
Megoldás forrása:<https://gitlab.com/puskasmate/bhax/tree/master/Prog2/Yoda>

A Star Wars című film híres karaktere, Yoda az átlagostól eltérő szórendet alkalmaz. A feladatunk az volt, hogy írjuk olyan programot, ami ha nem fordított szórendet használ, java.lang.NullPointerException-nel leáll.

```
public class Yoda
{
    public static void main (String []args)
    {
        String yodaa = null;
        if (yodaa.equals("Zeusz")) {
            System.out.println("ASDASDas");
        }else{
            System.out.println("Benzema");
        }
    }
}
```

```
}
```

Láthatjuk, hogy a java.lang.NullPointerException hibaüzenetet kapjuk vissza a program futtatásakor, ha nem követjük a programban a Yoda conditiont, mivel az equals nullreferenciára hibát ad.

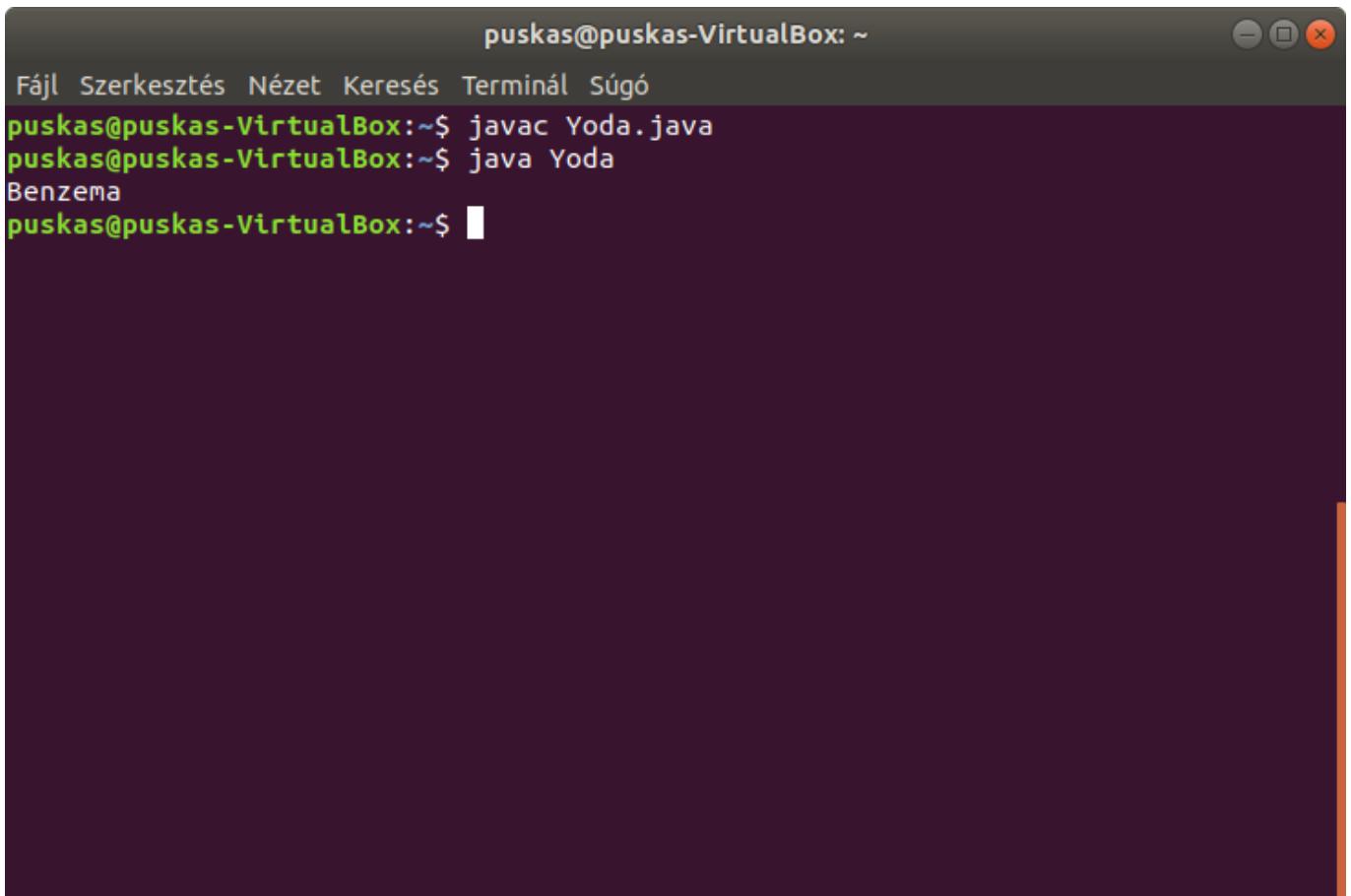


The screenshot shows a terminal window titled "puskas@puskas-VirtualBox: ~". It displays the following command-line session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ javac Yoda.java
puskas@puskas-VirtualBox:~$ java Yoda
Exception in thread "main" java.lang.NullPointerException
    at Yoda.main(Yoda.java:9)
puskas@puskas-VirtualBox:~$
```

```
public class Yoda
{
    public static void main (String[]args)
    {
        String yodaa = null;
        if ("Zeusz".equals(yodaa)) {
            System.out.println("ASDASDas");
        }else{
            System.out.println("Benzema");
        }
    }
}
```

Ez a forráskód viszont futtatható, mivel követi a Yoda conditiont.



A screenshot of a terminal window titled "puskas@puskas-VirtualBox: ~". The window shows the following command-line session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ javac Yoda.java
puskas@puskas-VirtualBox:~$ java Yoda
Benzema
puskas@puskas-VirtualBox:~$
```

9.4. Kódolás from scratch

Megoldás forrása:https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#pi_jegyei

```
public class PiBBP {  
  
    String d16PiHexaJegyek;  
  
    public PiBBP(int d) {  
  
        double d16Pi = 0.0d;  
  
        double d16S1t = d16Sj(d, 1);  
        double d16S4t = d16Sj(d, 4);  
        double d16S5t = d16Sj(d, 5);  
        double d16S6t = d16Sj(d, 6);  
  
        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;  
  
        d16Pi = d16Pi - StrictMath.floor(d16Pi);  
  
        StringBuffer sb = new StringBuffer();
```

```
Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};  
  
while(d16Pi != 0.0d) {  
  
    int jegy = (int)StrictMath.floor(16.0d*d16Pi);  
  
    if(jegy<10)  
        sb.append(jegy);  
    else  
        sb.append(hexaJegyek[jegy-10]);  
  
    d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);  
}  
  
d16PiHexaJegyek = sb.toString();  
}  
  
public double d16Sj(int d, int j) {  
  
    double d16Sj = 0.0d;  
  
    for(int k=0; k<=d; ++k)  
        d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);  
  
  
    return d16Sj - StrictMath.floor(d16Sj);  
}  
  
public long n16modk(int n, int k) {  
  
    int t = 1;  
    while(t <= n)  
        t *= 2;  
  
    long r = 1;  
  
    while(true) {  
  
        if(n >= t) {  
            r = (16*r) % k;  
            n = n - t;  
        }  
  
        t = t/2;  
  
        if(t < 1)  
            break;  
    }  
}
```

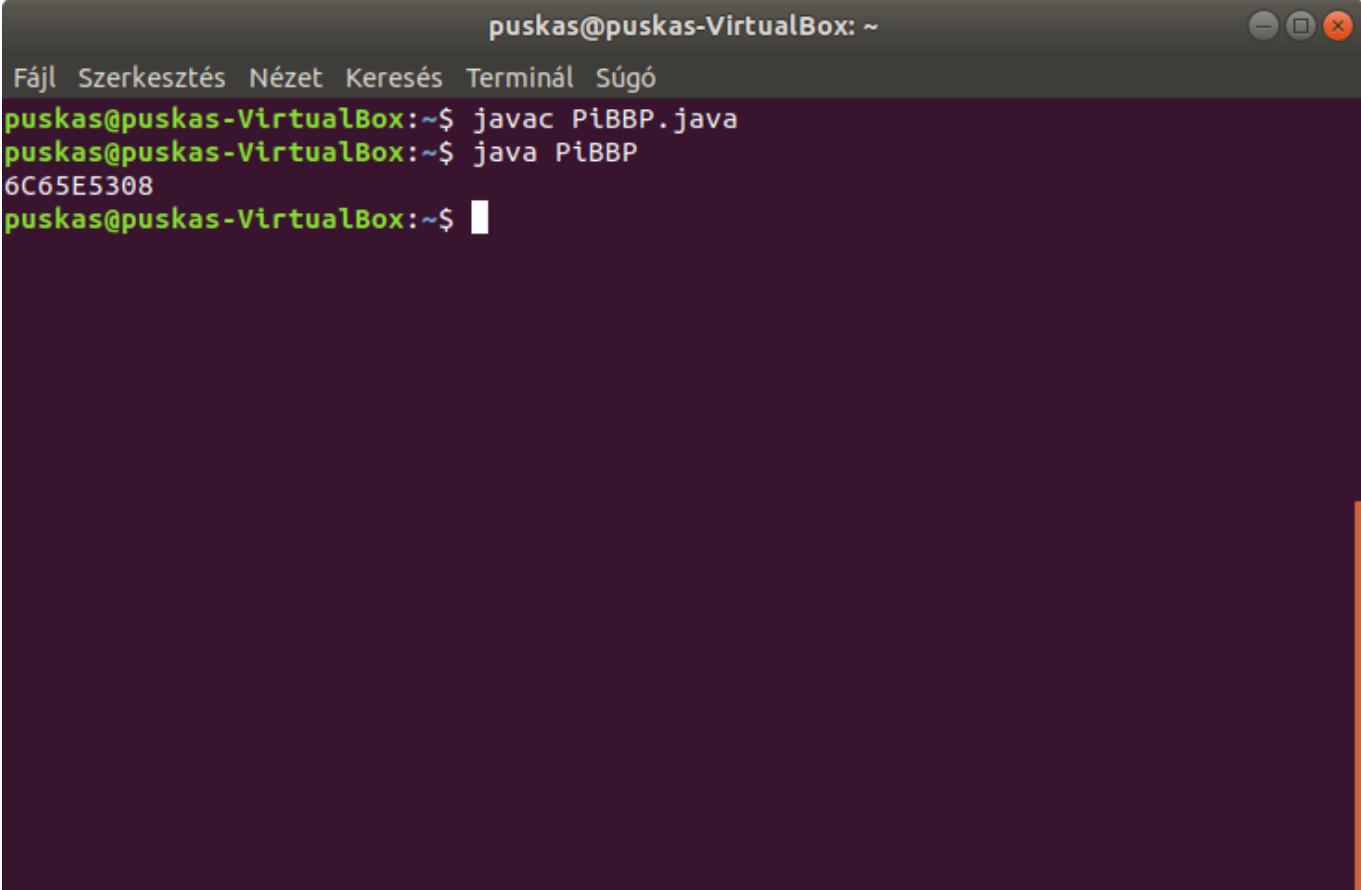
```
r = (r*r) % k;  
}  
  
return r;  
}  
  
public String toString() {  
  
    return d16PiHexaJegyek;  
}  
  
public static void main(String args[]) {  
    System.out.print(new PiBBP(1000000));  
    System.out.println("");  
}  
}
```

A Bailey-Borwein-Plouffe formula a π kiszámítására létrehozott algoritmus. 1995-ben Simon Plouffe fedezte fel, és a publikált tanulmány szerzőiről nevezte el.

A program a pi szám hexadecimális értékét fogja visszaadni, és az alábbi algoritmus alapján fog működni.

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

A program sikeresen kiszámítja a π hexadecimális értékét.



A screenshot of a terminal window titled "puskas@puskas-VirtualBox: ~". The window has a dark background and light-colored text. At the top, there are icons for minimizing, maximizing, and closing the window. The terminal shows the following command-line session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ javac PiBBP.java
puskas@puskas-VirtualBox:~$ java PiBBP
6C65E5308
puskas@puskas-VirtualBox:~$
```

10. fejezet

Helló, Liskov!

10.1. Liskov helyettesítés sértése

Megoldás forrása:

Feladatunk arról szólt, hogy írunk olyan programot, mely a Liskov helyettesítési elvet (Liskov Substitution Principle) sérti meg. A Liskov helyettesítési elv az objektumorientált programozási nyelvekben jelenik meg. Az elv a következő: legyen T egy osztály, S pedig legyen T-nek egy alosztálya. Azokon a helyeken, ahol T-t használunk, használhatunk S-t is anélkül, hogy a programrészben bármi változás történne, tehát a T típusú objektumok helyén használhatunk S típusú objektumokat is.

Lássuk a programot C++-ban:

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

Létrehozunk egy Madar nevű osztályt, ami egy repul függvényvel fog rendelkezni, mely a madarakat fogja reptetni.

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

A Program nevű osztályunk egy Madar objektumot fog reptetni a repül függvény segítségével.

```
class Sas : public Madar  
{};  
  
class Pingvin : public Madar // ezt úgy is lehet/kell olvasni, hogy a ←  
    pingvin tud repülni  
{};
```

Létrehozunk egy Sas és egy Pingvin nevű alosztályt. Ez a két osztály a Madar osztály alosztályai lesznek.

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

    Sas sas;
    program.fgv ( sas );

    Pingvin pingvin;
    program.fgv ( pingvin ); // sérül az LSP, mert a P::fgv röptetné a ←
                             // Pingvint, ami ugye lehetetlen.

}
```

A main függvényben madarakat reptetünk, viszont láthatjuk, hogy Pingvin típusú objektumra is alkalmazzuk a repul függvényt, viszont köztudott, hogy a pingvinek bár madarak, nem tudnak repülni.

Programunk Java-ban majdnem megegyezik, csak a szintaktikában kellett néhány változtatást elvégezni.

```
class Madar {
    public

        void repul() {

    }

    static class Program {

        void fgv(Madar madar) {
            madar.repul();
        }
    }

    static class Sas extends Madar

    {
    }

    static class Pingvin extends Madar

    {
```

```
}

;

public static void main(String[] args) {

    Program program=new Program();
    Madar madar=new Madar();
    program.fgv(madar);

    Sas sas=new Sas();
    program.fgv(sas);

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

}
}
```

10.2. Szülő-gyerek

Megoldás forrása:

Feladatunk az volt, hogy szemléltettük, hogy habár egy osztály (nevezzük szülőnek) alosztály (nevezzük gyereknek) örökli, azaz rendelkezik a szülőosztály adataival, viszont fordítva ez nem lesz igaz, tehát a szülőosztály nem fog rendelkezni a gyerekosztály adataival.

Megvalósítás C++-ban:

```
#include <iostream>

class Szulo{
public:
    bool dolgozik;
    Szulo() {
        bool dolgozik=true;
    };
};
```

Deklaráljuk a szülőosztályt, amely jelen esetben csak egy adattal fog rendelkezni, méghozzá azzal, hogy az adott szülő dolgozik.

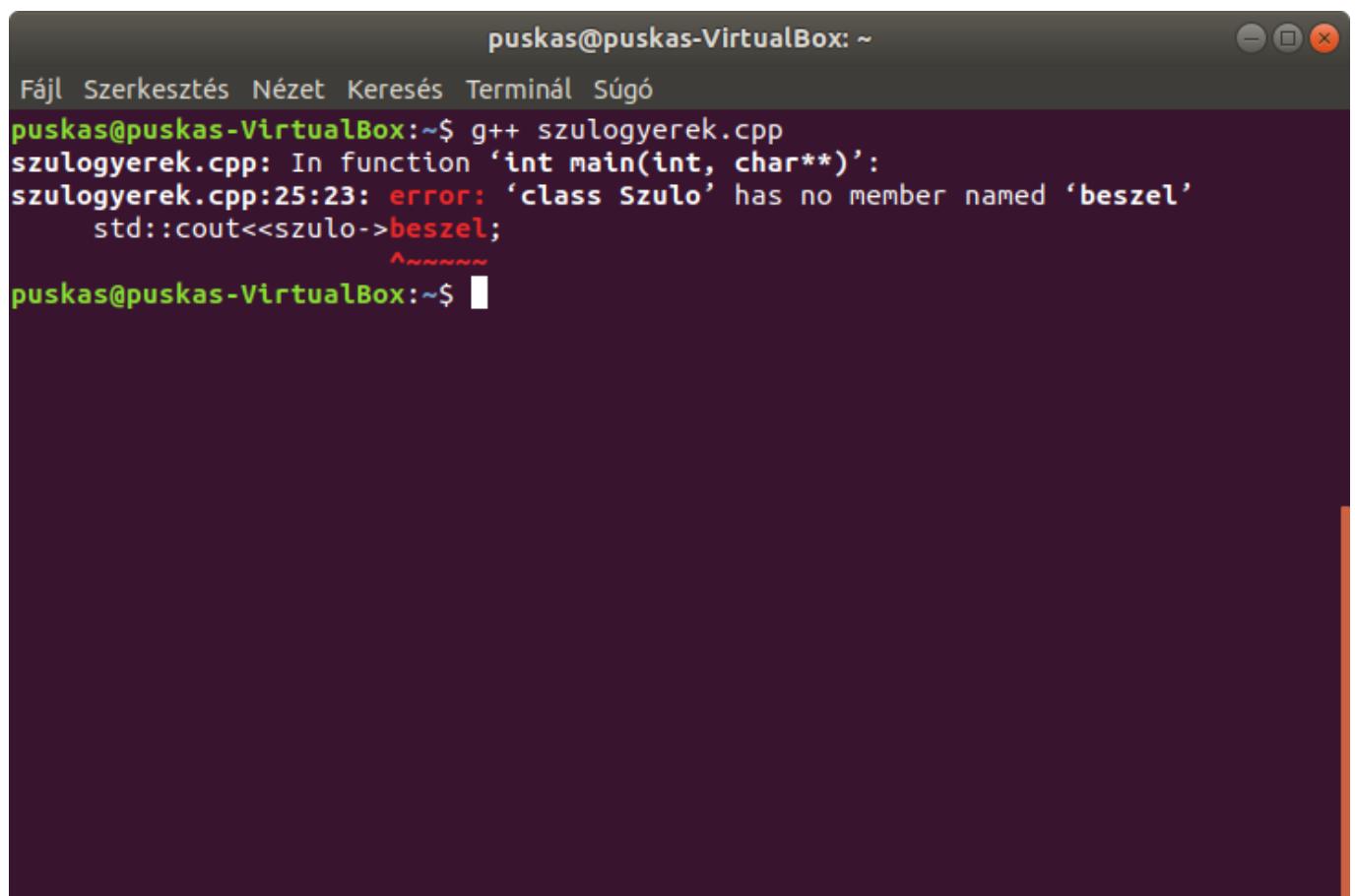
```
class Gyerek:public Szulo{
public:
    bool beszel;
    Gyerek() {
        bool beszel=false;
```

```
    };  
};
```

A Szulo osztály alosztályaként deklaráljuk a Gyerek osztályt, ami egy beszel nevű logikai változóval fog rendelkezni, ez jelzi, hogy az adott gyerek még nem tud beszélni.

```
int main( int argc, char **argv ) {  
    Szulo* szulo=new Szulo;  
    Gyerek* gyerek=new Gyerek;  
  
    std::cout<<szulo->beszel;  
}
```

A main függvényben ki szeretnénk iratni, hogy az adott szülő beszél-e, viszont habár a gyerek osztály igen, a szülő osztály nem rendelkezik beszel paraméterrel, ezáltal a program nem fog lefordulni.



The screenshot shows a terminal window titled "puskas@puskas-VirtualBox: ~". The window has standard Linux-style window controls at the top right. The terminal content is as follows:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
puskas@puskas-VirtualBox:~$ g++ szulogyerek.cpp  
szulogyerek.cpp: In function ‘int main(int, char**)’:  
szulogyerek.cpp:25:23: error: ‘class Szulo’ has no member named ‘beszel’  
    std::cout<<szulo->beszel;  
                           ^~~~~~  
puskas@puskas-VirtualBox:~$
```

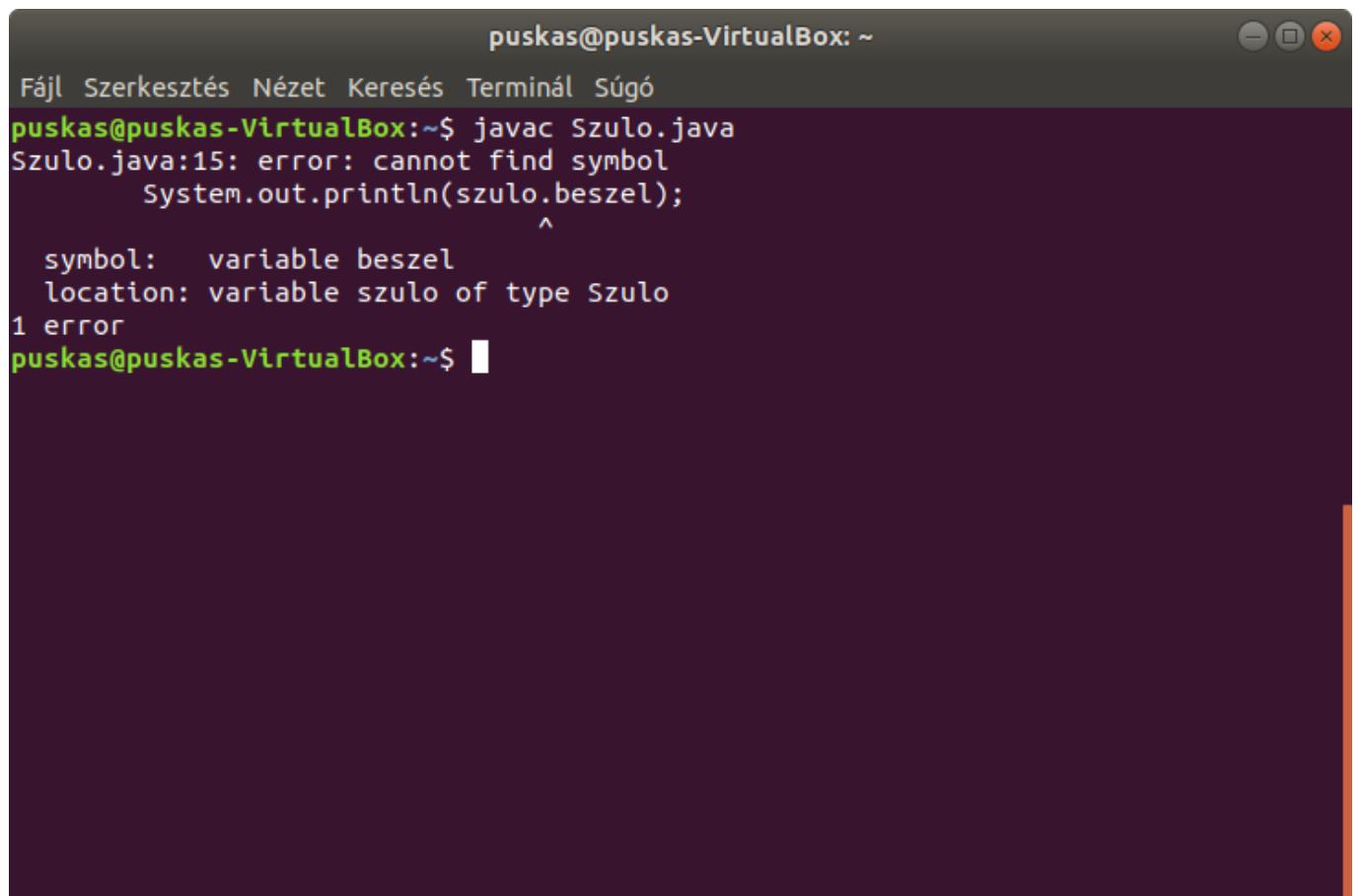
Java megoldás:

```
class Szulo {  
  
    boolean dolgozik=true;  
  
    static class Gyerek extends Szulo{
```

```
        boolean beszel=false;
    }
    public static void main(String[] args) {
        Szulo szulo=new Szulo();
        Gyerek gyerek=new Gyerek();

        System.out.println(szulo.beszel);

    }
}
```



The screenshot shows a terminal window titled "puskas@puskas-VirtualBox: ~". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ javac Szulo.java
Szulo.java:15: error: cannot find symbol
    System.out.println(szulo.beszel);
                           ^
      symbol:   variable beszel
      location: variable szulo of type Szulo
1 error
puskas@puskas-VirtualBox:~$
```

10.3. Anti OO

Megoldás forrása:<https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat-apas03.html#id561066>

Feladatunk az volt, hogy a BPP algoritmussal határozzuk meg a Pi hexadecimális értékének 3 számjegyét. Ez a 3 számjegy a 10^6 . helyen lévő, a 10^7 . helyen lévő, és a 10^8 . helyen lévő számjegy. Ezt a programot 4 nyelven kellett elészítenünk, majd pedig összevetni a futási időket, amit egy táblázat segít szemléltetni a forráskódok és a futásról készült képek alatt.

A program C nyelven:

```
#include <stdio.h>
#include <math.h>
#include <time.h>
/*
 * pi_bbp_bench.c
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 * A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
 * a PiBBPBench osztályt, amit pedig átírtuk C nyelvre.
 *
 */
long
n16modk (int n, int k)
{
    long r = 1;

    int t = 1;
    while (t <= n)
        t *= 2;

    for (;;)
    {
        if (n >= t)
        {
            r = (16 * r) % k;
            n = n - t;
        }

        t = t / 2;

        if (t < 1)
            break;

        r = (r * r) % k;
    }

    return r;
}
```

```
/* {16^d Sj}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
double
d16Sj (int d, int j)
{

    double d16Sj = 0.0;
    int k;

    for (k = 0; k <= d; ++k)
        d16Sj += (double) n16modk (d - k, 8 * k + j) / (double) (8 * k + j);

    /*
        for(k=d+1; k<=2*d; ++k)
        d16Sj += pow(16.0, d-k) / (double)(8*k + j);
    */

    return d16Sj - floor (d16Sj);
}

/*
 * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
main ()
{
    double d16Pi = 0.0;

    double d16S1t = 0.0;
    double d16S4t = 0.0;
    double d16S5t = 0.0;
    double d16S6t = 0.0;

    int jegy;
    int d;

    clock_t delta = clock ();
```

A programban az alábbi helyen szükséges módosítanunk, hogy a különböző hatványokon kapott számjegyeket visszakapjuk.

```
for (d = 1000000; d < 1000001; ++d)
```

Ez a prefix a 10^6 . számjegyet adja vissza, amennyiben ezen változtatni szeretnénk, a 0-k számát növeljük 1-gyel, így visszakapjuk a 10^7 . számjegyet és így tovább. Mind a 4 nyelven megírt porgramban ugyanitt és ugyanígy kell végrehajtanunk a módosítást, így ezt külön többször nem emelném ki.

```
{  
  
    d16Pi = 0.0;  
  
    d16S1t = d16Sj (d, 1);  
    d16S4t = d16Sj (d, 4);  
    d16S5t = d16Sj (d, 5);  
    d16S6t = d16Sj (d, 6);  
  
    d16Pi = 4.0 * d16S1t - 2.0 * d16S4t - d16S5t - d16S6t;  
  
    d16Pi = d16Pi - floor (d16Pi);  
  
    jegy = (int) floor (16.0 * d16Pi);  
  
}  
  
printf ("%d\n", jegy);  
delta = clock () - delta;  
printf ("%f\n", (double) delta / CLOCKS_PER_SEC);  
}
```

Egy kép a futásról:

```
puskas@puskas-VirtualBox: ~  
Fájl Szerkesztés Nézet Keresés Terminál Súgó  
puskas@puskas-VirtualBox:~$ gcc PiBPP.c -o PiBPP -lm  
PiBPP.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]  
main ()  
^~~~  
puskas@puskas-VirtualBox:~$ ./PiBPP  
6  
1.914375  
puskas@puskas-VirtualBox:~$ gcc PiBPP.c -o PiBPP -lm  
PiBPP.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]  
main ()  
^~~~  
puskas@puskas-VirtualBox:~$ ./PiBPP  
7  
23.011836  
puskas@puskas-VirtualBox:~$ gcc PiBPP.c -o PiBPP -lm  
PiBPP.c:77:1: warning: return type defaults to ‘int’ [-Wimplicit-int]  
main ()  
^~~~  
puskas@puskas-VirtualBox:~$ ./PiBPP  
12  
262.417018  
puskas@puskas-VirtualBox:~$
```

Megoldás C++-ban:

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <iostream>
/*
 * pi_bbp_bench.c
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 * A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
 * a PiBBPBench osztályt, amit pedig átírtuk C nyelvre.
 *
 */
long
n16modk (int n, int k)
{
    long r = 1;

    int t = 1;
    while (t <= n)
        t *= 2;

    for (;;)
    {
        if (n >= t)
        {
            r = (16 * r) % k;
            n = n - t;
        }

        t = t / 2;

        if (t < 1)
            break;

        r = (r * r) % k;
    }

    return r;
}
```

```
/* {16^d Sj}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
double
d16Sj (int d, int j)
{

    double d16Sj = 0.0;
    int k;

    for (k = 0; k <= d; ++k)
        d16Sj += (double) n16modk (d - k, 8 * k + j) / (double) (8 * k + j);

    /*
        for(k=d+1; k<=2*d; ++k)
            d16Sj += pow(16.0, d-k) / (double)(8*k + j);
    */

    return d16Sj - floor (d16Sj);
}

/*
 * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
 * [BBP ALGORITMUS] David H. Bailey: The
 * BBP Algorithm for Pi. alapján.
 */
main ()
{
    double d16Pi = 0.0;

    double d16S1t = 0.0;
    double d16S4t = 0.0;
    double d16S5t = 0.0;
    double d16S6t = 0.0;

    int jegy;
    int d;

    clock_t delta = clock ();

    for (d = 1000000; d < 1000001; ++d)
    {

        d16Pi = 0.0;

        d16S1t = d16Sj (d, 1);
        d16S4t = d16Sj (d, 4);
```

```
d16S5t = d16Sj (d, 5);
d16S6t = d16Sj (d, 6);

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

d16Pi = d16Pi - floor (d16Pi);

jegy = (int) floor (16.0 * d16Pi);

}

std::cout<<jegy<<std::endl;
delta = clock () - delta;
std::cout<<((double) delta / CLOCKS_PER_SEC)<<std::endl;
}
```

Futásról készült kép:

The screenshot shows a terminal window titled "puskas@puskas-VirtualBox: ~". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ g++ PiBPP.cpp -o PiBPP
puskas@puskas-VirtualBox:~$ ./PiBPP
6
1.95192
puskas@puskas-VirtualBox:~$ g++ PiBPP.cpp -o PiBPP
puskas@puskas-VirtualBox:~$ ./PiBPP
7
22.648
puskas@puskas-VirtualBox:~$ g++ PiBPP.cpp -o PiBPP
puskas@puskas-VirtualBox:~$ ./PiBPP
12
264.164
puskas@puskas-VirtualBox:~$
```

Megoldás Java-ban:

```
public class PiBBPBench {

    public static 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 - Math.floor(d16Sj);
}

public static long n16modk(int n, int k) {

    int t = 1;
    while(t <= n)
        t *= 2;

    long r = 1;

    while(true) {

        if(n >= t) {
            r = (16*r) % k;
            n = n - t;
        }

        t = t/2;

        if(t < 1)
            break;

        r = (r*r) % k;
    }

    return r;
}

public static void main(String args[]) {

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
    double d16S6t = 0.0d;

    int jegy = 0;

    long delta = System.currentTimeMillis();

    for(int d=1000000; d<1000001; ++d) {
```

```
d16Pi = 0.0d;

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

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

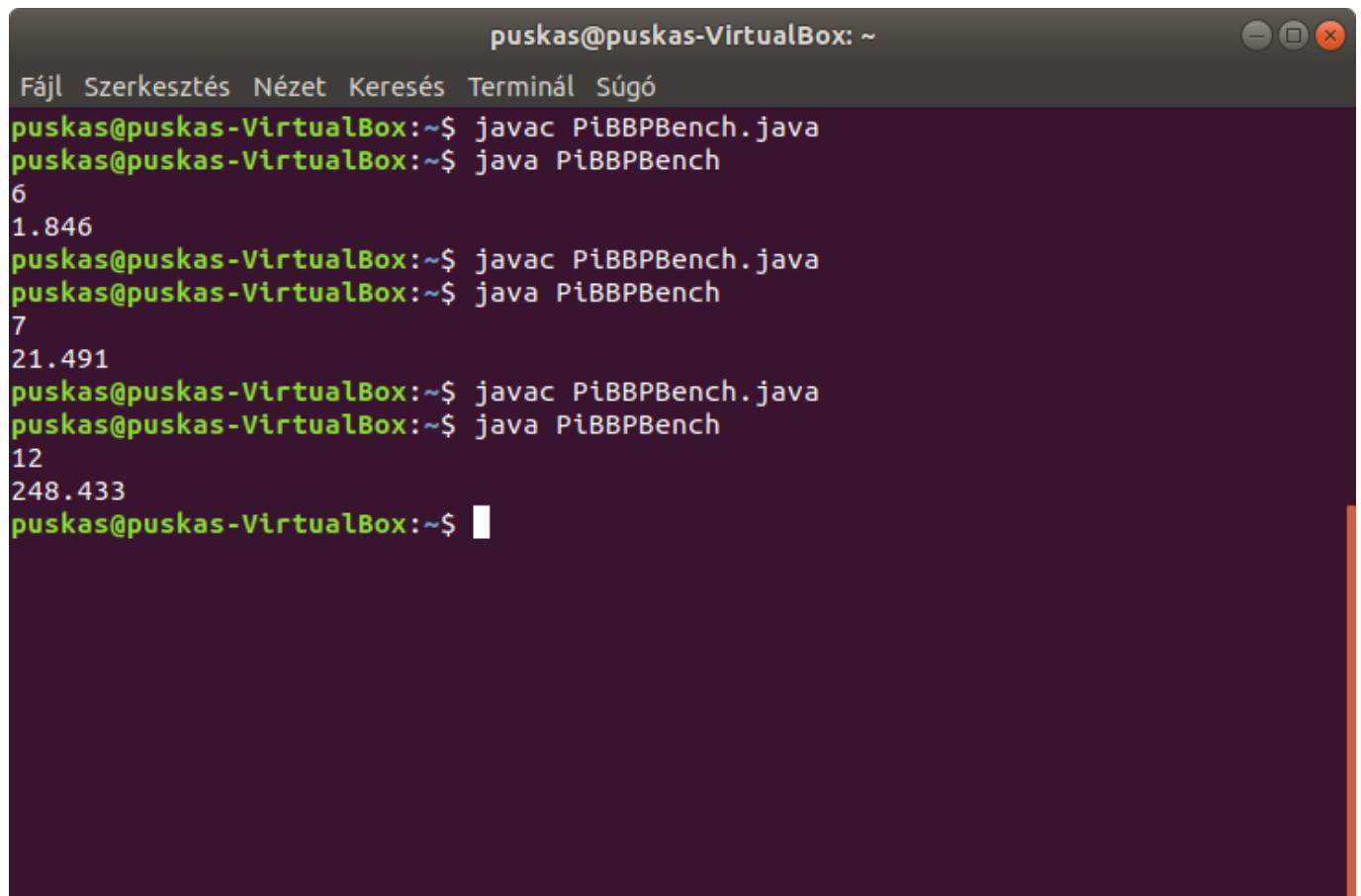
d16Pi = d16Pi - Math.floor(d16Pi);

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

}

System.out.println(jegy);
delta = System.currentTimeMillis() - delta;
System.out.println(delta/1000.0);
}
}
```

A futásról készült kép:



The screenshot shows a terminal window titled "puskas@puskas-VirtualBox: ~". The window has standard Linux-style window controls (minimize, maximize, close) at the top right. The terminal interface includes a menu bar with "Fájl", "Szerkesztés", "Nézet", "Keresés", "Terminál", and "Súgó". The command line shows the user's session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ javac PiBBPBench.java
puskas@puskas-VirtualBox:~$ java PiBBPBench
6
1.846
puskas@puskas-VirtualBox:~$ javac PiBBPBench.java
puskas@puskas-VirtualBox:~$ java PiBBPBench
7
21.491
puskas@puskas-VirtualBox:~$ javac PiBBPBench.java
puskas@puskas-VirtualBox:~$ java PiBBPBench
12
248.433
puskas@puskas-VirtualBox:~$
```

Megoldás C#-ban:

```
/*
 * FileName: PiBBPBench.cs
 * Author: Bátfai Norbert, nbatfai@inf.unideb.hu
 * DIGIT 2005, Javat tanítok
 */
/// <summary>
/// A PiBBPBench C# átirata.
/// </summary>
/// <remark>
/// A PiBBP.java-ból kivettük az "objektumorientáltságot", így kaptuk
/// a PiBBPBench osztályt, amit pedig átírtuk C# nyelvre.
///
/// (A PiBBP osztály a BBP (Bailey–Borwein–Plouffe) algoritmust a Pi hexa
/// jegyeinek számolását végző osztály. A könnyebb olvahatóság
/// kedvéért a változó és metódus neveket megpróbáltuk az algoritmust
/// bemutató [BBP ALGORITMUS] David H. Bailey: The BBP Algorithm for Pi.
/// cikk jelöléseihez.)
/// </remark>
public class PiBBPBench {
    /// <remark>
    /// BBP algoritmus a Pi-hez, a [BBP ALGORITMUS] David H. Bailey: The
    /// BBP Algorithm for Pi. alapján a {16^d Sj} részlet kiszámítása.
    /// </remark>
    /// <param>
    /// d a d+1. hexa jegytől számoljuk a hexa jegyeket
    /// </param>
    /// <param>
    /// j Sj indexe
    /// </param>
    public static 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);

        /*
        for(int k=d+1; k<=2*d; ++k)
            d16Sj += System.Math.pow(16.0d, d-k) / (double)(8*k + j);
        */

        return d16Sj - System.Math.Floor(d16Sj);
    }
    /// <summary>
    /// Bináris hatványozás mod k, a 16^n mod k kiszámítása.
    /// </summary>
    /// <param>
    /// n kitevő
    /// </param>
    /// <param>
```

```
/// k    modulus
/// </param>
public static 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;
}
/// <remark>
/// A [BBP ALGORITMUS] David H. Bailey: The
/// BBP Algorithm for Pi. alapján a
/// {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
/// kiszámítása, a {} a törtrészt jelöli. A Pi hexa kifejtésében a
/// d+1. hexa jegytől
/// </remark>
public static void Main(System.String[] args) {

    double d16Pi = 0.0d;

    double d16S1t = 0.0d;
    double d16S4t = 0.0d;
    double d16S5t = 0.0d;
    double d16S6t = 0.0d;

    int jegy = 0;

    System.DateTime kezd = System.DateTime.Now;

    for(int d=1000000; d<1000001; ++d) {
```

```
d16Pi = 0.0d;

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

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

d16Pi = d16Pi - System.Math.Floor(d16Pi);

jegy = (int)System.Math.Floor(16.0d*d16Pi);

}

System.Console.WriteLine(jegy);
System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
}
}
```

Futásról készült kép:

The screenshot shows a terminal window titled "puskas@puskas-VirtualBox: ~". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ mcs PiBPP.cs
puskas@puskas-VirtualBox:~$ mono PiBPP.exe
6
1,77732
puskas@puskas-VirtualBox:~$ mcs PiBPP.cs
puskas@puskas-VirtualBox:~$ mono PiBPP.exe
7
20,987595
puskas@puskas-VirtualBox:~$ mcs PiBPP.cs
puskas@puskas-VirtualBox:~$ mono PiBPP.exe
12
236,722284
puskas@puskas-VirtualBox:~$ █
```

Láthatjuk, hogy a C# végzett az első helyen futási időt tekintve.

| Nyelv | 10^6 | 10^7 | 10^8 |
|-------|----------|-----------|------------|
| C | 1.914375 | 23.011836 | 262.417018 |
| C++ | 1.95192 | 22.648 | 264.164 |
| C# | 1.77732 | 20.987595 | 236.722284 |
| Java | 1.846 | 21.491 | 248.433 |

10.1. táblázat. A táblázat:

10.4. Ciklomatikus komplexitás

A ciklomatikus komplexitás Thomas J. McCabe nevéhez fűződik, ugyanis 1976-ban ő publikálta ezt a metrikát, ami egy forráskód komplexitását, azaz bonyolultságát fejezi ki számban kifejezve. A számítás a gráfelméleten alapszik. A számítás képlete a következő:

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

ahol E a gráf éleinek a száma, N a gráfban lévő csúcsok száma és P az összefüggő komponensek száma. A ciklomatikus számot a következőképp számoljuk ki: $M=E-N+P$.

Feladatunk az volt, hogy számoljuk ki valamelyik programunk ciklomatikus komplexitását. A feladatban a korábban már használt "binfa.c" nevű program ciklomatikus komplexitását vizsgáljuk meg. Több program is képes a komplexitás kiszámolására illetve kiírására. Jelen esetben a gnu complexity-hez fordulunk segítségért. Ez egy C állományok komplexitását kiszámoló program, melyet terminálból a következő parancs beírásával tudunk beszerezni:

```
sudo apt-get install complexity
```

A programok komplexitását pedig a következőképp tudjuk megnézni:

```
complexity --histogram --score --thresh=0 'allomany_neve.c'
```

A "binfa.c" forráskód komplexitása:

```
puskas@puskas-VirtualBox: ~
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-VirtualBox:~$ complexity --histogram --score --thresh=0 'binfa.c'
Complexity Scores
Score | ln-ct | nc-lns| file-name(line): proc-name
      1       6       6  binfa.c(115): szabadit
      1       8       7  binfa.c(20): uj_elem
      2      13      13  binfa.c(97): kiir
      6      51      41  binfa.c(37): main

Complexity Histogram
Score-Range Lin-Ct
 0-9          67 ****
Scored procedure ct:        4
Non-comment line ct:       67
Average line score:        4
25%-ile score:            2 (75% in higher score procs)
50%-ile score:            6 (half in higher score procs)
75%-ile score:            0 (25% in higher score procs)
Highest score:             6 (main() in binfa.c)
puskas@puskas-VirtualBox:~$
```

Láthatjuk, hogy a programban 4 függvény szerepel. Ez a négy függvény a szabadit, az uj_elem, a kiir és a main függvény. Mellettük láthatjuk a komplexitásuk értékét.

11. fejezet

Helló, Mandelbrot!

11.1. Reverse engineering UML osztálydiagram

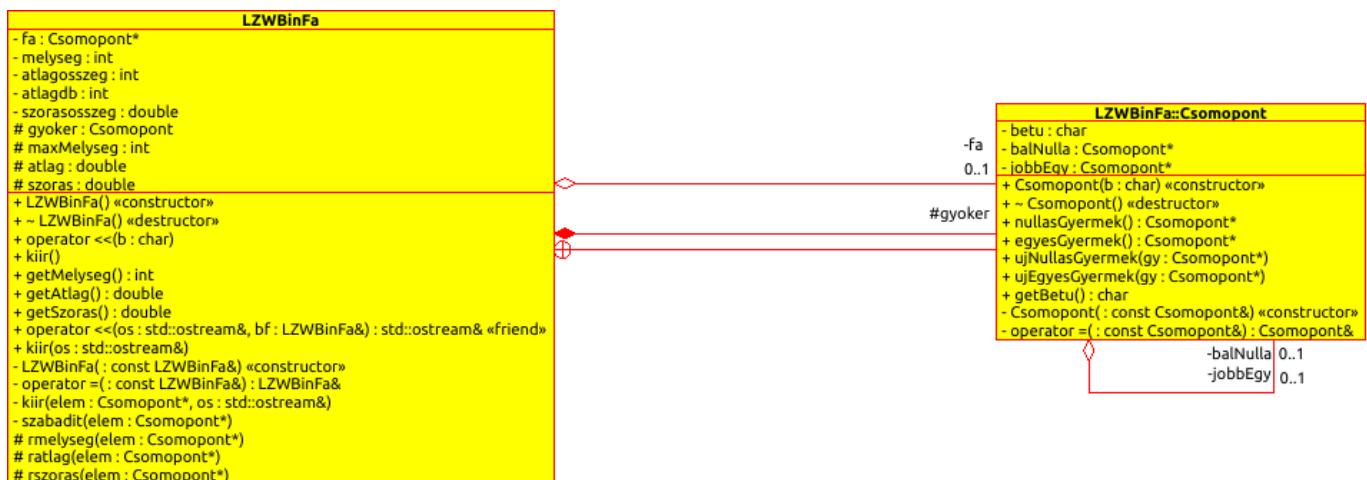
A Reverse engineering annyit tesz, hogy egy már meglévő forráskóból készítünk UML osztálydiagramot. A forráskóban előforduló osztályokat egy téglalappal ábrázoljuk, amit 3 részre bontunk: az osztály neve, az osztály attribútumai és az osztály függvényei, eljárásai.

Feladatunk az volt, hogy szerkesszük meg az UML osztálydiagramját az első védési programunknak, a binfának (z3a7.cpp).

Az osztálydiagram megszerkesztéséhez az Umbrello nevű programot használjuk, amely könnyen beszerezhető az alábbi parancs beírásával:

```
sudo apt install umbrello
```

A program telepítése után a kódimportáló varázsló segítségével a forráskód alapján létrehozzuk az osztálydiagramot, ami a következőképp fog kinézni:



Láthatjuk, hogy a két osztály amit kaptunk összeköttetésben áll egymással. 2 összeköttetési típust különbözötünk meg.

Aggregáció és kompozíció. Az aggregáció ún. "rész-egész" kapcsolat. Azt jelenti, hogy az egyik objektumban megtalálható a másik objektum is. Láthatjuk, hogy az `LZWBinFa` osztályban megtalálható az

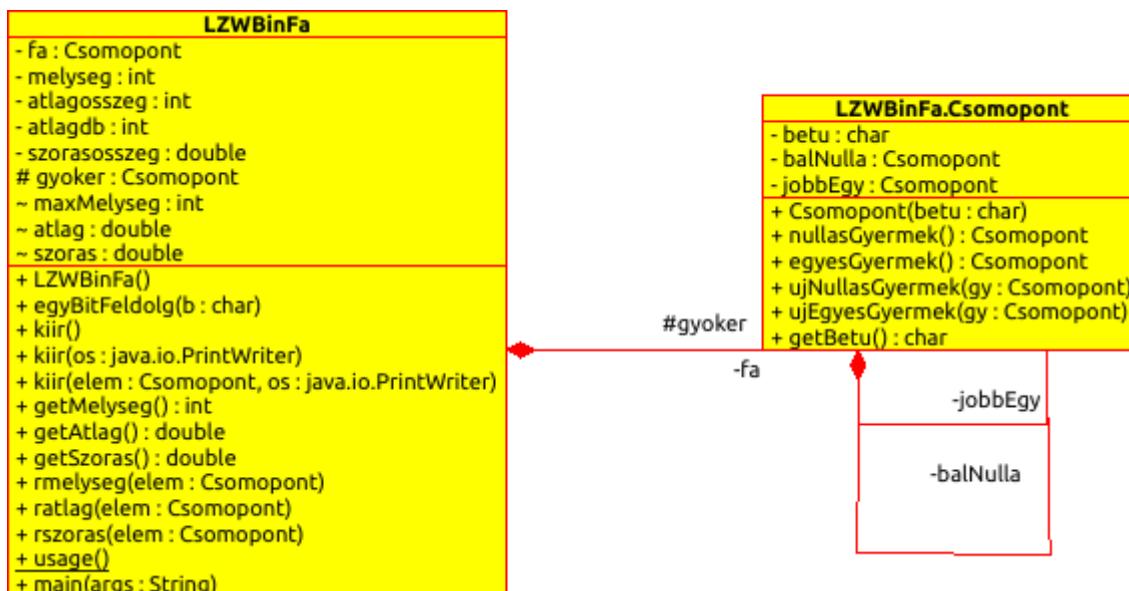
LZWBinFa::Csomopont is. Az aggregációk két fajtája van. A gyenge aggregáció (az egyenes vonal végen üres rombusz) és az erős aggregáció (telt rombusz), vagy másnéven kompozíció. A különbség a két alfaj között egyszerű. A gyenge aggregációban az az osztály, amelyik tartalmazza a másikat, létezhet a tartalmazott osztály nélkül is. Az erős aggregációban viszont a tartalmazó osztály nem létezhet a tartalmazott osztály nélkül (gyökér nélkül nincs fa).

11.2. Forward engineering UML osztálydiagram

A Forward engineering a Reverse engineering másik változata, ilyenkor ugyanis nem forráskódból állítunk elő osztálydiagramot, hanem osztálydiagramból állítunk elő forráskódot. Feladatunk az volt tehát, hogy készítsünk egy osztálydiagramot, majd pedig generálunk belőle egy forráskódot.

A feladat megoldására az előző feladatban használt Umbrello programot használtam. A fentebb lévő program java átíratát használtam fel, abból generáltam egy osztálydiagramot, és a diagram alapján a kódgenerálási varázsló segítségével generáltam java forráskódot.

Az osztálydiagram, amely alapján a forráskód készült:



Az ebből kapott forráskód pedig a következő:

```

import java.io.PrintWriter;
import LZWBinFa.Csomopont;

/**
 * Class LZWBinFa
 */
public class LZWBinFa {

    //
    // Fields
    //
  
```

```
protected Csomopont gyoker;
private Csomopont fa;
private int melyseg;
private int atlagosszeg;
private int atlagdb;
private double szorasosszeg;

//  
// Constructors  
//  
public LZWBinFa () { };  
  
//  
// Methods  
//  
  
//  
// Accessor methods  
//  
  
/**  
 * Set the value of gyoker  
 * @param newVar the new value of gyoker  
 */  
protected void setGyoker (Csmopont newVar) {  
    gyoker = newVar;  
}  
  
/**  
 * Get the value of gyoker  
 * @return the value of gyoker  
 */  
protected Csmopont getGyoker () {  
    return gyoker;  
}  
  
/**  
 * Set the value of fa  
 * @param newVar the new value of fa  
 */  
private void setFa (Csmopont newVar) {  
    fa = newVar;  
}  
  
/**  
 * Get the value of fa  
 * @return the value of fa  
 */  
private Csmopont getFa () {
```

```
    return fa;
}

/**
 * Set the value of melyseg
 * @param newVar the new value of melyseg
 */
private void setMelyseg (int newVar) {
    melyseg = newVar;
}

/**
 * Get the value of melyseg
 * @return the value of melyseg
 */
private int getMelyseg () {
    return melyseg;
}

/**
 * Set the value of atlagosszeg
 * @param newVar the new value of atlagosszeg
 */
private void setAtlagosszeg (int newVar) {
    atlagosszeg = newVar;
}

/**
 * Get the value of atlagosszeg
 * @return the value of atlagosszeg
 */
private int getAtlagosszeg () {
    return atlagosszeg;
}

/**
 * Set the value of atlagdb
 * @param newVar the new value of atlagdb
 */
private void setAtlagdb (int newVar) {
    atlagdb = newVar;
}

/**
 * Get the value of atlagdb
 * @return the value of atlagdb
 */
private int getAtlagdb () {
    return atlagdb;
}
```

```
/**  
 * Set the value of szorasosszeg  
 * @param newVar the new value of szorasosszeg  
 */  
private void setSzorasosszeg (double newVar) {  
    szorasosszeg = newVar;  
}  
  
/**  
 * Get the value of szorasosszeg  
 * @return the value of szorasosszeg  
 */  
private double getSzorasosszeg () {  
    return szorasosszeg;  
}  
  
//  
// Other methods  
//  
  
/**  
 */  
public void LZWBinFa()  
{  
}  
  
/**  
 * @param b  
 */  
public void egyBitFeldolg(char b)  
{  
}  
  
/**  
 */  
public void kiir()  
{  
}  
  
/**  
 * @param os  
 */  
public void kiir(java.io.PrintWriter os)  
{  
}
```

```
/***
 * @param elem
 * @param os
 */
public void kiir(LZWBinFa.Csomopont elem, java.io.PrintWriter os)
{
}

/***
 * @return int
 */
public int getMelyseg()
{
}

/***
 * @return double
 */
public double getAtlag()
{
}

/***
 * @return double
 */
public double getSzoras()
{
}

/***
 * @param elem
 */
public void rmelyseg(LZWBinFa.Csomopont elem)
{
}

/***
 * @param elem
 */
public void ratlag(LZWBinFa.Csomopont elem)
{}
```

```
/** @param elem */
public void rszoras(LZWBInFa.Csomopont elem)
{
}

/** */
public static void usage()
{
}

/** @param args */
public static void main(String args)
{
}

}
```

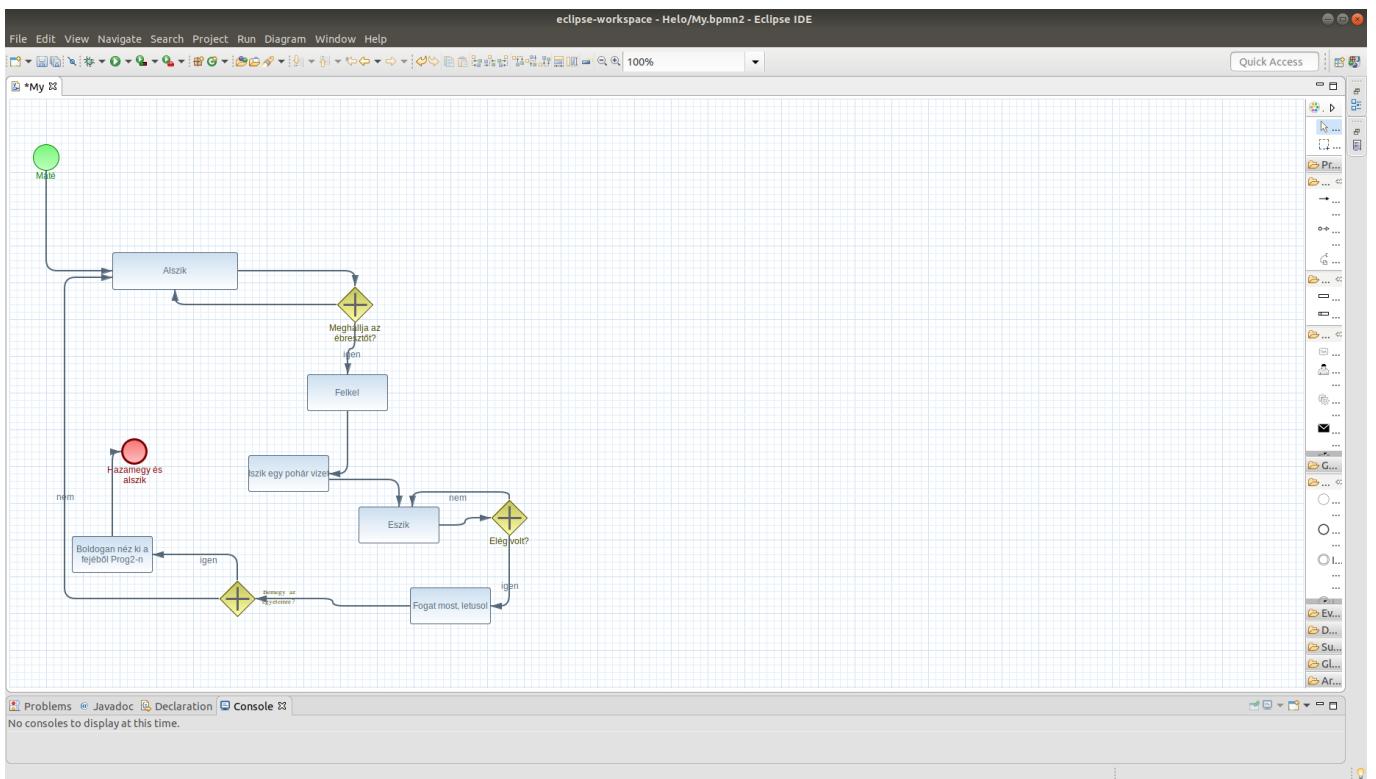
A feladat tökéletesen szemlélteti, hogy hosszabb programok (mint például a binfa) esetében egy ilyen megoldás nagyon hasznos lehet, ugyanis jobban átláthatóbb a forráskód, és az osztályok attribútumai és függvényei is definiálva vannak, már csak a viselkedésüket kell megadnunk.

11.3. BPMN

A BPMN (Business Process Model and Notation) üzleti folyamatok grafikus reprezentálására kötött létre. A nyelvet üzleti folyamatok modellezésére használják, könnyebbé teszi a folyamatok megfelelő sorrendbe szervezését és megértését. Feladatunk az volt, hogy BPMN-ben egy folyamatot készítsünk.

A feladat megoldása több féle program segítségével is történhet. Ebben az esetben először is fel kell telepítenünk az eclipse-t. Következő lépés a BPMN plugin feltelepítése, ezután pedig új projekt létrehozásakor BPMN projektet kell kiválasztanunk.

A feladat egy nagyon lebutított hétfői napomat írja le:



Láthatjuk, hogy a BPMN segítségével teljesen egyértelműen meg tudjuk állapítani, mely folyamat után melyik másik folyamat jön, illetve teljesen átlátható egy ilyen ábra egy átlagember számára is.

12. fejezet

Helló, Chomsky!

12.1. ENCODING

Feladatunk az volt, hogy küszöböljük ki a fordítási hibákat a linkelt MandelbrotHalmazNagyító.java forráskódban.

A feladatban forrásként megadott zip letöltése után veszteséges tömörítésbe ütközünk. Az állományok neve ugyanis olvashatatlan, így első dolgunk az állományok átnevezése volt.

A

```
javac MandelbrotHalmazNagyító.java
```

parancs kiadása után errorok sorát kapjuk vissza.

```
puskas@puskas-Aspire-A515-51G: ~/Maa
Fájl Szerkesztés Nézet Keresés Terminál Súgó
UTF-8
    // vizsgáljuk egy adott pont iteraciót:
    ^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xE1) for encoding
UTF-8
    // vizsgáljuk egy adott pont iteraciót:
    ^
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xF3) for encoding
UTF-8
    // vizsgáljuk egy adott pont iteraciót:
    ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xE9) for encoding
UTF-8
    // Az egérmutató pozícija
    ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xF3) for encoding
UTF-8
    // Az egérmutató pozícija
    ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xED) for encoding
UTF-8
    // Az egérmutató pozícija
    ^
100 errors
```

Láthatjuk, hogy a fordító a kódolásra panaszkodik, ugyanis a javac alapból utf-8-as betűkészletet használ, a forráskód betűkészlete tehát minden bizonnal eltér ettől a betűkészlettől.

Az ékezetes betűk viszont több nyelvben is megjelennek, illetve több betűkészlet is engedi ezek használatát.

Tanár Úr által linkelt könyvben egy kevés keresés után megtalálható a példaprogram leírása, és a következő kép nagyban leszűkíti a megfelelő betűkészlet megtalálását:

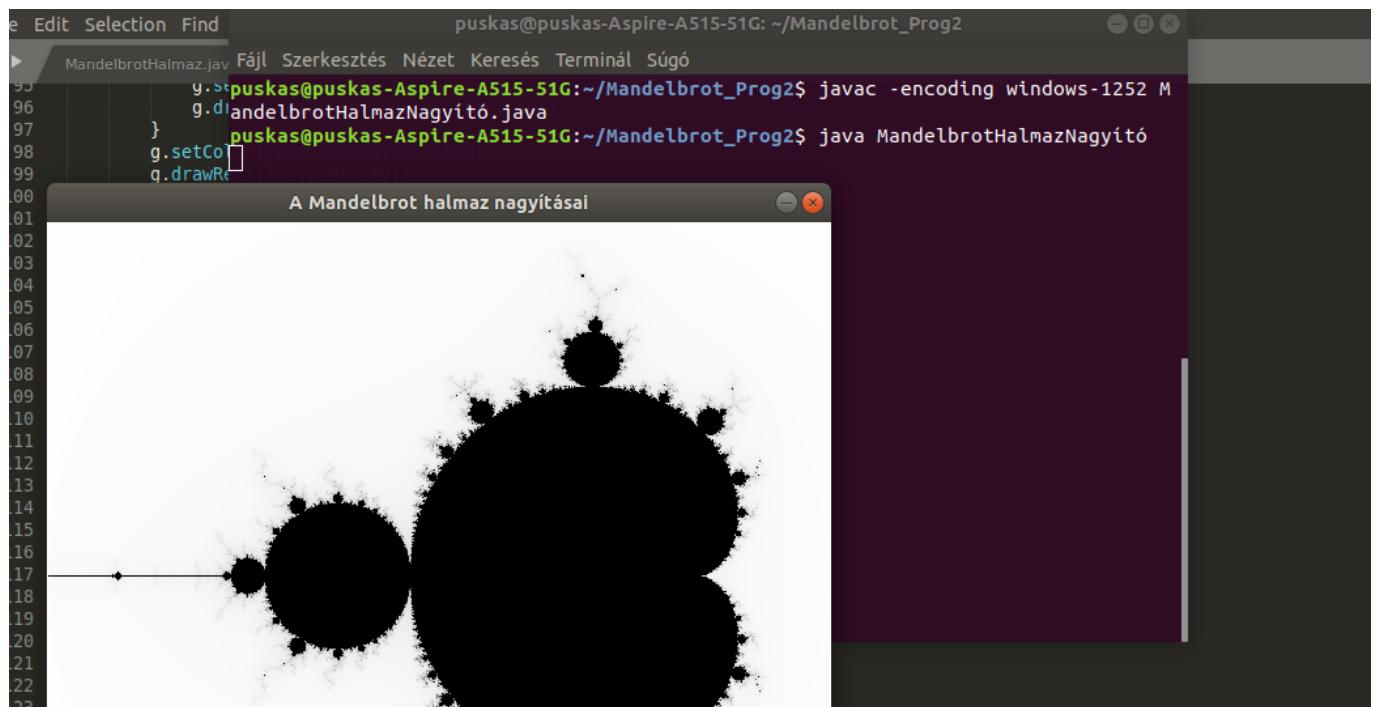
```
C:\...> javac MandelbrotHalmazNagyító.java
C:\...> java MandelbrotHalmazNagyító
```

Láthatjuk, hogy a fordításhoz és futtatáshoz szükséges parancs egy windows operációs rendszerrel rendelkező gépen lett beütve, mivel ilyen fájlrendszer windows-ra utal.

Rövid keresés után megtaláltam a windows által használt betűkészleteket, és azt találtam, hogy a windows-1252 a Latin 1-es betűkészlettel egyezik meg, amely tartalmazza az általunk használt ékezetes betűket. A megoldás tehát egyszerű, a javac után egy kapcsolóval megadjuk a kódolás típusát a következőképp:

```
javac -encoding windows-1252 MandelbrotHalmazNagyító.java
```

A futásról készült kép:



```
puskas@puskas-Aspire-A515-51G:~/Mandelbrot_Prog2$ javac -encoding windows-1252 M
puskas@puskas-Aspire-A515-51G:~/Mandelbrot_Prog2$ java MandelbrotHalmazNagyító
A Mandelbrot halmaz nagyításai
```

12.2. I334d1c4^5

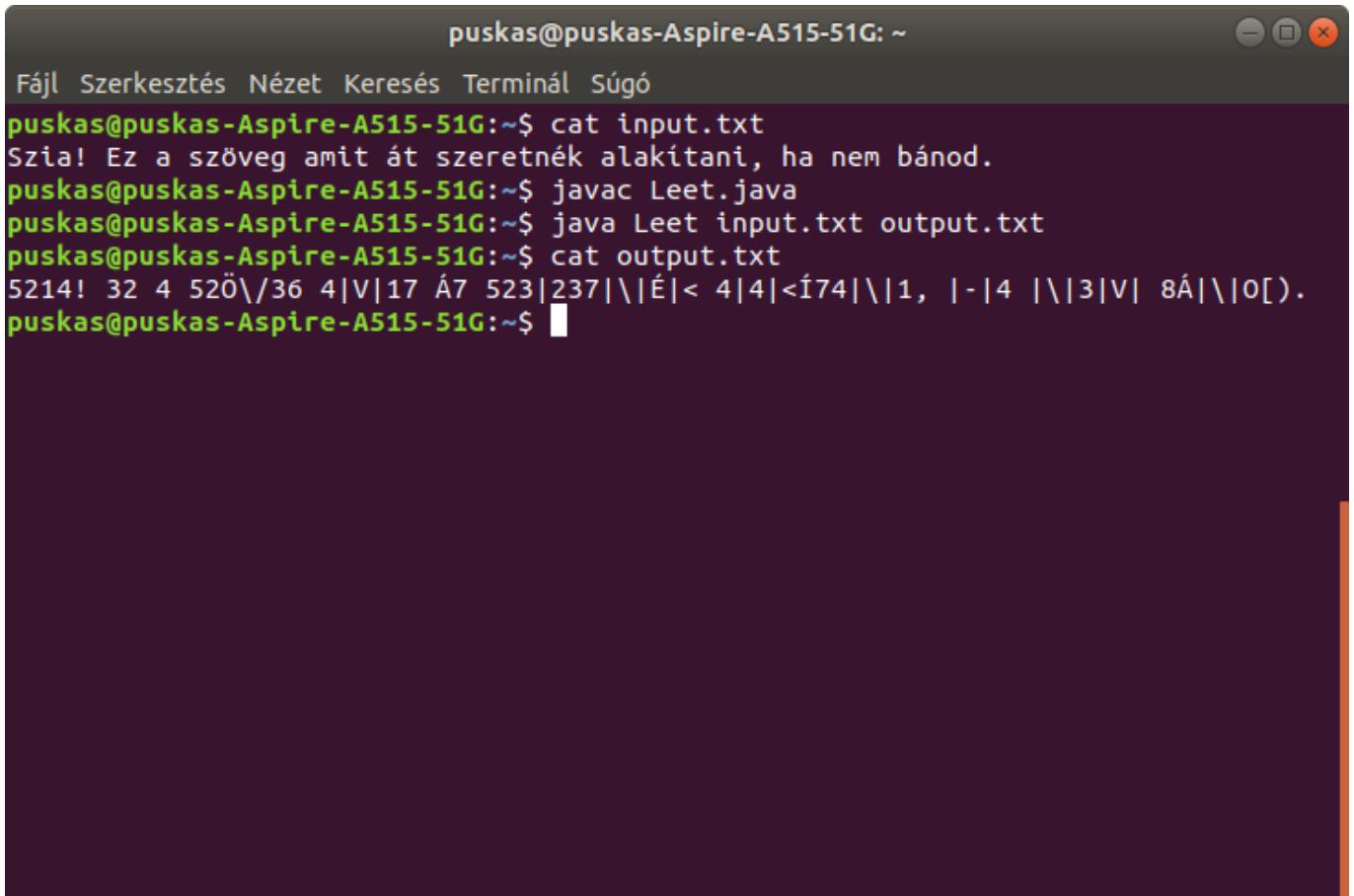
Megoldás forrása:

A Leet olyan eljárás, amikor az írott angol szöveg karaktereit más ASCII készletű karakterekkel helyettesítjük. A leetspeak eredetileg az illegális tevékenykedők nyelveként funkcionált.

A leetspeak használatával az adott szöveget a keresők nem találták meg, és a megszokottól eltérő karakterhasználat miatt az illetéktelenek nem is tudták értelmezni a szöveget.

Korábban Prog1 során már megjelent a leet, viszont akkor egy lexer fájlként kellett a forráskódot értelmezünk. A mostani feladatban nem .l kiterjesztésű állományt kellett felhasználnunk, hanem egy konvertáló programot kellett írnunk C++ vagy Java nyelven, és a program egy osztályként kellett hogy tartalmazza a leet-et.

Nézzük is a futásról készült képet:



The screenshot shows a terminal window titled "puskas@puskas-Aspire-A515-51G: ~". The terminal displays the following command-line session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~$ cat input.txt
Szia! Ez a szöveg amit át szeretnék alakítani, ha nem bánod.
puskas@puskas-Aspire-A515-51G:~$ javac Leet.java
puskas@puskas-Aspire-A515-51G:~$ java Leet input.txt output.txt
puskas@puskas-Aspire-A515-51G:~$ cat output.txt
5214! 32 4 52Ö\|36 4|V|17 Á7 523|237|\|É|< 4|4|<Í74|\|1, |-|4 |\|3|V| 8Á|\|0[].
puskas@puskas-Aspire-A515-51G:~$
```

Maga a forráskód pedig:

```
public class Leet {
    public static void main(String[] args) throws Exception{
        if(args.length != 2){
            System.out.println("usage: Leet input_file_name ←
                               output_file_name");
            System.exit(-1);
    }
}
```

A Leet osztály létrehozása után a main-ben a usage függvény tartalma, jelen esetben a használata jelenik meg a felhasználó előtt abban az esetben, ha rosszul futtatná a programot.

```
java.io.FileReader file = new java.io.FileReader(args[0]);
    java.io.FileWriter fw = new java.io.FileWriter(args[1]);
    LeetCipher lc = new LeetCipher();
```

Létrehozunk egy file változót, amibe az argumentumként adott 0. elem (input)-ból olvasunk, majd egy fw változót, ami pedig az első elembe (kimenet) fog írni. Ezután egy LeetCipher osztályú lc változót fogunk inicializálni. Az osztály definiálása lentebb látható.

```
int k = 0;
    while ((k=file.read()) != -1) {
        fw.write(lc.chiper((int)Character.toUpperCase((char)k)));
    }
    file.close();
```

```
        fw.close();
    }
}
```

Létrehozunk egy k változót, ez elején int típusú, majd egy while függvény következik, ez nagyjából úgy néz ki, hogy amíg a read függvény képes olvasni a bemenetről, addig a kimenetre írunk, méghozzá a következőképpen: az lc-re meghívjük az osztály chiper változóját, ennek paraméterül a k karakter nagybetűs változatát adjuk, melyet int típusúvá kényszerítünk, a toUpperCase függvénynek paraméterül adott k értéket pedig char típusúvá kényszerítjük. Amikor már nem tud olvasni, az inputot és outputot is lezárjuk.

```
class LeetCipher {
    private String[] leetchars = new String[]{
        "4", "8", "<", "()", "3", "|=", "6", "|-", "1", "_|", "|<", "|", "V ←",
        "|", "|\\|", "O",
        "|>", "0.", "|2", "5", "7", "|_|", "\\\\", "\\\X/", "}" {", "\\", "2"
    };
}
```

Jöhet a LeetCipher osztály megadása. A leetchars string típusú tömb a helyettesítő karaktereket fog tartalmazni, melyek az angol abc betűinek felelhetők meg.

```
private String[] leetnums = new String[]{
    "O", "I", "Z", "E", "A", "S", "G", "T", "B", "g"
};
```

A leetnums tömb pedig rendre a 0-9 számjegyek helyettesítőértékei lesznek.

```
public String chiper(int ch) {
    if (ch >= 65 && ch <= 90) {
        return leetchars[ch - 65];
    }
    else if (ch >= 48 && ch <= 57) {
        return leetnums[ch - 48];
    }
    else {
        return String.valueOf((char) ch);
    }
}
```

A chiper függvény arra fog rávizsgálni, hogy a beolvasott karakter számjegy, betű, vagy egyéb karakter e. Ha a ch értéke 65 és 90 között van, akkor betű lesz az értéke, mivel az angol abc nagybetűinek decimális értéke 65-től kezdődik (A), és 90-nél végződik (Z). Ebben az esetben a leetchars függvényből fogja a megfelelő stringet visszaadni. Ezt úgy kell elképzelni, hogy vegyük pl. az "M" betűt. Az M betű értéke 77 lesz. Ha a 77-ből kivonunk 65-öt, akkor 12-t kapunk, tehát a leetchars tömb 12. elemét fogja a függvény visszaadni, ami a "V". Ha 48 és 57 között van a beolvasott karakter értéke, akkor a beolvasott karakter számjegy, mivel a 0 értéke 48, a 9 értéke pedig 57. Ekkor a leetnums tömb elemei közül kapjuk vissza a megfelelő elemet, működése megegyezik a karakterek beolvasásával, viszont itt a 65 helyett 48-at vonunk ki, mivel a számjegyek decimális értéke 48-től kezdődik.

Amennyiben a beolvasott karakter sem nem szám, sem nem betű, akkor a visszaadjuk az eredeti értékét, mely jelen esetben úgy történik, hogy a decimális alakját átkonvertálja karakterré.

12.3. Full screen

Feladatunk az volt, hogy írunk javaban egy teljes képernyős programot. Ehhez tippként Tanár Úr által linkelt labirintus játékot kaptuk, ezt fogjuk átnézni.

Letöltéskor ugyanabba a problémába ütközünk, mint az Encoding feladatnál, így a fájlneveket itt is át kell írnunk, hogy olvashatóak legyenek.

A programot sikeresen felélesztettem, viszont ez csak egy terminálba kirajzolós 2D-s program:

The terminal window title is "puskas@puskas-Aspire-A515-51G: ~/proba". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
Játékidő: 10 lépés
Hányszor ettek meg: 0
puskas@puskas-Aspire-A515-51G:~/proba$ java javattanitok.LabirintusVilág labirin
tus.txt
| | | FAL | | FAL | | FAL | FAL | FAL
| | | | | | | | |
| FAL | | FAL | | FAL | | FAL | |
| | | | FAL | | FAL | | |
| FAL | FAL | | | FAL | FAL | | FAL
| | | | FAL | | | | |
| FAL | | | FAL | | FAL | FAL |
| | | | FAL | | FAL | | |
| FAL | | | | | | | FAL
| | | | FAL | | | | FAL | FAL
A labirintus a(z) 1. lépéssben:
| | | FAL | | FAL | | FAL | FAL | FAL
| | | | | | | | |
| FAL | | FAL | | FAL | | FAL | |
| | K | | FAL | | FAL | S | | K
| FAL | FAL | | | K | FAL | FAL | | FAL
| | | | FAL | | | | |
| FAL | | | | FAL | | FAL | FAL |
| | | | FAL | SK | FAL | | FAL | |
| | FAL | | | | | | | FAL
```

Következő lépés tehát a full screen-es változat lesz. Ehhez a LabirintusVilágOsztály.java állományra lesz szükségünk.

Az állományt sikeresen fordítottam illetve futtattam, viszont fehér képernyőt kaptam, amit sajnos egy rendszer újraindítás követett, ugyanis semmit nem érzékelte a gép, de legalább az előző feladatot el sem mentettem így újra kellett kezdenem (:.

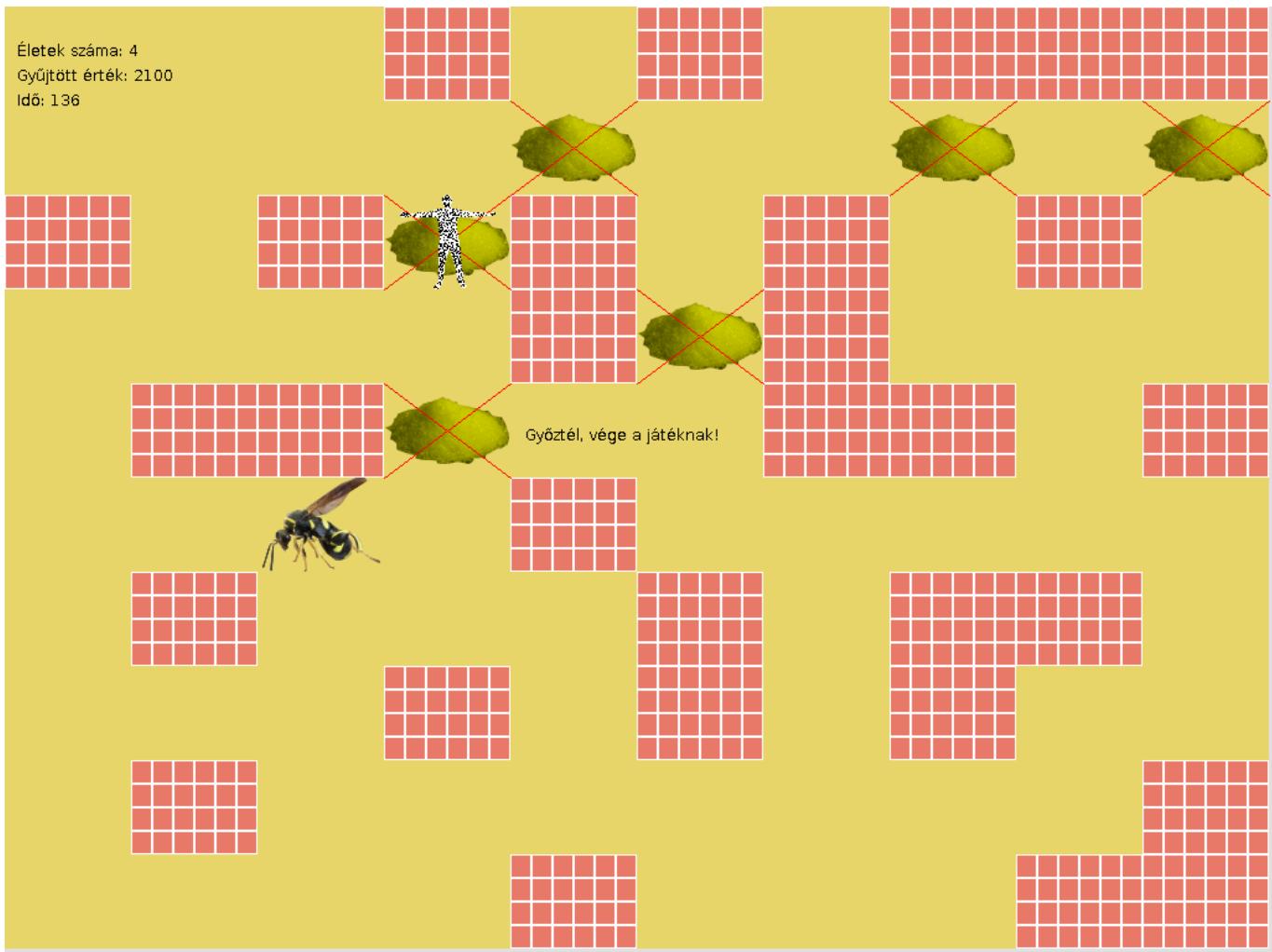
```
public void képErőforrásokBetöltése(java.awt.GraphicsConfiguration
    graphicsConfiguration) {

    falKép = kompatibilisKép("fal.png", graphicsConfiguration);
    járatKép = kompatibilisKép("járat.png", graphicsConfiguration);
    hősKép = kompatibilisKép("hős.png", graphicsConfiguration);
    szörnyKép = kompatibilisKép("szörny.png", graphicsConfiguration);
    kincsKép = kompatibilisKép("kincs.png", graphicsConfiguration);
```

}

Ez a pár sor döbbentett rá hogy kéne pár képfájl..

Miután beszereztem a szükséges képfájlokat, a program probléma nélkül futott.



A forráskód számunkra a teljes képernyős mód miatt fontos. A full screen-ért a teljes Képernyős Mód függvény a felelős. De mindenek előtt nézzük meg, hogy történik a monitorunk beállítása:

```
/* Teljes képernyős módba próbálunk váltani. */
// A lokális grafikus környezet elkérése
java.awt.GraphicsEnvironment graphicsEnvironment
    = java.awt.GraphicsEnvironment.getLocalGraphicsEnvironment ←
        ();
// A grafikus környzetből a képernyővel dolgozunk
graphicsDevice = graphicsEnvironment.getDefaultScreenDevice();
```

A `java.awt.GraphicsEnvironment` a `GraphicsDevice` objektumok, vagyis a rendelkezésre álló grafikus eszközök gyűjteménye lesz. Ezt a `getLocalGraphicsEnvironment` függvény meghívásával érhetjük el, és a `graphicsEnvironment` nevű változóban fogjuk tárolni. A `graphicsDevice` változót beállítjuk az alapértelmezett `ScreenDevice`-ra, vagyis a monitorunkra.

```
boolean fullScreenTámogatott = graphicsDevice.isFullScreenSupported();
```

Egy boolean változóba lekérdezzük, hogy támogatja e a kijelzőnk a full screen-t. A graphichDevice a forráskód elején kerül beállításra:

```
// A fullscreenbe kapcsoláshoz
java.awt.GraphicsDevice graphicsDevice;
// A megjelenítéshez
java.awt.image.BufferStrategy bufferStrategy;
```

Eztán a fullScreenTamogatott függvényt kell megvizsgálnunk.

```
if(fullScreenTamogatott) {
    graphicsDevice.setFullScreenWindow(this);
    // az aktuális képernyő jellemzők (szélesség, magasság, ←
    // színmélység,
    // frissítési frekvencia) becsomagolt elkérése
    java.awt.DisplayMode displayMode
        = graphicsDevice.getDisplayMode();
    // és kiíratása
    szélesség = displayMode.getWidth();
    magasság = displayMode.getHeight();
int színMélység = displayMode.getBitDepth();
    int frissítésiFrekvencia = displayMode.getRefreshRate();
    System.out.println(szélesség
        + "x" + magasság
        + ", " + színMélység
        + ", " + frissítésiFrekvencia);
```

A függvényben megvizsgáljuk, hogy a grafikus megjelenítőnk támogatja e a teljesképernyős módot. Ha igen, akkor a setFullScreenWindows a megadott képernyőre, jelen esetben a graphicsDevice-ra hívódik meg. A java.awt.DisplayMode az adott grafikus eszköz megjelenítési módjait kéri le, szélességet, magasságot és színmélységet illetve frissítési frekvenciát.

```
java.awt.DisplayMode[] displayModes
    = graphicsDevice.getDisplayModes();
    // Megnézzük, hogy támogatja-e az 1024x768-at, mert a
    // példa játékunkhoz ehhez a felbontáshoz készítettük a képeket
boolean dm1024x768 = false;
for(int i=0; i<displayModes.length; ++i) {
    if(displayModes[i].getWidth() == 1024
        && displayModes[i].getHeight() == 768
        && displayModes[i].getBitDepth() == színMélység
        && displayModes[i].getRefreshRate()
        == frissítésiFrekvencia) {
        graphicsDevice.setDisplayMode(displayModes[i]);
        dm1024x768 = true;
        break;
    }

}
if(!dm1024x768)
    System.out.println("Nem megy az 1024x768, de a példa ←
        képméretei ehhez a felbontáshoz vannak állítva.");
```

```
    } else {
        setSize(szélesség, magasság);
        validate();
        setVisible(true);
    }

    createBufferStrategy(2);

    bufferStrategy = getBufferStrategy();

}
```

A displayModes tömbbe beletöljük a lehetséges megjelenítési felbontásokat, majd megnézzük, hogy az 1024x768-as felbontás támogatott e. Ezt úgy tesszük, hogy egy for ciklust indítunk, mely végigzongorázik a displayModes elemein, és megvizsgálja, hogy a szélesség és magasság és színmélység és képfrissítési ráta megegyezik e az adott értékekkel, ha igen, beállítjuk a felbontást 1024x768-ra, ha nem, kiíratjuk, hogy nem sikerült 1024x768-as felbontással a megjelenítés.

13. fejezet

Helló, Stroustrup!

13.1. JDK Osztályok

Feladat forrása:

Feladatunk az volt, hogy írunk olyan boost C++ programot, amely kilistázza, hogy hány darab .java kiterjesztésű állomány (java osztály) van a jdk forrásában található src.zip állományban.

Ehhez a korábbi prog1-es kurzus fénkard nevű forráskódjából kellett felhasználnunk egy függvényt.

```
void read_file ( boost::filesystem::path path, int &szamlalo )
{
    if ( is_regular_file ( path ) ) {
        string ext ( ".java" );
        if ( !ext.compare ( boost::filesystem::extension ( path ) ) ) {
            string actjavaspath = path.string();
            size_t end = actjavaspath.find_last_of ( "/" );
            string act = actjavaspath.substr ( 0, end );
            szamlalo++;
        }
    } else if ( is_directory ( path ) )
        for ( boost::filesystem::directory_entry & entry : boost::filesystem::directory_iterator ( path ) )
            read_file ( entry.path(), szamlalo );
}
```

A függvény fentebb látható. Lényegében megegyezik a fénkard.cpp read_acts függvényével, viszont az ext stringet át kell írnunk, hogy java végződésű állományokra alkalmazzuk a keresést. Ez a függvény benéz

a mappákba, és a java állományokat kilistázza, valamint a szamlalo változónk miatt meg is számolja azokat. Jöhet is a main függvény:

```
int main( int argc, char *argv[] )
{
    char* filename=argv[1];
    string str = "mkdir ";
    string str2 = "cd ";
    string str3 = "unzip /usr/lib/jvm/openjdk*/src.zip -d ";

    str = str + filename;
    str2 = str2 + filename;

    str3 = str3 + filename+("/");
}
```

Egy char változóba eltároljuk a parancssori argumentumként kapott első elemet, ez egy mappanév lesz, ugyanis a program futtatásakor meg kell adnunk egy mappanevet, ahová ki szeretnénk csomagolni az src.zip-et.

Miután stringekbe írtuk a szükséges rendszerutasításokat, a stringek végéhez hozzácsatoljuk a filename változót, amely a következőket eredményezni:

- Az str string miatt létrehoz a megadott néven egy mappát
- Az str2 string miatt belelép ebbe a mappába
- Az str3 string miatt pedig kicsmagolja ide az src.zip állományt

```
const char* command=str.c_str();

const char* second_command=str2.c_str();

const char* third_command=str3.c_str();

system(command);
system(second_command);
system(third_command);
```

Ezután a stringeket átkonvertáljuk char*-á. Erre azért van szükség, ugyanis egy szimpla string nem fogja végrehajtani a fentebb lévő utasításokat, így ezeket a parancsokat át kell adnunk a system függvénynek, viszont stringet nem tudunk paraméterül megadni. Mostmár tehát a system függvények használatával létrehoztunk egy mappát, és kicsomagoltuk ebbe a mappába a src.zip állományt.

```
string path=argv[1];
boost::filesystem::path a(argv[1]);
int szamlalo=0;
read_file(a, szamlalo);
cout<< "Az src.zip-ben található java osztályok száma: "<< szamlalo << endl;
return 0;
```

```
}
```

Ezután ráengedjük a mappára a `read_file` függvényünket, amely megszámolja a `.java` kiterjesztésű fájlokat. A forráskódban különös szerepe van az alábbi header-nek:

```
#include "../boost_1_71_0/boost/filesystem.hpp"
```

Ezen nyilván gépenként változtatni kell, mivel nem mindenkinél ugyanott található a boost mappa, és azon belül is a `filesystem.hpp` állomány, így meg kell adnunk a gépünkön a boost mappa elérési tartományát. Amennyiben nincs letöltve a boost, a következő linken tudjuk beszerezni: <https://www.boost.org/users/download/>

A programot a következőképpen tudjuk fordítani:

```
g++ jdk_classes.cpp -o jdk_classes -lboost_system -lboost_filesystem -lboost_program_options -std=c++14
```

A futtatás pedig így néz ki:

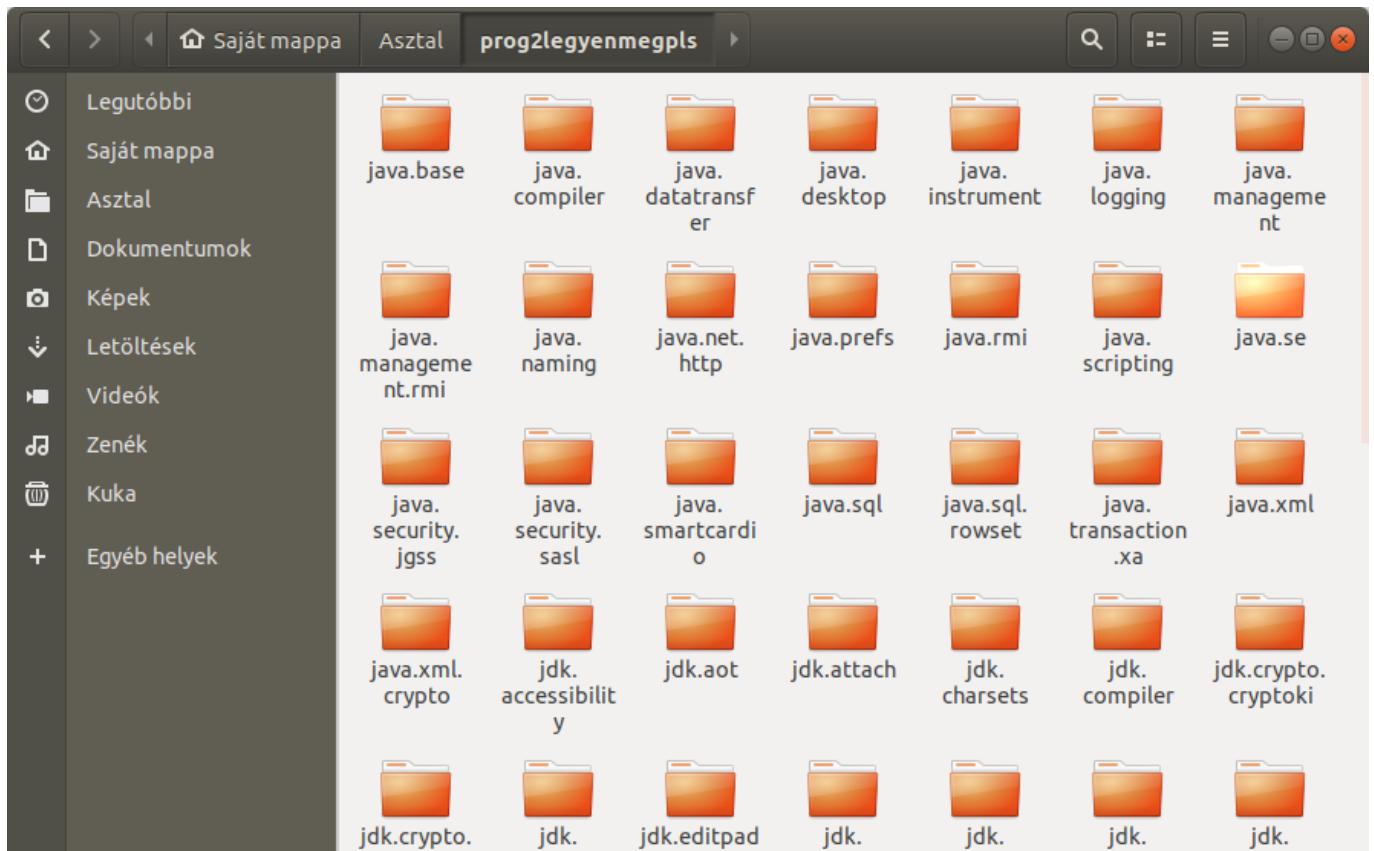
```
./jdk_classes mappanev
```

A program futásáról itt egy kép:

The screenshot shows a terminal window titled "puskas@puskas-Aspire-A515-51G: ~/Asztal". The window contains the following text output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
inflating: prog2legyenmegpls/jdk.xml.dom/org/w3c/dom/css/CSSStyleRule.java
inflating: prog2legyenmegpls/jdk.xml.dom/org/w3c/dom/css/CSSCharsetRule.java
inflating: prog2legyenmegpls/jdk.xml.dom/org/w3c/dom/css/ElementCSSInlineStyle.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipPath.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipConstants.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipFileSystemProvider.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipCoder.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipFileAttributes.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ByteArrayChannel.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipInfo.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipUtils.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipFileSystem.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipDirectoryStream.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/JarFileSystem.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipFileStore.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/JarFileSystemProvider.java
inflating: prog2legyenmegpls/jdk.zipfs/jdk/nio/zipfs/ZipFileAttributeView.java
inflating: prog2legyenmegpls/jdk.zipfs/module-info.java
Az src.zip-ben található java osztályok száma: 18294
```

A következő képen pedig az látható, hogy létrehozta a parancssori argumentumként megadott néven a mappát, és kicsomagolta az `src.zip` állományt az adott mappába.



13.2. Másoló-mozgató szemantika és Összefoglaló egybevonva

Megoldás forrása:

A másoló szemantikára akkor van szükségünk, mikor egy meglévő objektumból szeretnénk mégegy példányt megvalósítani. A másoló szemantikához másoló konstruktort kell írnunk. A másolásra két módszer van. Van az ún. shallow copy, vagy magyarul sekély másolás, amely annyit tesz, hogy lényegében nem történik másolás, a másolt objektum megkapja a másolandó objektum memóriacímét. Tehát az új objektumunk egy referencia lesz az eredeti objektumra, amely problémákhöz vezethet, mivel amennyiben az eredeti objektumot változtatjuk, mivel a másolt objektum az eredeti memóriacímére mutat, ezért az is változni fog. Példa erre a Prog1-es védésem, mikor másoló konstruktort kellett írni a binfához, az én megoldásom viszont sekély másolást végzett, tehát a másolás sikeresen megtörtént, viszont amennyiben az eredeti objektumot kitöröltük, a program futtatásakor hibába ütközünk. Ez annak köszönhető, hogy a másolt objektum az eredeti memóriacímére mutatott, viszont annak megszűnése után a másolt objektum nem tudott tovább élni, mivel megszűnt a cím amire mutatott. Létezik viszont egy másik megoldás, a mély másolás. Ebben az esetben konkrét másolás történik. Ilyenkor a háttérben új memóriaterület foglalás zajlik, így értelemszerűen a shallow copy-val szemben ez a megoldás több időt vesz igénybe. Ez a megoldás akkor szükséges, ha az eredeti objektumon még változtatni szeretnénk, mivel itt ténylegesen új objektum jön létre, ezért ha a másolt objektumon módosítást végezzük, az eredeti nem változik és ez fordítva is igaz. Összetett objektumok esetében rekurzív másolást kell végeznünk, ami azt jelenti, hogy az összes adattag tulajdonságait másolnunk kell. A mozgató szemantika végkimenetében megegyezik a másoló szemantikával, ugyanis itt is az új objektum meg fog egyezni a régi objektummal. Itt mozgató konstruktor megírására van szükségünk. A mozgató szemantika lényege, hogy az eredeti objektumot „belemozgatjuk” az új objektumba,

ezáltal az eredeti objektum kinullázódik, és az új objektumba kerülnek az adatai, tulajdonságai, ekkor a régi objektum már nem veszi el feleslegesen a területet a memóriában, így ha tudjuk, hogy az eredeti objektum adatait nem kell megőrizni, ez a megoldás a legpraktikusabb, mivel több szabad terület marad a memóriában. A mozgatáshoz alkalmaznunk kell az objektum adataira az std::move függvényt, ami a mozgatásért felelős. Fontos dolog, hogy lvalue és rvalue értékátadásról is beszélhetünk, a kettő között pedig lényeges különbség van. Az „lvalue” értékek olyan objektumot jelölnek, melynek a memóriában lefoglalt helye van. Erre példa az int típusú változók: int helo=0. Azonban ha ezzel valamiféle műveletet végzünk, például hozzáadunk ötöt, rvalue értéket kapunk, mivel a „helo+5”-nek nincs előre lefoglalt terület a memóriában. Mozgatáskor „rvalue” referenciát használunk, mely a C++11 újítása, és “**&&**” operátorral használjuk, az „lvalue” referenciára egyébként a “**&**” operátort használjuk.

Feladatunk az volt, hogy mutassuk be a C++11 másoló és mozgató szemantikáját.

Kezdjük is a másolával.

Ehhez a feladathoz a bevprog óta jól ismert védési programot, a z3a7.cpp-t fogjuk használni.

Mint az korábban már védési feladat is volt, ehhez készítünk másoló konstruktort. Az alábbi sorral kell kiegészítenünk az alap konstruktort:

```
LZWBInFa(const LZWBInFa & regi) {
    std::cout << "Másoló konstruktor" << std::endl;
    gyoker = masol(regi.gyoker);
}
```

Ez a rész az LZWBInFa típusok implementálásakor meg fog hívódni, mivel ez a függvény is a konstruktor része. Ez annyit tesz, hogyha új LZWBInFa-t hozunk létre, és paraméterül kap egy másik binfát, akkor az új fa gyökerére hívja meg a masol függvényt, és a régi gyökeret adja neki paraméterül.

```
Csomopont *masol(Csomopont * elem) {
    Csomopont* ujelem = NULL;
    if (elem != NULL) {

        ujelem=new Csomopont (elem->getBetu());
        ujelem->ujEgyesGyermek(masol(elem->egyesGyermek()));
        ujelem->ujNullasGyermek(masol(elem->>nullasGyermek()));
    }
    return ujelem;
}
```

A masol függvény a másoló konstruktor meghívásakor fog meghívódni. Ez a függvény deklarál egy új csomópontot, amelynek NULL kezdőértéket ad. Ezután, amennyiben a régi Csomopontunk nem NULL értékkel bír, rekurzívan átmásoljuk a régi Csomopont egyes és nullas gyermekeit, majd ez az ujelem lesz visszaadva.

Az LZWBInFa konstruktorát ki kell egészítenünk még az “**=**” operátor túlterhelésével, ami az alábbi módon történik:

```
LZWBInFa & operator=(LZWBInFa & regi) {           //egyenlőség jel túlterhelés
    this->gyoker = regi.gyoker;
    this->fa = regi.fa;
    return *this;
}
```

Ez annyit fog tenni, hogy az "=" operátor használatakor további funkciók kerüljenek végrehajtásra. Jelen esetben az fog történni, hogy az egyenlőségjel bal oldalán található -jelen esetben- LZWBinFa gyökerét, és magát a fát egyenlővé teszi az egyenlőségjel jobb oldalán található eredeti LZWBinFa gyökerével és fájával.

```
LZWBinFa *binFa2= new LZWBinFa(*binFa);
```

Itt hívódik meg a másoló konstruktur.

Láthatjuk, hogy sikerült a másolás:

```
puskas@puskas-Aspire-A515-51G: ~
Fájl Szerkesztés Nézet Keresés Terminál Súgó
-----0(2)
-----0(3)
-----0(4)
---/(0)
----1(3)
-----1(5)
----0(4)
----1(2)
----0(1)
----0(2)
-----0(3)
depth = 5
mean = 3.6
var = 0.894427
Másoló konstruktor
0x559b6c855340
Másolt fa:
-----1(2)
-----0(3)
----1(1)
-----1(3)
----0(2)
-----0(3)
-----0(4)
```

Térjünk át a mozgató konstruktorra.

```
LZWBinFa (LZWBinFa && regi) {
    std::cout << "LZWBinFa move ctor" << std::endl;
    gyoker = nullptr;
    *this = std::move(regi);
}
```

A mozgató konstruktorunk fentebb látható. Itt az eredeti gyökért nullává tesszük, és az std::move bal oldalán lévő elemet egyenlővé tesszük a régi objektummal.

```
LZWBinFa& operator=(LZWBinFa && regi) {
    std::cout << "LZWBinFa move assign" << std::endl;
    std::swap(gyoker, regi.gyoker);
    std::swap(fa, regi.fa);
    return *this;
```

```
}
```

Az egyenlőséggel operátort ebben az esetben is túlterheljük. Itt viszont másképp járunk el. Az új objektumunk gyökerét kicseréljük a régi objektumunk gyökerével, majd visszaadjuk ezt az értéket.

Tehát amint létrehozzuk a binFa3 objektumunkat, a fentebb látható két függvény egyből meghívódik, mint azt a nyomonkövetés miatt láthatjuk is.

```
LZWBinFa binFa3=std::move(binFa);
```

A fentebb látható módon hozzuk létre a mozgatott fánkat. Ez a sor lesz az, ami a fentebb kiemelt két függvényt megindítja. Az eső függvényünk tehát kinullázza és "belemozgatja" a régi fát az újannon létrehozott binFa3-ba, a mozgató értékadás pedig felcseréli a két fa adatait, ezzel az eredeti fa lesz kinullázva, amint azt láthatjuk is lentebb.

The screenshot shows a terminal window titled 'puskas@puskas-Aspire-A515-51G: ~'. The window contains the following text output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
-----0(4)
-----1(2)
----0(1)
----0(2)
-----0(3)

depth = 5
mean = 3.6
var = 0.894427
LZWBinFa move ctor
LZWBinFa move assign
Az eredeti fa mozgatás után:

depth = -1
mean = -nan
var = 0
A mozgatással keletkezett fa:
-----1(2)
-----0(3)
----1(1)
----1(3)
----0(2)
-----0(3)
-----0(4)
```

14. fejezet

Helló, Gödel!

14.1. Gengszterek

A robotautó bajnokság célja egy egységes platform biztosítása kutatási forgalomirányítás vizsgálatához.

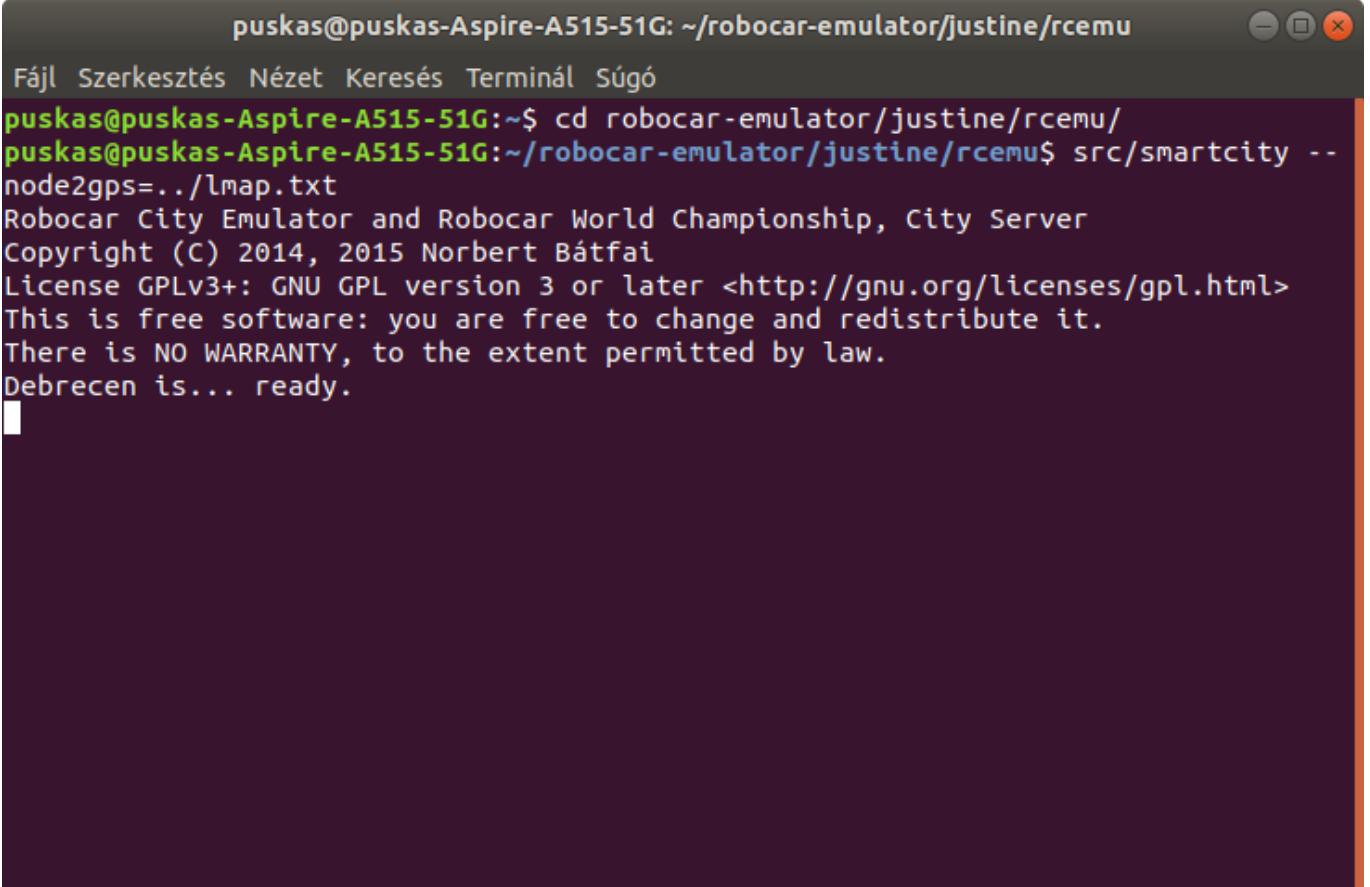
A program lényege, hogy gengszterek, és őket kergető rendőrök taláhatóak Debrecen térképén, folyamatos mozgásban, míg a rendőrök el nem kapják a gengsztereket.

A projekt felélesztése az alábbi linken található meg: <https://github.com/Flibielt/robocar-emul-fbclid=IwAR1CH-VLCKLwktFpy0G-T0fQZcLh-AgalyySPbbsbydPLBn0q3am40U5lMo#hiba-jC3%ADt%C3%A1sok-%C3%A9s-haszn%C3%A1lat>

A szükséges csomagok telepítése után a program futtatás a következőképp (nézne) ki.

Először Debrecen városát kell elindítanunk az RCEMU mappából.

```
src/smartcity --node2gps=.. /lmap.txt
```

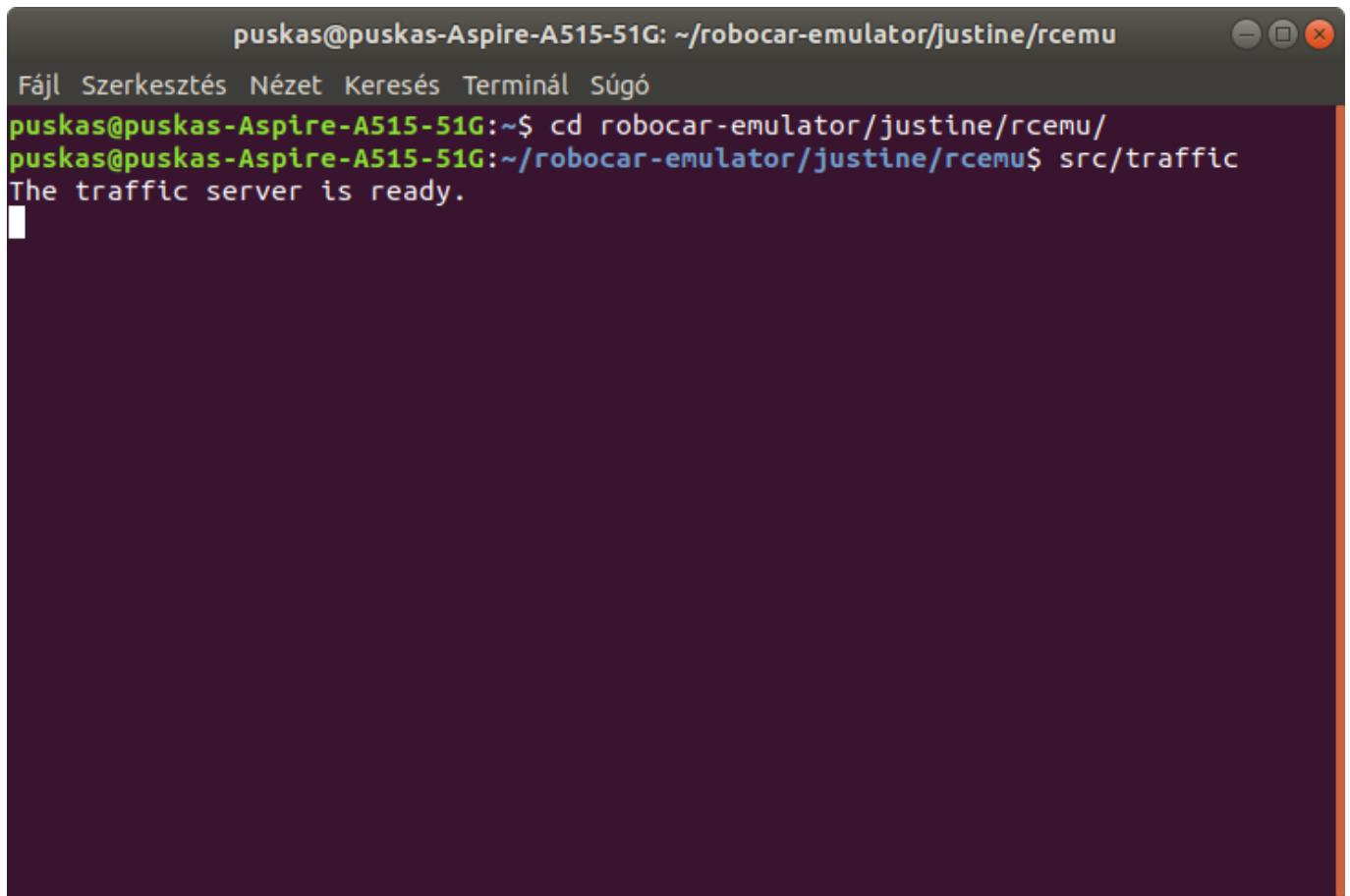


A screenshot of a terminal window titled "puskas@puskas-Aspire-A515-51G: ~/robocar-emulator/justine/rcemu". The window shows the following command-line session:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~$ cd robocar-emulator/justine/rcemu/
puskas@puskas-Aspire-A515-51G:~/robocar-emulator/justine/rcemu$ src/smartercity --node2gps=../lmap.txt
Robocar City Emulator and Robocar World Championship, City Server
Copyright (C) 2014, 2015 Norbert Bátfai
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Debrecen is... ready.
```

Ezután egy új ablakot kell megnyitnunk, és ugyancsak az RCEMU mappában maradva a közlekedést is elindítjuk az alábbi parancs beütölgésével.

```
src/traffic
```



```
puskas@puskas-Aspire-A515-51G: ~/robocar-emulator/justine/rcemu
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~$ cd robocar-emulator/justine/rcemu/
puskas@puskas-Aspire-A515-51G:~/robocar-emulator/justine/rcemu$ src/traffic
The traffic server is ready.
```

És itt kezdődik az utolsó szakasz, ekkor már az RCWIN mappában dolgozunk. Következő lépésként egy .jar fált kéne felépítenünk az alábbi parancs kiadásával:

```
mvn clean compile package site assembly:assembly
```

Már csak egy lépés van hátra a program futtatásához, méghozzá az, hogy futtassuk a programot az alábbi parancssal:

```
java -jar target/site/justine-rcwin-0.0.16-jar-with-dependencies.jar ../ ←
lmap.txt
```

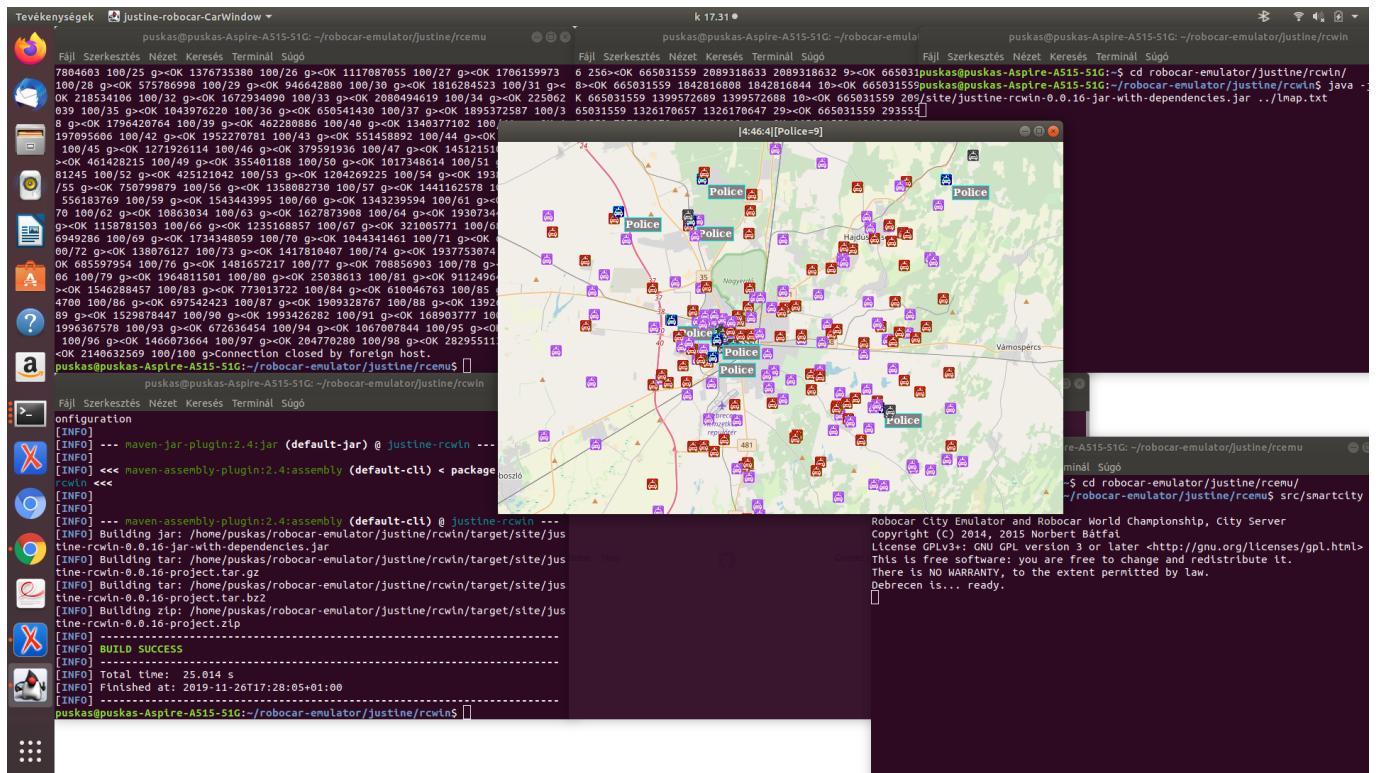
Ezzel kirajzolódik számunkra a térkép, viszont így még nem teljes a szimuláció, el kell ugyanis a térképen helyeznünk a gengsztereket és az őket üldöző rendőröket. Ezt a lentebb látható két parancs bepötyögésével tudjuk elérni, az rcemu könyvtárban dolgozva:

```
src/samplemyshmclient --team=Police
```

Elhelyeztük a rendőröket.

```
(sleep 1; echo "<init Norbi 100 g>"; sleep 1)|telnet localhost 10007
```

Elhelyeztük a rendőröket, és gyakorlatilag készen is vagyunk.



A futásról a fentebb látható képet készítettem, sajnos az összes terminál ablak nem fért fel láthatóan, viszont fentebb az összes parancs le van írva.

A feladat viszont az volt, hogy mutassunk rá a gengszterek rendezésére a programban.

A lambda rendezés az alábbi pár sorban történik.

```
std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( Gangster x, 
    Gangster y )
{
    return dst ( cop, x.to ) < dst ( cop, y.to );
};
```

Itt annyi történik, hogy a sort függvényben a gengszterek vektorát rendezzük. Átadjuk neki a vektor elejét és végét, illetve összehasonlítjuk a gengszterek rendőrökkel való távolságát, és mivel rendezés függvényben vagyunk, az a genszter lesz az első eleme a vektornak, amelyik a legközelebb áll egy rendőrhöz.

14.2. STL Map érték szerinti rendezése

A C++-ban a map típusok tárolóként vannak jelen. A map típus egy növekvően rendezett szociatív tömb. Ebben a tömbben std::pair <> segítségével kulcs-adat párokat tárolunk. A kulcs és adat is paraméter lesz. A kulcs értéket rendezéshez és megkülönböztetéshez használjuk. A map érték (adat) egy ehhez társított tartalom. A kulcs és adat típusa lehet eltérő típusú is. Nézzük is a programot.

```
typedef std::pair<std::string, int> pair;
```

Létrehozunk egy mapot, amit pair néven implementálunk, és egy string illetve int típusú változót fog tárolni.

```
std::map<std::string, int> map= {
    {"Travis Scott", 999}, {"Kendrick Lamar", 522}, {"Kanye West", ←
        625}, {"Video Info 1", 9999}
};
```

Ezután létrehozunk egy mapot, és feltöljük 5 kulcs-adat párral, ami a fent megadott formalitáshoz hasonlóan string-int párok lesznek.

```
std::vector<pair> vec;

std::copy(map.begin(), map.end(), std::back_inserter(vec));
```

Ezután létrehozunk egy üres vektort, és a fentebb létrehozott map értékpárjait belemásoljuk ebbe a vektorba. A továbbiakban erre a vektorra lesz szükségünk.

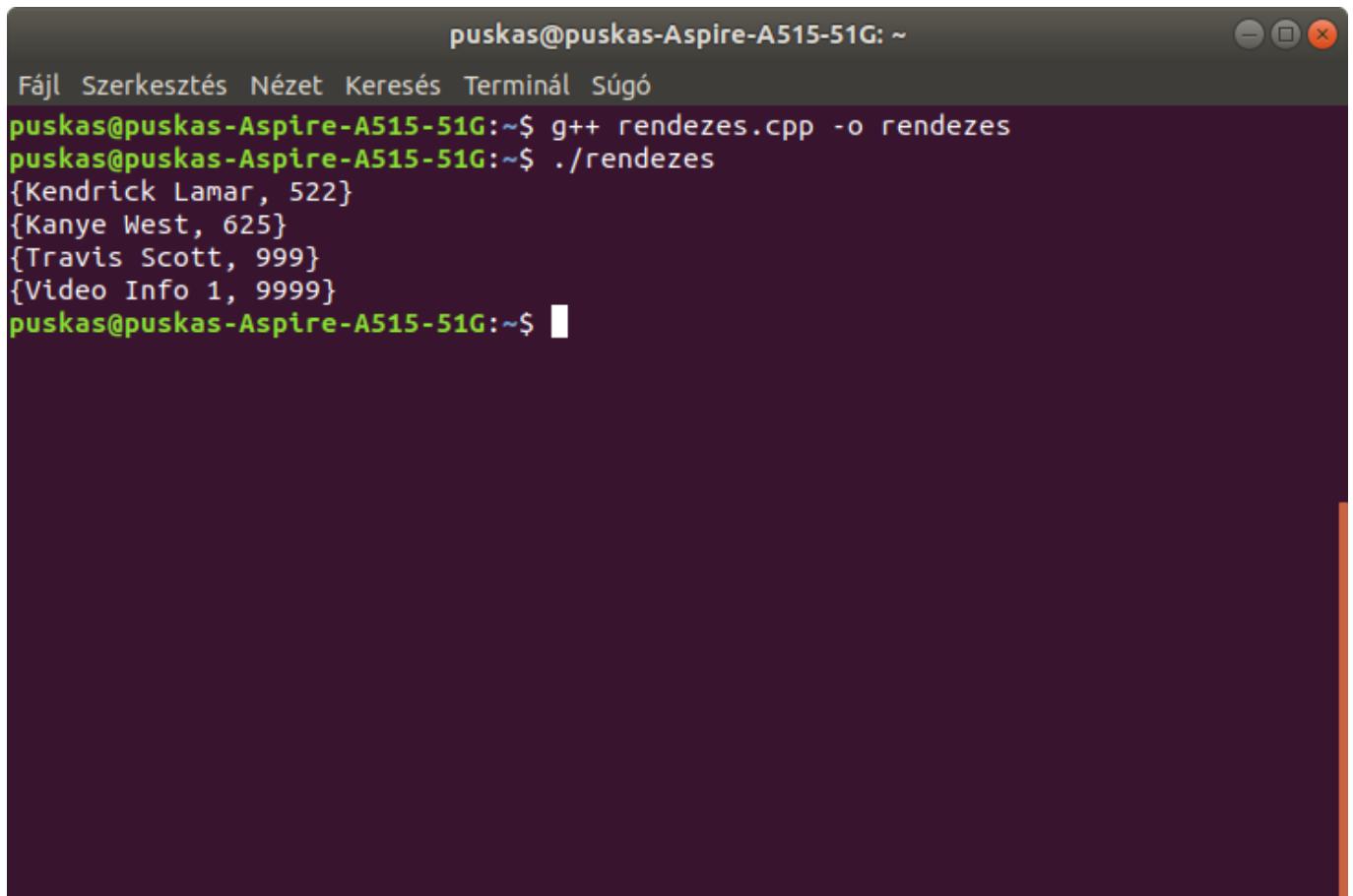
```
std::sort(vec.begin(), vec.end(),
          [] (const pair& l, const pair& r) {
              return l.second < r.second;
});
```

A sort függvénynek átadjuk paraméterül a vektorunkat, és a párokat rendezzük. A rendezés mapérték szerint történik, vagyis a bal oldali pár második (map) értéke legyen kisebb a jobb oldali pár mapértékétől.

```
for(auto const &pair: vec){
    std::cout << '{' << pair.first << ", " << pair.second << '}' << std::endl;
}
```

Ezután egy for each ciklussal végigmegyünk a vektoron és kiíratjuk a párok kulcs és adatértékét.

A program futásáról készült kép:



```
puskas@puskas-Aspire-A515-51G: ~
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~$ g++ rendezes.cpp -o rendezes
puskas@puskas-Aspire-A515-51G:~$ ./rendezes
{Kendrick Lamar, 522}
{Kanye West, 625}
{Travis Scott, 999}
{Video Info 1, 9999}
puskas@puskas-Aspire-A515-51G:~$
```

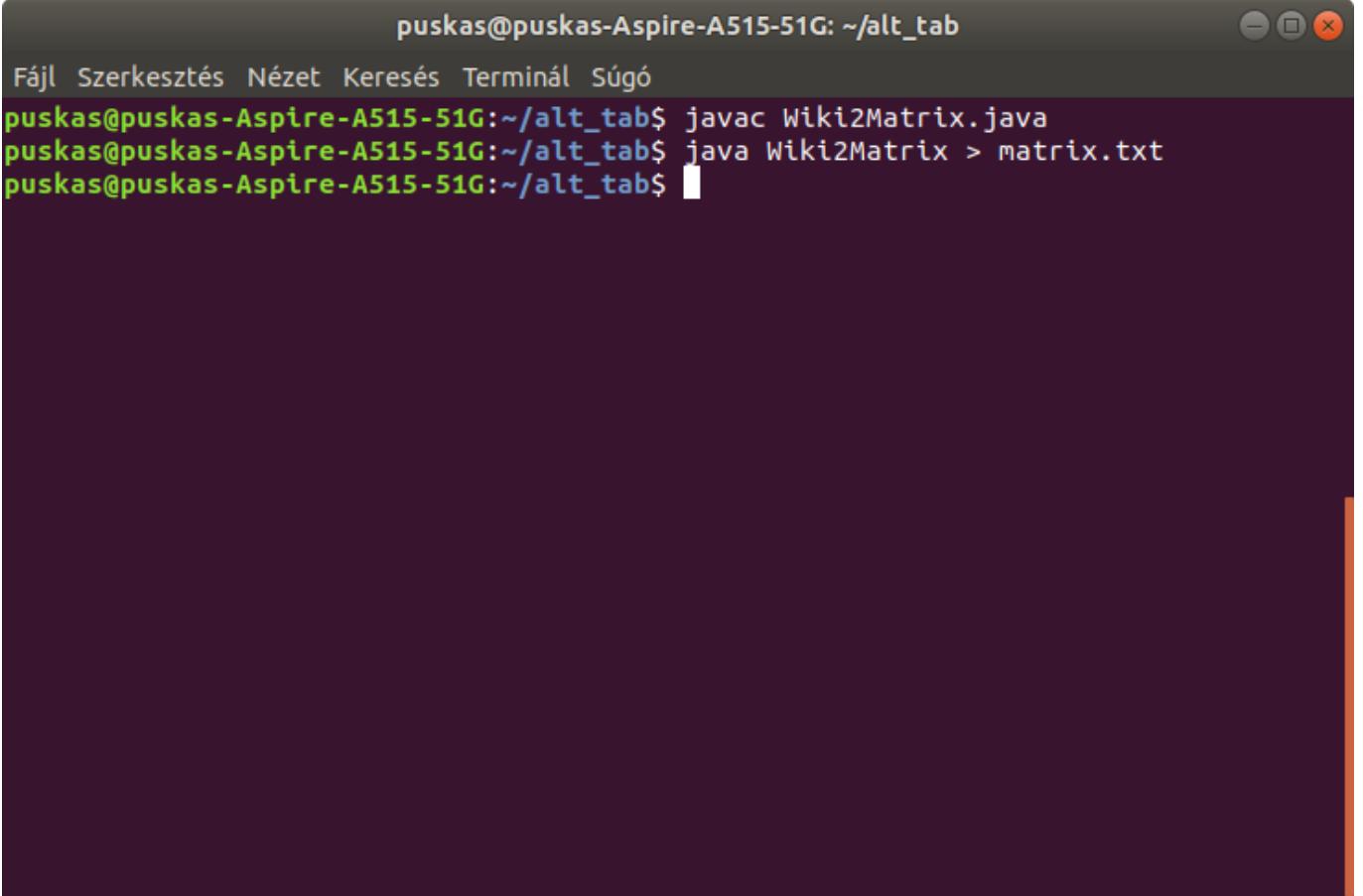
14.3. Alternatív tabella

Feladatunk az volt, hogy mutassunk rá a programban a `java.lang.Interface Comparable<T>` szerepére.

Az alternatív tabella egy, a szokásostól eltérő módon állítja sorrendbe a csapatokat. Az eredeti pontozás ugyanis úgy működik, hogy egy csapat egy győzelemért 3, döntetlenél 1, vereségért pedig 0 pontot kap. Az alternatív tabellában ez másképp van, ugyanis itt az is figyelembe van véve, hogy milyen erősséggű egyik illetve másik csapat, így itt nem lesz egyenértékű, ha pl egy erős csapat egy gyenge vagy egy erős csapatot győz le.

A projekt működéséhez először a `Wiki2Matrix.java` programra van szükség. Ebben át kell írnunk néhány dolgot, hogy a tabella az aktuális csapatokat tartalmazza. Ezután futtatjuk, és egy `txt`-be beleírjuk a kimenetét. Ez előállítja nekünk a mátrixot, amire szükségünk lesz.

Ez a következőképp történik:

A screenshot of a terminal window titled "puskas@puskas-Aspire-A515-51G: ~/alt_tab". The window contains the following text:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~/alt_tab$ javac Wiki2Matrix.java
puskas@puskas-Aspire-A515-51G:~/alt_tab$ java Wiki2Matrix > matrix.txt
puskas@puskas-Aspire-A515-51G:~/alt_tab$
```

The terminal has a dark background and light-colored text. The title bar and window controls are visible at the top.

Ez a txt-be egy linkmátrixot fog nekünk generálni. Ami lentebb látható:

Ezután nincs más dolgunk, mint ezt a linkmátrixot az AlternativTabella.java L nevű kétdimenziós tömbjébe bemásolni, és ebben a programban is aktualizálni a csapatok nevét, pontját.

Ezután ha futtatjuk, láthatjuk az aktuáli tabellát:

```
puskas@puskas-Aspire-A515-51G: ~/alt_tab
Fájl Szerkesztés Nézet Keresés Terminál Súgó
norma = 0.009455786115962213
φsszeg = 0.9999999999999997
***
0.08997865094516633
0.12009327027808978
0.07575767716539045
0.10749660836285888
0.052391857703979944
0.07292288727378253
0.09346964541402003
0.08360553434843401
0.10851105424424129
0.08422951306493812
0.11154330119909847
****

Csapatok rendezve:

| -
| Ferencváros
| 46
| Debreceni VSC
| 0.1200
| -
```

Itt látható számunkra a lényeg:

```
java.util.List<Csapat> rendezettCsapatok = java.util.Arrays.asList(csapatok ←
);
java.util.Collections.sort(rendezettCsapatok);
```

Ez egy rendezettCsapatok nevű listát készít a csapatokból. Ez úgy történik, hogy a csapatok tömbre ráengedjük a java.util.Arrays.asList függvényt. Majd a Collenctions.sort függvénytel rendezi ezeket. A csapatok rendezése a compareTo függvény alapján fog történni, ami lentebb látható.

```
class Csapat implements Comparable<Csapat> {

    protected String nev;
    protected double ertek;

    public Csapat(String nev, double ertek) {
        this.nev = nev;
        this.ertek = ertek;
    }

    public int compareTo(Csapat csapat) {
        if (this.ertek < csapat.ertek) {
            return -1;
        } else if (this.ertek > csapat.ertek) {
            return 1;
        }
    }
}
```

```
    } else {
        return 0;
    }
}
```

Ebben az osztályban történik az összehasonlítás. Felvesszük a szükséges értékeket, majd a Comparable osztályban a compareTo függvény fog meghívódni. A Comparable interfészt a java.lang csomag tartalmazza, és egy függvényt, a compareTo-t tartalmazza. A compareTo függvény két paramétert kér, két Csapat objektumot, majd ezeket megvizsgálja, ha a vizsgálandó csapat értéke kisebb mint a paraméterként kapott csapat értéke, akkor -1-et ad vissza, ha nagyobb, 1-et ad vissza, ha pedig egyenlő akkor nullát. A fentebb deklarált rendezettCsapatok esetében a csapatokra a compareTo függvény fog meghívódni, tehát a lista rendezése ezzel a függvénnel fog történni.

15. fejezet

Helló, Valami!

15.1. FUTURE Tevékenység Editor

Feladatunk az volt, hogy a FUTURE Tevékenység Editor java forrásán keressünk egy hibát, majd pedig javítsuk ezt.

Mielőtt hozzákezdenénk, egy kis ismertető, hogy mi is az a tevékenység editor.

A tevékenység editor egy olyan program, mely lehetőséget ad arra, hogy számon tartson, az adott napon mit vittünk véghez, és ezért milyen tapasztalatot szerezhettünk, és mennyit.

Mindezek előtt viszont arra is van lehetőségünk, hogy magunkat jellemizzük más felhasználók számára. Például megtudjuk adni a korunkat, a súlyunkat és a magasságunkat, valamint adhatunk egy becslést, hogy a saját képességünket, tapasztalatunkat és tudásunkat hány ponttal illetnénk.

A programban altevénységeket hozhatunk létre, mint például egy LoL verseny, és ezt tulajdonságokkal ruházzatjuk fel, mint pl. koncentráció, gondolkodás stb., és a program egy könyvtárszerkezetet fog létrehozni.

A programban rövid használat után számos hiba fedezhető fel. Egyik legszembetűnőbb talán, hogy egy tevékenységhez csak egy altevénységet enged létrehozni.

Ennek az az oka, hogy az altevénységeket ugyanazon a néven akarja létrehozni

The screenshot shows a terminal window and a file explorer window. The terminal window displays the command:

```
puskas@puskas-Aspire-515-31U:/asa/future/cs/10$ java AltevencyEditor
```

and its output:

```
Cannot create City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL/Új altevénység
Cannot create City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL/Új altevénység
Cannot create City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL/Új altevénység/Új altevénység
```

The file browser window shows a directory structure under 'Csak' (Only) in 'Csak' (Only) folder:

- City/Debrecen/Sport/Esport/DEAC-Hackers
- City/Debrecen/Sport/Esport/DEAC-Hackers/DEAC-Hackers.props
- City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny
- City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL
- City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL.props
- City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL/Új altevénység
- City/Debrecen/Sport/Esport/DEAC-Hackers/Verseny/LoL/Új altevénység/Új altevénység

Ezt a hibát egyszerűen kijavíthatjuk. A forráskódban ez a rész lesz az új altevékenység létrehozásáért felelős:

```
subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> {
    java.io.File file = getTreeItem().getValue();

    java.io.File f = new java.io.File(file.getPath() + System. ←
        getProperty("file.separator") + "Új altevékenység");

    if (f.mkdir()) {
        javafx.scene.control.TreeItem<java.io.File> newAct
//          = new javafx.scene.control.TreeItem<java.io. ←
        File>(f, new javafx.scene.image.ImageView(actIcon));
        = new FileTreeItem(f, new javafx.scene.image. ←
            ImageView(actIcon));
        getTreeItem().getChildren().add(newAct);
    } else {
        System.err.println("Cannot create " + f.getPath());
    }
});
```

Ezt a problémát könnyen kiküszöbölhetjük, ha az egészet egy végtelen ciklusba tesszük, és az új altevékenység után egy számlálót helyezük el, és az új altevékenységeket megszámozzuk, ezáltal nem ugyanaz lesz a nevük. A kijavított változat így néz ki:

```
subaMenuItem.setOnAction((javafx.event.ActionEvent evt) -> {
    int i=1;
    while(true) {
        java.io.File file = getTreeItem().getValue();

        java.io.File f = new java.io.File(file.getPath() + System. ←
            getProperty("file.separator") + "Új altevékenység"+i);

        if (f.mkdir()) {
            javafx.scene.control.TreeItem<java.io.File> newAct
//              = new javafx.scene.control.TreeItem<java.io. ←
            File>(f, new javafx.scene.image.ImageView(actIcon));
            = new FileTreeItem(f, new javafx.scene.image. ←
                ImageView(actIcon));
            getTreeItem().getChildren().add(newAct);
            break;
        } else {
            ++i;
            System.out.println("Nem hozható létre a(z) "+f.getPath ←
                () + ", ezért más néven lett létrehozva.");
        }
    }
});
```

});

Egy while ciklusban növeljük az i értékét, és hozzáírjuk az aktuális i-t a altevékenység nevéhez. A feltételes utasításban megnézzük, hogy létre lehet-e hozni az adott néven az altevékenységet (mappát). Amennyiben létrehozható, létrehozza, majd tevékenység editorban is megjeleníti grafikusan, és kilép a ciklusból. Hamis ágon pedig növeli az i értékét, és jelzi a felhasználónak, hogy más néven lett létrehozva, így megint az igaz ág fog teljesülni, amiben pedig kilép a ciklusból, így a program nem fog lefagyni.

Láthatjuk, hogy a program már képes létrehozni több altevékenységet is.

15.2. SamuCam

Az volt a feladatunk, hogy mutassunk rá a SamuCam programban a kamerakezelésre.

A projekttel kapcsolatban annyit kell tudni, hogy az opencv könyvtárat telepíteni kell, máskülönben nem tudjuk felépíteni illetve futtatni a programot.

A program annyit csinál, hogy képes felismerni emberi arcokat. Amennyiben a kamera egy arcot rögzít, vagy valamilyen fényképet, amin egy arc található, kis idő elteltével képes megjegyezni az arcot.

Ezért fontos futtatás előtt letöltenünk a lbpcascade_frontalface.xml fájlt, ugyanis ez lesz felelős az emberi arc felismeréséért.

Amennyiben azt akarjuk, hogy a laptopunk webkamerája képét rögzítse a program, a következő sort kell megkeresnünk:

```
videoCapture.open ( videoStream );
```

Ez a sor a SamuCam.cpp forráskódban lesz, méghozzá a void SamuCam::openVideoStream() függvényben szerepel. Ahhoz, hogy a webkameránk képe legyen rögzítve, a "videoStream"-et írjuk át 0-ra.

Térjünk is át a kamerakezelésre.

```
SamuCam::SamuCam ( std::string videoStream, int width = 176, int height = ←
    144 )
: videoStream ( videoStream ), width ( width ), height ( height )
{
    openVideoStream();
}
```

Láthatjuk, hogy a SamuCam konstruktorában a egy videoStream nevű string, és szélesség illetve magasság méretek lesznek tárolva. A videoStream string megkapja a videoStream értéke által a kamera ip címét, a width és height miatt pedig a méretei lettek beállítva.

```
void SamuCam::openVideoStream()
{
    videoCapture.open ( 0 );

    videoCapture.set ( CV_CAP_PROP_FRAME_WIDTH, width );
    videoCapture.set ( CV_CAP_PROP_FRAME_HEIGHT, height );
    videoCapture.set ( CV_CAP_PROP_FPS, 10 );
}
```

Az openVideoStream függvényben adjuk meg a kamera ip-címét, és azt, hogy mekkora felbontású képet rögzítsen, hány fps-sel.

Következő függvényünk a run függvény lesz:

```
void SamuCam::run()
{
    cv::CascadeClassifier faceClassifier;

    std::string faceXML = "lbpcascade_frontalface.xml"; // https://github.com ←
        /Itseez/opencv/tree/master/data/lbpcascades

    if ( !faceClassifier.load ( faceXML ) )
    {
        qDebug() << "error: cannot found" << faceXML.c_str();
        return;
    }
}
```

Létrehozunk egy cv::CascadeClassifier (az opencv miatt van rá lehetőségünk) változót, amely azért lesz felelős, hogy betöltsse a lbpcascade_frontalface.xml-t, ami ugye az emberi arcot írja le. Az xml nevét egy string-ben eltároljuk, és a load metódussal rávizsgálunk, hogy betudja e tölteni a CascadeClassifier az említett xml-t. Amennyiben nem, mert nem található meg, error-t fog dobni, és egyértelműen jelzi a felhasználó számára a hibát. Az xml egyébként a kommentben található linken szerezhető be.

```
while ( videoCapture.isOpened() )
{
    QThread::msleep ( 50 );
    while ( videoCapture.read ( frame ) )
    {

        if ( !frame.empty() )
        {

            cv::resize ( frame, frame, cv::Size ( 176, 144 ), 0, 0, cv::INTER_CUBIC );

            std::vector<cv::Rect> faces;
            cv::Mat grayFrame;

            cv::cvtColor ( frame, grayFrame, cv::COLOR_BGR2GRAY );
            cv::equalizeHist ( grayFrame, grayFrame );

            faceClassifier.detectMultiScale ( grayFrame, faces, 1.1, 3,
                0, cv::Size ( 60, 60 ) );

            if ( faces.size() > 0 )
            {

                cv::Mat onlyFace = frame ( faces[0] ).clone();

                QImage* face = new QImage ( onlyFace.data,
                                            onlyFace.cols,
                                            onlyFace.rows,
                                            onlyFace.step,
                                            QImage::Format_RGB888 );

                cv::Point x ( faces[0].x-1, faces[0].y-1 );
                cv::Point y ( faces[0].x + faces[0].width+2, faces[0].y + faces[0].height+2 );
                cv::rectangle ( frame, x, y, cv::Scalar ( 240, 230, 200 ) );

                emit faceChanged ( face );
            }

            QImage* webcam = new QImage ( frame.data,
                                         frame.cols,
                                         frame.rows,
                                         frame.step,
                                         QImage::Format_RGB888 );
        }
    }
}
```

```
    emit webcamChanged ( webcam ) ;

}

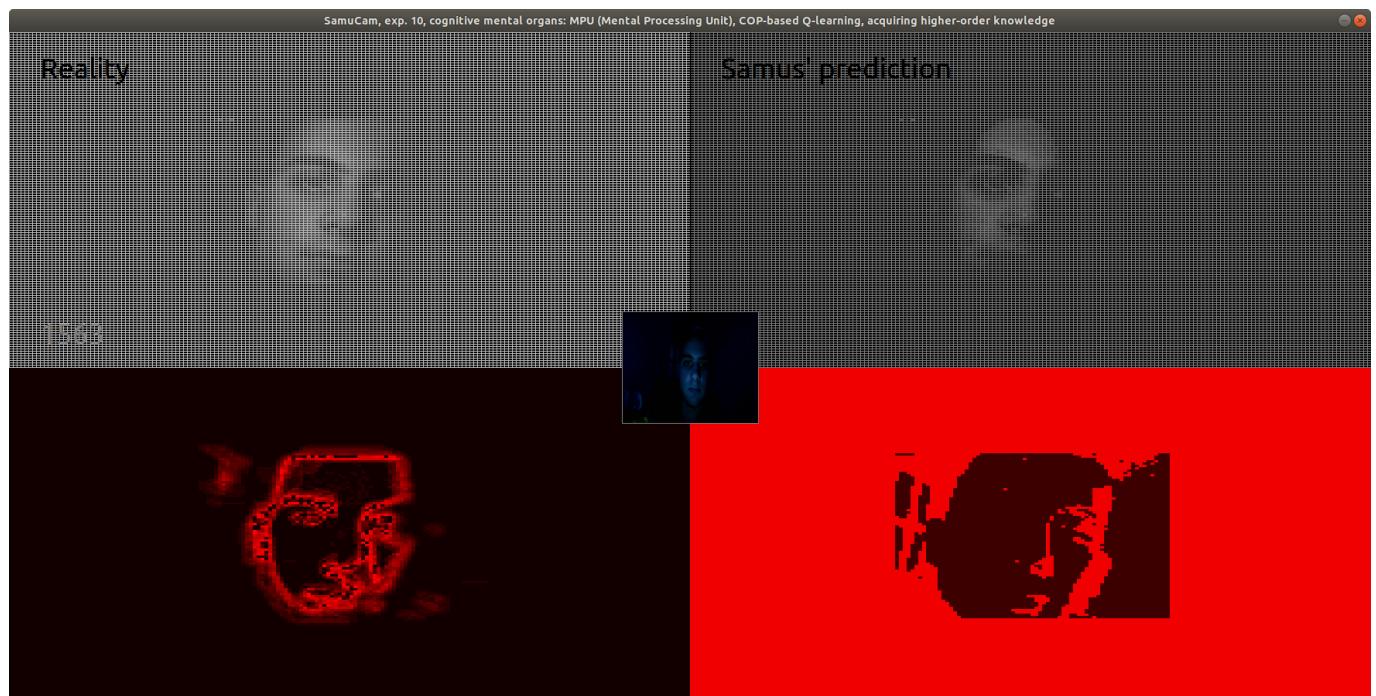
QThread::msleep ( 80 ) ;

}

if ( ! videoCapture.isOpened() )
{
    openVideoStream();
}

}
```

Az első while ciklus 50ms-re le fog állni. A második while ciklus a képkockákat fogja betölteni egyesével a read függvény segítségével. Ez a ciklus addig fog futni, amíg tudunk képkockát beolvasni. A feltételes utasítás valahogy így néz ki: amíg a frame nem üres, addig a beolvasott képkockát újraméretezzük, valamint az INTER_CUBIC nevű interpolációs metódussal interpoláljuk. A cv::Mat metódussal a face adatokat fogjuk szűrni és az első ilyen adatból egy QIMAGE objektumot építünk fel. A faceChanged az emit miatt egy signált fog küldeni, és frissíti az arcot a face adatokkal, majd a webcamChanged szintén egy signált küld, ezért a webcam képe frissülni fog, és 80ms-re leállítja a ciklust. Amint elfogynak a képkockák, kilépünk a ciklusból, és ha nincs elindítva a videoCapture, akkor elindítjuk.



Éeés a futtatásról készült kép, amelyen a telefon vakujával kellett megvilágítanom a fejem, mert valami fekete mágia folytán semmit nem lehet a webkamerámon látni felkapcsolt villany alatt. :D

15.3. BrainB

A BrainB projekt egy olyan játékról szól, melyben egy Samuentropy nevű, mozgó objektumon kell tartunk az egérmutatót. Ez elsőre egyszerűnek tűnik, viszont amíg rajta tartjuk az egeret, egyre több másik mozgó objektum jelenik meg, és sokszor le is fedik Samut, illetve egyre gyorsabb a mozgás is. Samu elvesztése esetén elkezdenek törlődni illetve lassulni az objektumok.

Feladatunk az volt, hogy miután felélesztettük a projektet, a Qt slot-signal mechanizmus működését fejtsük ki a programban.

Először ejtsünk pár szót magáról a mechanizmusról.

A Qt-t a slot-signal mechanizmus teszi egyedivé. Ez a mechanizmus a callback függvények összerende-lését teszi lehetővé. A Qt-ban az objektumok signalokat küldhetnek, melyek akkor kerülnek elküldésre, hogyha az adott objektummal valami történik. Az objektumoknak viszont Qt-ban vannak slotjai is, ezek tagfüggvények, melyek arra figyelnek, ha egy másik objektum signált küld, vagyis ezek a függvények a signal elküldése után hívódnak meg.

A slotokat köthetjük signalokhoz is, ami annyit tesz, hogy egy slot csak egy adott signal-ra fog figyelni. Az összekötés a connect függvényel történik, melynek szintaktikája a következő: connect (objektum_1, signal, objektum_2).

A signalok akkor is elküldődnek, ha nincs hozzájuk kötve slot.

A BrainBWin.cpp BrainBWin objektum konstruktőrában megtalálható a következő kódcsipet:

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

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

Ezen két connect függvényben jól látható a fentebb már említett szintaktika. A felső annyit tesz, hogy a brainBThread objektumban ha a hősök pozíciója megváltozik, azaz a heroesChanged függvény meghívódik, akkor szintén a brainBThread objektumban az updateHeroes függvénynek meg kell hívódnia, hogy a hősök pozíciója megváltozzon.

Az alsó függvény pedig azt csinálja, hogyha az endAndStats függvény meghívódik, elküldi a jelet lényegében saját magának, és meghívja megint az endAndStats függvényét, amely ha a forráskódban megnézzük, láthatjuk, hogy kilép a programból.

```
void BrainBWin::endAndStats ( const int &t )
{
    qDebug() << "\n\n\n";
    qDebug() << "Thank you for using " + appName;
    qDebug() << "The result can be found in the directory " + statDir;
    qDebug() << "\n\n\n";

    save ( t );
    close();
}
```

A futásról egy kép:



16. fejezet

Helló, Lauda!

16.1. Port scan

A port scan egy olyan program, amely a hálózat nyitott portjait képes megkeresni.

Port szkennert általában adminisztrátorok használnak, hálózatok biztonságának ellenőrzése érdekében. A porgram azt csinálja, hogy megnézi, hogy egy hostot hány port figyel.

A portszkenner forráskódja a következő:

```
public class KapuSzkenner {

    public static void main(String[] args) {

        for(int i=0; i<1024; ++i)

            try {

                java.net.Socket socket = new java.net.Socket(args[0], i);

                System.out.println(i + " figyeli");

                socket.close();

            } catch (Exception e) {

                System.out.println(i + " nem figyeli");

            }
    }
}
```

A program annyit fog csinálni, hogy 1024-ig indít egy for ciklust, majd megpróbál a parancssorban megadott ip című gép adott "i" portjával TCP kapcsolatot létesíteni. A programban azért szükséges kivételezés, mivel ha nem tud kapcsolódni az adott porthoz, vagyis nem figyeli, kivételt fog dobni, és leáll a

program. Ez elkerülhető, ugyanis a kivételkezelés miatt ilyenkor a catch blokk utasításai fognak végrehajtódni, jelen esetben kiírja, hogy az adott portot egy folyamat sem figyeli.

Sikeres kapcsolódás esetén a catch blokk viszont nem fog végrehajtódni, így jelezni fogja a program felénk, hogy az adott portot egy folyamat figyeli.

A program futásáról készült kép:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G: ~
619 nem figyeli
620 nem figyeli
621 nem figyeli
622 nem figyeli
623 nem figyeli
624 nem figyeli
625 nem figyeli
626 nem figyeli
627 nem figyeli
628 nem figyeli
629 nem figyeli
630 nem figyeli
631 figyeli
632 nem figyeli
633 nem figyeli
634 nem figyeli
635 nem figyeli
636 nem figyeli
637 nem figyeli
638 nem figyeli
639 nem figyeli
640 nem figyeli
641 nem figyeli
642 nem figyeli
```

Ha a main függvényünk így néz ki, azaz kivesszük a kivételkezelést:

```
/*for(int i=0; i<1024; ++i)

    try {*}*
        int i=0;
        while(i<1024) {
            java.net.Socket socket = new java.net.Socket(args[0], i);

            System.out.println(i + " figyeli");

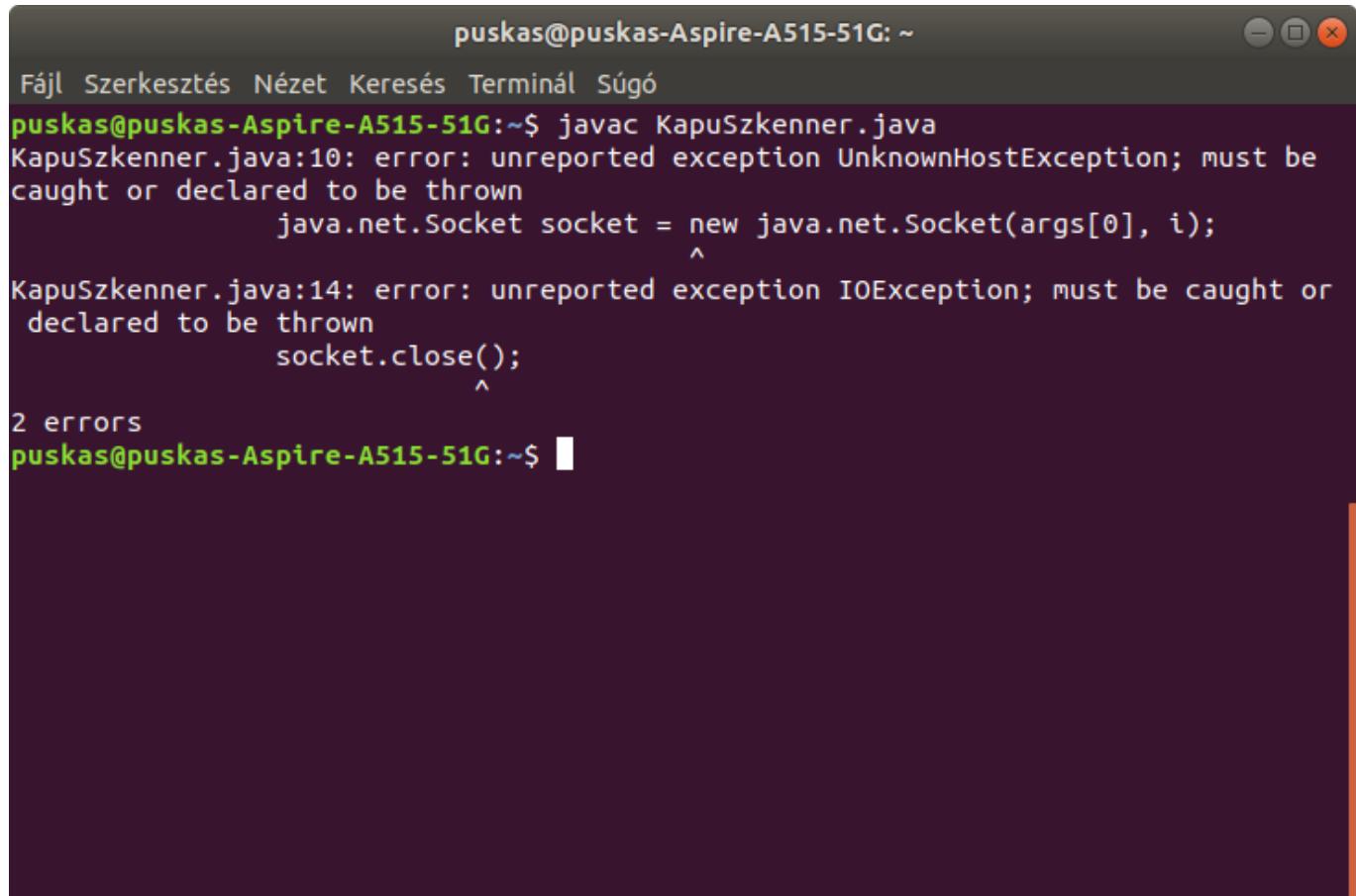
            socket.close();
            ++i;
        }

    /* } catch (Exception e) {

        System.out.println(i + " nem figyeli");
```

```
} */
```

Ez a következőt fogja jelenteni: a while ciklus 1024-ig fut, és az adott ip című gép i portjával próbál TCP kapcsolatot létesíteni. Jelen esetben nincs kivételkezelés, tehát ha nem sikerül TCP kapcsolatot létesíteni az aktuális i porttal, akkor nincs végrehajtandó catch ág, ami kivétel esetén hívódik meg, így a fordító hibát fog dobni, ami alább látható:



A screenshot of a terminal window titled "puskas@puskas-Aspire-A515-51G: ~". The window shows the following text output:

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~$ javac KapuSzkenner.java
KapuSzkenner.java:10: error: unreported exception UnknownHostException; must be
caught or declared to be thrown
        java.net.Socket socket = new java.net.Socket(args[0], i);
                           ^
KapuSzkenner.java:14: error: unreported exception IOException; must be caught or
declared to be thrown
        socket.close();
                ^
2 errors
puskas@puskas-Aspire-A515-51G:~$
```

16.2. AOP

Az aspektus orientált programozás összetett programok esetében hasznos. A programok áttekinthetősége és újrafelhasználhatósága érdekében aspektusokat fűzhetünk programunkhoz. Ez annyit jelent, hogy az alap forráskód módosítása nélkül leszünk képesek valamiféle módosítást végezni a programon, hogy futáskor a várt eredményt kapjuk.

Az aspektusokat aspektusnyelven írjuk, mely nagyban hasonlít a forráskód megírásakor használt objektumorientált programozási nyelvre. A megírt aspektust "hozzáfűzzük" az alaprogramhoz, ezáltal a forráskód átírása nélkül tudjuk a program viselkedését változtatni.

Az aspektus tehát a módosított programrész tartalmazza, így önmagában nem felhasználható. Jelenleg az AspectJ a legnépszerűbb és legfejlettebb aspektus orientált nyelv.

A feladatunk az volt, hogy a BinFa java átitratába szőjünk bele egy aspektust.

A feladat megoldásakor a Prog1-es bejárásokat használtam alapul, ugyanis a programhoz fűzött aspektusom a preorder kiírást fogja megvalósítani.

A feladat megoldásához szükségünk lesz az aspectjrt.jar állományra, melyet innen tudunk beszerezni: <https://mvnrepository.com/artifact/org.aspectj/aspectjrt>

```
privileged aspect LZWBInFaa {
```

Először is implementáljuk az aspektust.

```
void around(LZWBInFa.Csomopont elem, java.io.PrintWriter os , LZWBInFa f) :  
    target (f) && call (void kiir (LZWBInFa.Csomopont, java.io.PrintWriter) ←  
        ) && args (elem,os)
```

Az around advice az advice-ban megadott függvény viselkedését fogja megváltoztatni az általunk implementált módon. Paraméterül az around advice minden olyan elemet megkap, ami használva lesz. Jelen esetben a kiir függvény paramétereit kapja meg, illetve egy LZWBInFa objektumot, ugyanis a kiir függvényt egy LZWBInFa objektumra hívjuk meg. Célba vesszük az f objektmot a target kulcsszóval, ez lényegében annyit jelent, hogy erre fogjuk meghívni a függvényt, majd hívjuk a kiir függvényt a call kulcsszóval, melyet az eredeti paramétertípusaival implementálunk, majd argumentumként adjuk a neki a Csomopont típusú elem objektumot és az outputstreamet. Szóra fordítva ez a forráskód annyit csinál, hogy ha f-re meghívjuk a kiír függvényt, módosítsa a függvény funkcióját a lentebb láthatóra.

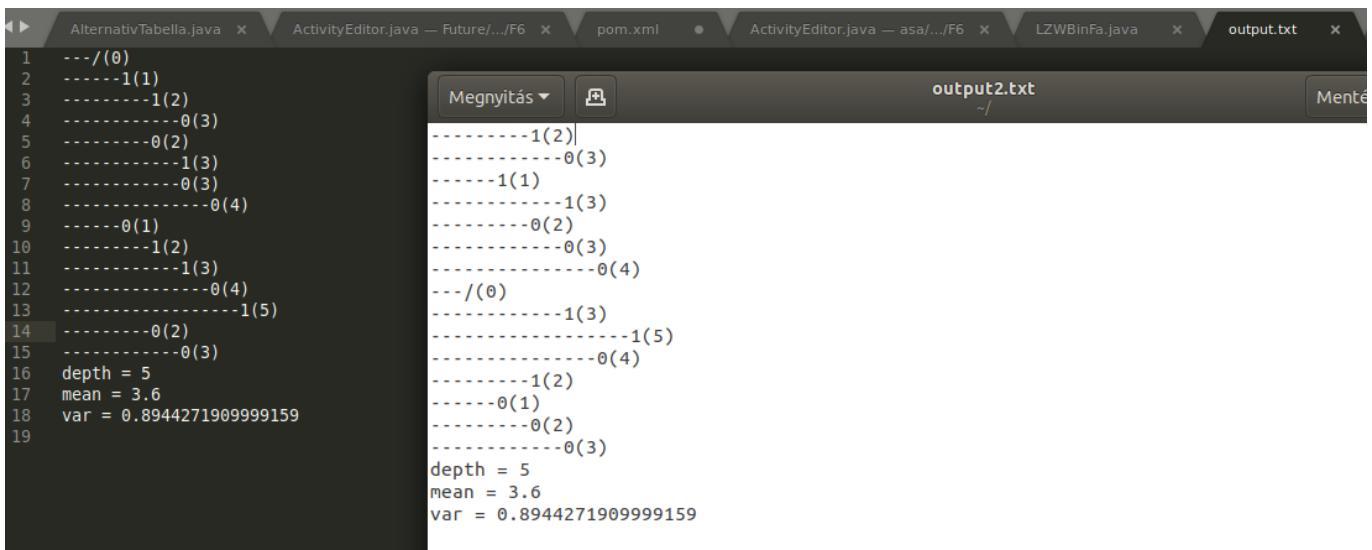
```
{  
if (elem != null) {  
    ++f.melyseg;  
    for (int i = 0; i < f.melyseg; ++i) {  
  
        os.print("---");  
    }  
    os.print(elem.getBetu());  
    os.print("(");  
    os.print(f.melyseg - 1);  
    os.println(")");  
    f.kiir(elem.egyesGyermek(), os);  
    f.kiir(elem.nullasGyermek(), os);  
  
    --f.melyseg;  
}  
}  
}
```

Ezután annyi dolgunk van még, hogy az új kiir függvényt implementáljuk, jelen esetben ez a postorder megvalósítást jelenti.

A programhoz a következőképp tudjuk hozzáfűzni az aspektust, majd pedig futtatni:

```
puskas@puskas-Aspire-A515-51G: ~
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~$ ajc LZWBInFa.java LZWBInFaa.aj
puskas@puskas-Aspire-A515-51G:~$ java -classpath aspectjrt.jar:. LZWBInFa ures.txt -o output.txt
puskas@puskas-Aspire-A515-51G:~$
```

Láthatjuk, hogy a tényleg változott az output:



The screenshot shows a Java development environment with several tabs open at the top: AlternativTabella.java, ActivityEditor.java — Future/.../F6, pom.xml, ActivityEditor.java — asa/.../F6, LZWBInFa.java, and output.txt. Below the tabs, there are two code editors. The left editor contains the following code:`1 ---/(0)
2 -----1(1)
3 -----1(2)
4 -----0(3)
5 -----0(2)
6 -----1(3)
7 -----0(3)
8 -----0(4)
9 -----0(1)
10 -----1(2)
11 -----1(3)
12 -----0(4)
13 -----1(5)
14 -----0(2)
15 -----0(3)
16 depth = 5
17 mean = 3.6
18 var = 0.8944271909999159
19`

```
The right editor shows the contents of the output.txt file:
```

`Megnyitás ▾
output2.txt
~/ Menté
-----1(2)
-----0(3)
-----1(1)
-----1(3)
-----0(2)
-----0(3)
-----0(4)
---/(0)
-----1(3)
-----1(5)
-----0(4)
-----1(2)
-----0(1)
-----0(2)
-----0(3)
depth = 5
mean = 3.6
var = 0.8944271909999159`

16.3. Junit teszt

Feladatunk az volt, hogy az első védési program java átiratában teszteljük le a szórás és mélység kiszámítására használt függvény pontosságát.

A junittal képesek vagyunk egy forráskód bizonyos függvényei által visszadobott értéket ellenőrizni, összehasonlítani. Lássuk is a megoldást.

Először kézzel ki kellett számolnunk a "01111001001001000111" stringre a szórást és a mélységet, viszont ez megtalálható az alábbi poszton:

https://progpter.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozo_fel_egy_pedat

Másodszor pedig a junit.jar és hamcrest.jar állományokat kell beszereznünk, jelen esetben mindenből a legfrissebb verziót használom.

A következő dolgunk az ellenőrző program megírása. A forráskód alább látható:

```
import org.junit.Test;  
import static org.junit.Assert.assertEquals;
```

Először is importáljuk a junit Test metódusát, amely a nevéről adódóan a tesztelésért felelős, majd szintén a junit-ból az assertEquals függvényt, amely a kapott értéket a kívánt értékkel fogja összehasonlítani.

```
public class LZWBInFaTest {  
    LZWBInFa binFa = new LZWBInFa();  
    String ertek = "01111001001001000111";  
    double melyseg = 4.0;  
    double szoras = 0.957427;
```

Következő dolgunk a használandó állományok implementálása. Jelen esetben egy LZWBInFa objektumot veszünk fel, majd egy string-ben tároljuk a fentebb említett értéket, ugyanis erre fogjuk a szórást és a mélységet számolni. A melyseg és szoras változó a fentebb említett kézzel számolt mélység és szórás érték lesz.

```
@Test  
    public void tesBitFeldolg() {  
        for (char b : ertek.toCharArray()) {  
            binFa.egyBitFeldolg(b);  
        }  
    }
```

Következzen a test metódus. Ez a metódus lesz felelős a tesztelésért. A for ciklusban a stringet alakítjuk át char típusára és töltjük bele egy char tömbbe. Ezután az alapprogramunk egyBitFeldolg függényét hívjuk segítségül, és az eddig üres binFa objektumunkra meghívjuk ezt a függvényt, és paraméterül a b chart adjuk, ezáltal felépül a fa.

```
assertEquals(szoras, binFa.getSzoras(), 0.0001);  
assertEquals(melyseg, binFa.getMelyseg(), 0.0);  
    }  
}
```

A program utolsó részében az assertEquals metódust hívjuk meg, melynek paraméterül a szoras változót, és a fára a getSzoras függénnyel kiszámolt szórás értékét adjuk, 0.0001 eltérés megengedésével. A második assertEquals metódus lényegében ugyanez lesz a mélységre számolva, viszont itt nem történhet eltérés.

```
puskas@puskas-Aspire-A515-51G: ~/Unit
Fájl Szerkesztés Nézet Keresés Terminál Súgó
puskas@puskas-Aspire-A515-51G:~/Unit$ javac LZWBInFa.java
puskas@puskas-Aspire-A515-51G:~/Unit$ javac -cp .:junit-4.13-rc-1.jar LZWBInFaTest.java
puskas@puskas-Aspire-A515-51G:~/Unit$ java -cp .:junit-4.13-rc-1.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore LZWBInFaTest
JUnit version 4.13-rc-1
.
Time: 0,007

OK (1 test)

puskas@puskas-Aspire-A515-51G:~/Unit$
```

Láthatjuk, hogy a futtatáshoz először az alap forráskódot kell lefordítanunk. Majd a -classpath kapcsoló használatával a használni kívánt osztály elérési útját adhatjuk meg, mely a . miatt az adott mappa lesz, és inkudáljuk még a junit.jar állományt, majd a fordítani kívánt fájl nevét adjuk meg. Ezután futtatjuk, majd az eddig argumentumok után a hamcrest.jar-t is odaírjuk, az org.junit.runner.JUnitcore fogja futtatni az LZWBInFaTest osztályt a terminálból. Ahogy azt a képen is láthatjuk, a teszt azt eredményezte, hogy a függvényeink pontosak.

17. fejezet

Helló, Calvin!

17.1. MNIST

A feladat az volt, hogy oldjuk meg az alap mnist-es feladatot, illetve oldjuk meg, hogy saját, kézzel írott képet is felismerjen. A forrás itt található: https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Az **mnist_softmax_UDPROG61.py** forráskódra lesz szükségünk. Futtatáskor errorok sora fog fogadni minket. Ennek rövides utánanézés során hamar véget vethetünk.

Az első javítanivaló a program 70. sorában található:

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y , ←  
y_) )
```

A `tf.nn.softmax_cross_entropy_with_logits()` függvénynek ha az api-ban utánanézünk, láthatjuk, hogy az új verzió már más szintaktikát követ. Itt ugyanis nevesítenünk kell az `y` és `y_` értékeket, vagyis így kell kinézniük:

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits( ←  
logits=y, labels=y_) )
```

A következő errort forrását már nehezebb volt megtalálni. Ez ugyanis a kód 46. sorában található:

```
img = tf.image.decode_png(file)
```

A függvénynek utánanézve azt olvashatjuk, hogy egy channel paramétert is kér tőlünk. Ez ugye több értékkel is bírhat. Alapértelmezetten 0, amely a png képek dekódolására használatos, lehet 1, ami a grayscale képet ad vissza, lehet 2, ami rgb képet, és lehet 3, ami rgba képet ad vissza. Ahhoz, hogy megtudjuk, melyikre lesz szükségünk, az mnist database-re vessünk egy pillantást. Láthatjuk, hogy a tanulásra használt képek fekete hátérszínnel és fehér betűszínnel bírnak, illetve 28x28 pixelt tartalmaznak. Ez két dolgot eredményez, egyrészt a grayscale kép csatornájára, vagyis az egyesre lesz szükségünk, másrészt pedig az általunk felismertetni kívánt kép is ilyen értékekkel kell bírjon. Kezdjük is a forráskód elemzését.

```
def readimg():  
    file = tf.compat.v1.io.read_file("sajat8a.png")  
    img = tf.image.decode_png(file, channels=1)  
    return img
```

A readimg függvényünk a kép beolvasásáért lesz felelős, melye először beolvassa a grayscale képet, aztán dekódolja, végül visszaadja a képet.

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

A main-ben x néven egy placeholderet hozunk létre, mely egy olyan tensornak foglal helyet, minden kap "enni". Amikor nem etetjük meg, errort fog dobni. A placeholder függvény egy olyan értéket fog visszaadni, amelyet közvetlenül nem tudunk kiértékelni, csak felhasználni feedinghez. Az x értéke tehát egy valós szám lesz, amely a képünket kapja értékül. A placeholder paraméterei között a [None, 784] egy shape-ként szerepel, mely azt jelenti, hogy a placeholder első dimenziója a none miatt bármennyi lehet, itt most 1 lesz, mivel egy képünk van, a második dimenziója pedig 784 elemből áll. A W súlyértek lesz, a tf.Variable függvénynek paraméterül adjuk a pixelek számát (28x28), majd a számjegyek számát (0-9). A b érték egy olyan tensor súlyát adja vissza, melynek 10 nullás eleme van. Az y összeszorozza az x és a W mátrixot, majd hozzáadja a b-t. Majd kezdődik a tanítás és a tesztelés egy meglévő mnist modellel.

```
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
        print(i/10, "%")
print("-----")
```

A tanítás fentebb látható. A tanítás, mint ahogy a for ciklusban is látható 1000 képpel történik, majd 100 képenként kiírjuk az adott feldolgozási állapotot úgy, hogy a elosztjuk az i értékét tízzel, ezzel végülis 10%-onként történik a kiíratás.

Itt jön képbe a saját kép felismerése.

```
img = readimg()
image = img.eval()
image = image.reshape(28*28)
```

Először beolvassuk a képet a readimg függvényvel, majd az image-be kiírtékeljük az eval függvény segít-ségével, majd a reshape függvényvel újra leképezzük az image-et egy egydimenziós vektorra.

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

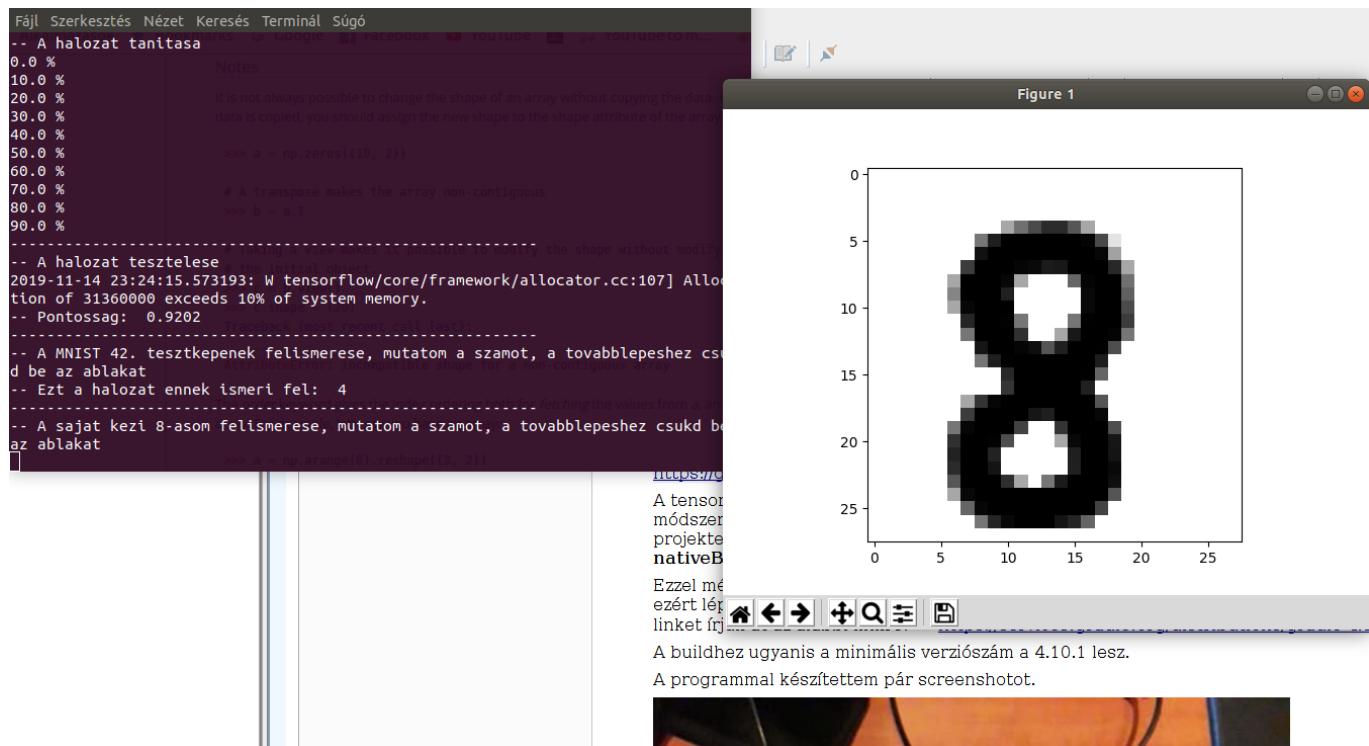
classification = sess.run(tf.argmax(y_conv, 1), feed_dict={x: [image], ←
    keep_prob: 1.0})

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

Matlab függvények következnek, elmentjük és kirajzoljuk a képet.

```
feed_dict={x: [image]}
```

A programban ez a kis kódcsipet fogja "etetni" az x-et a képunkkel.



A program sikeresen fut, és az általam rajzolt 8-as invertált képe fentebb kirajzolva lárható.

The figure shows a terminal window with the command `puskas@puskas-Aspire-A515-51G:~`. The window displays the same Python code as the Jupyter Notebook, with the output of the print statement showing the digit '8'.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
10.0 %
20.0 %
30.0 %
40.0 %
50.0 %
60.0 %
70.0 %
80.0 %
90.0 %
-----
-- A halozat tesztelese
2019-11-14 23:24:15.573193: W tensorflow/core/framework/allocator.cc:107] Allocation of 31360000 exceeds 10% of system memory.
-- Pontosság:  0.9202
-----
-- A MNIST 42. tesztképenek felismerése, mutatom a számot, a továbbíthatóban csukd be az ablakat
-- Ezt a halozat ennek ismeri fel:  4
-----
-- A saját kezi 8-asom felismerése, mutatom a számot, a továbbíthatóban csukd be az ablakat
-- Ezt a halozat ennek ismeri fel:  8
-----
puskas@puskas-Aspire-A515-51G:~$
```

A program sikeresen felismerte a kézzel rajzolt nyolcasomat, hurrá! :)

17.2. Deep Mnist

Feladatunk az előző feladathoz köthető. A program működése az előző programhoz nagyon hasonló, ez tulajdonképpen az előző verziónak a mély változata. A feladatban Rácz András volt segítségemre.

A forráskód ugyanazon repóban megtalálható, futtatáskor viszont errorok sorain kell végigzongorázunk, mivel deprecated a forráskód, ezért muszáj pár sort átírni. A függvényeket ugyanis az új verzióban egy compat.v1 előttaggal kell illetnünk, aminek a tf előtag és a függvény előtt közt kell szerepelnie.

Amint az errorokat kijavítottuk, futtatás előtt még a forráskódon további módosítások szükségesek, ugyanis a feladat ismételten az volt, hogy saját modellt is képes legyen felismerni.

```
def readimg():
    file = tf.compat.v1.read_file("sajat8a.png")
    img = tf.image.decode_png(file, 1)
    return img
```

Ehhez ugye az előző példával megegyezően definiálnunk kell egy readimg függvényt, ami a saját képünket fogja beolvasni, neve pedig ugyanaz lesz, mint az előző példában.

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

# Create the model
x = tf.compat.v1.placeholder(tf.float32, [None, 784])

# Define loss and optimizer
y_ = tf.compat.v1.placeholder(tf.float32, [None, 10])

# Build the graph for the deep net
y_conv, keep_prob = deepnn(x)

with tf.name_scope('loss'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=y_,
                                                            logits=y_conv)
cross_entropy = tf.reduce_mean(cross_entropy)

with tf.name_scope('adam_optimizer'):
    train_step = tf.compat.v1.train.AdamOptimizer(1e-4).minimize( ←
        cross_entropy)

with tf.name_scope('accuracy'):
    correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
    correct_prediction = tf.cast(correct_prediction, tf.float32)
accuracy = tf.reduce_mean(correct_prediction)

graph_location = tempfile.mkdtemp()
print('Saving graph to: %s' % graph_location)
train_writer = tf.compat.v1.summary.FileWriter(graph_location)
```

```
train_writer.add_graph(tf.compat.v1.get_default_graph())
```

A grayscale kép formázásáért a fentebb látható kódcsipet a felelős.

```
with tf.compat.v1.Session() as sess:  
    sess.run(tf.compat.v1.global_variables_initializer())  
    for i in range(20000):  
        batch = mnist.train.next_batch(50)  
        if i % 100 == 0:  
            train_accuracy = accuracy.eval(feed_dict={  
                x: batch[0], y_: batch[1], keep_prob: 1.0})  
            print('step %d, training accuracy %g' % (i, train_accuracy))  
            train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})  
  
        print('test accuracy %g' % accuracy.eval(feed_dict={  
            x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```

A fentebb látható rész a tanítás. Ez közvetlenül a saját kép beolvasása előtt helyezkedik el. 20000 képpel fog történni a tanítás. A feltételes utasításban az áll, hogy minden 100. kép beolvasása után kiíratjuk, hogy hol tartunk, és hogy mekkora pontossággal ismer fel a program képeket. Értelemszerűen minnél több képet tanult már meg a program annál nagyobb pontossággal ismer fel képeket. A tanítás része jóval részletesebben történik, mint az előző példa esetében, így ez a példa több futtatási időt eredményez, ez esetben 50 perc körül volt. A tanítás után következik a saját képünk beolvasása:

```
img = readimg()  
image = img.eval()  
image = image.reshape(28*28)  
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot. ←  
    cm.binary)  
matplotlib.pyplot.savefig("nyolc.png")  
matplotlib.pyplot.show()  
  
classification = sess.run(tf.argmax(y_conv, 1), feed_dict={x: [image], ←  
    keep_prob:})  
  
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
```

A fentebb látható kódcsipetet a main fölé kell beillesztenünk, ugyanis így érjük el ismételten, hogy saját képet olvasson be. Ez a rész megegyezik az előzőben található részzel, beolvassuk a képet, újraméretezzük, elmentjük és megjelenítjük.

Nézzük is meg, felismeri e a már előző példában használt nyolcast:

Láthatjuk, hogy a felismerés sikeres volt.

17.3. Android telefonra a TF objektum detektálója

Az volt a feladatunk, hogy a TensorFlow objektum detektálóját töltük le, és próbáljuk ki.

A program a telefon kameráján rögzített objektumokat próbálja megállapítani. A programot a következő

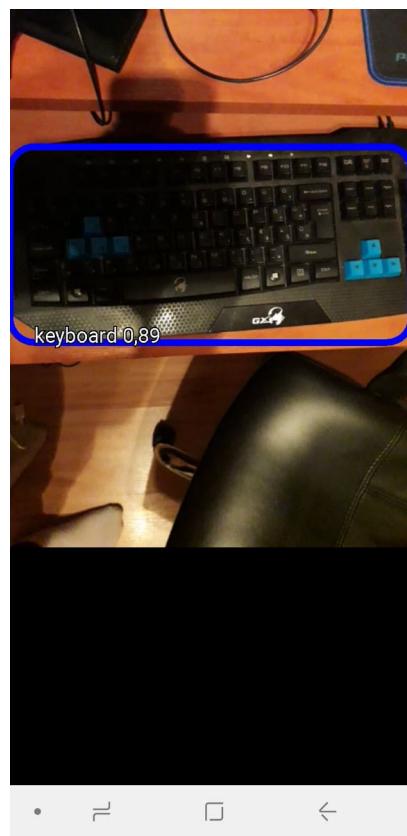
linken tudjuk beszerezni: <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android>

A tensorflow repó klónozása után megkezdhetjük az apk fájl építését. Ehhez a legegyszerűbb módszer az Android Studio segítségével történik. Még mielőtt azonban felépíténénk benne a projektet, a build.gradle állomány 45. sorában lévő stringet át kell írnunk erre: **def nativeBuildSystem = 'none'**

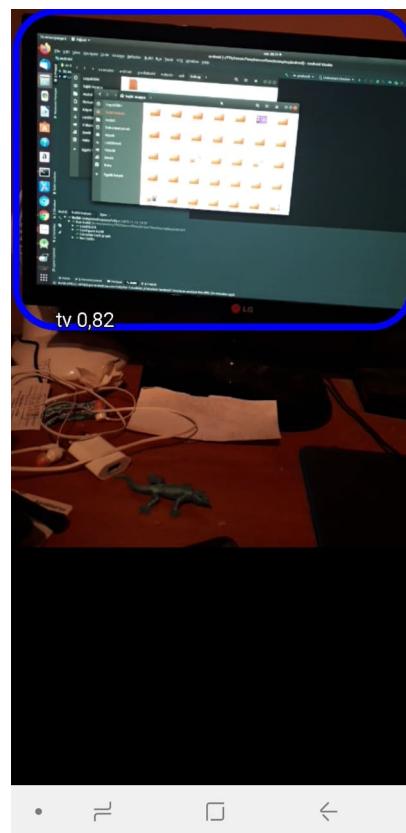
Ezzel még nem végeztünk, importáláskor errorba ütközhetünk a gradle verziószáma miatt, ezért lépjünk a gradle mappába, azon belül a wrapper mappába, az itt található **gradle-wrapper.properties** állomány 6. sorában található linket írjuk át az alábbi linkre: <https://services.gradle.org/distributions/gradle-4.10.1-all.zip>

A buildhez ugyanis a minimális verziószám a 4.10.1 lesz.

A programmal készítettem pár screenshotot.







Láthatjuk, hogy a program egész ügyesen képes felismerni a tárgyakat, viszont van még elég sok hiányossága, sajnos még túl sok objektumot téveszt el.

18. fejezet

Helló, Berners-Lee!

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

A Python programozási nyelv Guido van Rossum nevéhez fűződik, ugyanis ő alkotta meg ezt a magas szintű, dinamikus, objektumorientált és platformfüggetlen nyelvet, melyet leginkább prototípus készítésére illetve tesztelésére szoktak alkalmazni. A python egy egész hamar elsajátítható szkriptnyelv, mely sok beépített eljárást tartalmaz, így komolyabb problémák megoldására is alkalmas. Támogat magas szintű típusokat is, ezzel a fejlesztést könnyíti meg. Előnyei miatt a nyelvet mobil operációs platform alá is elkészítették, ezzel egyszerűbbé tették a szoftver és prototípusfejlesztést mobil készülékre.

A python forráskód esetében nincs szükség fordításra, ugyanis elegendő a forrást megadni, az pedig automatikusan futtatja is az alkalmazást. C/C++ és Java nyelven is egy azonos jellegű probléma kiküszöbölése nagyobb erőfeszítést jelent mint Python nyelven.

A magas szintű adattípusok miatt a Python képes sokkal nagyobb problémák leírására, mint amekkorát az awk vagy a Perl nyelv használatával le tudunk írni. A Python egy nagyon magas szintű programozási nyelv.

A Python nyelvhez egy kódkönyvtár is készült, amelyben rengeteg újrahasznosítható, előre megírt modult tartalmaz, ez felgyorsítja az alkalmazásfejlesztést. Ilyen modulok pl. fájlkezelésre, hálózatkezelésre és rendszerhívásokra használt modulok.

A Python egy köztes nyelv, nincs szükség fordításra és linkelésre.

A Python tömör, mégis könnyen olvasható programok készítését teszi lehetővé, ezek rövidebbek mint az ugyanilyen utasítássorozattal rendelkező C, C++ vagy akár Java programok.

A python nyelvben a változók definiálására nincs szükség, illetve sem zárójeleket, sem pedig pontos vesszőt nem használunk. Ezeket a forráskódokban új sorral illetve tabulátor használatával tudjuk helyettesíteni. A programblokkok itt behúzással jönnek létre, a végét pedig onnan láthatjuk, hogy az adott sor behúzása az előttelévőknél kisebb. Az utasítások a python-ban a sor végéig tartanak, ez engedi meg a pontosvesszők használatának mellőzését. Nyilvánvalóan előfordulhat, hogy egy utasítás meghaladja a sor hosszát, ilyenkor egy \ jel használatával érhetjük el, hogy a következő sort is utasításnak tekintse a python. Itt fontos még megjegyezni, hogy úgy, ahogyan a többi nyelven megírt forráskódban is, itt is helyezhetünk el a kódban megjegyzést, ezt a # jel használatával tudjuk megvalósítani.

A python-ban is előjönnek a pointerek, referenciaiák. Itt ugyanis a változók azok objektumra mutató referenciaiak lesznek. A referenciacat a „del” előtaggal tudjuk törölni, a memóriaáramlás megszorítása érdekében

pedig azt az objektumot, amelyre nem mutat referencia, törölni fogja nekünk a garbage collector. Pythonban tehát nincs a változóknak típusa. Na de akkor honnan tudja a program, hogy mégis milyen típust akarunk használni. A válasz meglepően egyszerű, és lényegre törő: kitalálja.

A változókat két típusra bonthatjuk:

1. Primitívek

- bool
- int
- float
- none

2. Objektumok

- str (string)
- list (lista)
- dict (szótár, asszociatív tömb)

Globális és lokális változókról beszélhetünk, a pythonban a többi nyelvtől eltérően az alapértelmezett változótípus a lokális, ezért ha egy változót a program további részeiben is szeretnénk használni, a függvény elején szükséges definiálnunk a global névtér használatával. A python annyiban egyszerűsíti a dolgunkat változók terén, hogy képes átképezni a hibásan implementált változótípust. Például az int, float és long változótípusok között támogatja a konverziót. A print szó már ismerős lehet számunkra. A pythonban ennek a szónak a használata mellett van lehetőségünk pl változók kiíratására, amiket a többi nyelvhez hasonlóan vesszővel kell elválasztanunk, ha többet adunk meg.

Természetesen a ciklusok, feltételes utasítások a pythonban is rendelkezésünkre állnak. A forráskódban címkeket is helyezhetünk el, ezeket a label szó használatával tudjuk megvalósítani, és a goto utasítással akárhonnan a megadott labelhez ugrunk.

A többi nyelvhez hasonlóan a kivételkezelés is megjelenik, ez a try-catch blokkhoz hasonlóan try-except blokkban történik, működése rokonaihoz hasonló. A try blokkban történik a végrehajtás, amíg semmi hiba nem ugrik fel, amint hiba történik, az except blokk utasításai hajtódnak végre.

A függvények definiálása itt a def szóval történik. A függvényeknek ugyanúgy itt is paramétereit lesznek, melyek érték szerint adódnak át, viszont vannak kivételek is. Ilyen kivételre példa a mutable típus. A python nyelv is támogatja az osztályok használatát, ezáltal az osztályok példányosításával keletkezett objektumok használatát is. Itt is értelemszerűen megjelenik az öröklődés is. Az osztályoknak itt is implementálhatunk függvényeket, itt viszont az első paraméter „self” kell, hogy legyen. A konstruktorok itt az `__init__` metódus használatával jelenhetnek meg.

18.2. Java 2 Útikalauz Programozóknak 5.0

A Java egy teljesen objektumorientált programozási nyelv. Ez annyit tesz, hogy a Java programok objektumkból és osztályokból állnak.

Az osztály két összetevőből áll: mezőkből és metódusokból, utóbbit más nyelvekben függvényeknek hívunk. A mezőben adatokat tárolhatunk, a metódusok pedig ezeken az adatokon végeznek valamiféle általunk megadott műveletet. A Java fordítóprogram bájtkód formátumra fordítja le a forráskódot, amely biztonsági szempontból előnyös, viszont hátrányos a sebességre nézve.

A forrásszöveget olvasva könnyen észrevehető, hogy a Java nyelv a C és a C++ nyelvből fejlődött ki. Sok szerkezetben hasonlít is az utóbbi két nyelvhez, viszont alapvető pontokban el is tér tőlük.

Változók

A változók a Java nyelvben „nem osztály”, azaz egyszerű típusok, ezeket egyszerű adatok ábrázolására használjuk.

- boolean: logikai true (igaz) illetve false (hamis) értékkel
- char: 16 bites Unicode karakter
- byte: 8 bites előjeles egész szám
- short: 16 bites előjeles egész szám
- int: 32 bites előjeles egész szám
- long: 64 bites előjeles egész szám
- float: 32 bites lebegőpontos racionális szám
- double: 64 bites lebegőpontos racionális szám

A változóhoz az „=” operátor használatával tudunk értéket rendelni. A fentebb felsorolt egyszerű típusok esetében valódi értékadás történik, összetett típusoknál viszont csak egy referencia átmásolást jelent. A változók értéke kezdeti értékadás előtt nincs definiálva, ezért ha még nem adtunk nekik értéket, nem szabad felhasználni, ugyanis a fordítóprogram ezesetben hibát fog jelezni.

Konstansok

A programban gyakran előfordul, hogy egy értéket több helyen is szeretnénk szerepeltetni, ezért egy kényelmes megoldás, ha ezt az értéket definiáltatjuk a programban csak 1 helyen, és később csak egy névvel hivatkozunk rá, viszont ezt az értéket már nem tudjuk megváltoztatni. Egy konstans megadása a final kulcsszóval történik.

Megjegyzések A megjegyzéseket a forráskód átláthatóbbá tétele miatt szoktuk alkalmazni. pl. ha forráskódunkat egy másik programozó kezébe adjuk, egyértelművé tehetjük számára például egy változó szerepét.

A Java nyelvben 3 módszer van megjegyzések írására.

- „//” használatával egysoros megjegyzést készíthetünk
- „/*” használatával több soros, hosszabb megjegyzést készíthetünk, a megjegyzés végén „*/” használatával jelezzük, hogy a megjegyzés véget ért
- „/**” használatával a dokumentációs megjegyzés válik lehetővé, ezt ugyanúgy kell lezárnunk, mint a több soros megjegyzést

Tömbök itt is megjelennek. A tömbökben objektumhivatkozások lesznek eltárolva. Ebben a nyelvben ha többdimenziós tömböt szeretnénk létrehozni, alternatív megoldást kell választanunk, ugyanis ebben a nyelvben többdimenziós tömb implementálására alapból nincs lehetőségünk. Java-ban az a felfogás ural-kodik, ha többdimenziós tömbökről van szó, hogy a tömbben létrehozhatunk tömböket, ezáltal lényegében többdimenziós tömböt kapunk.

A java egy objektumorientált programozási nyelv. Az objektumok az osztályok példányai. Osztályokat a Class kulcsszó használatával tudunk létrehozni. Az objektumokat a new előtaggal tudjuk példányosítani. Ha van például egy Emlős osztályunk, a következőképpen tudunk példányosítani Emlős objektumot: Emlős kutya = new Emlős ();. A javaban a new operátor foglal helyet az objektumnak a memóriában. Az osztálynak a tulajdonságai a memóriában a C/C++-szal ellentétben nem memóriacímre mutató referenciát, tartalmaznak, hanem az osztály adott tulajdonságára mutató referenciát. Az osztályok referenciaiák is. A Javaban a memóriaszivárgás elkerülése érdekében, ha egy objektumot nem használunk, azt alapból törölni fogja.

A kivételkezelés a C nyelvek és a python nyelvhez hasonlóan itt is megjelenik. Ez try-catch blokkban fog létrejönni. A try blokkban kell elhelyeznünk a forráskódot, ami amennyiben nem tartalmaz hibát, végrehajtódik, viszont ha a fordító hibát talál, a catch blokkban történi utasítások fognak végrehajtódni.

A java-ban az operátorok megegyeznek a C és C++-ban használt operátorokkal.

A típuskonverzió itt is megjelenik. A konverzió viszont történhet automatikusan illetve explicit módon. Automatikus konverzió akkor következhet be, ha például primitív típusokkal végünk valamilyen műveletet, és ekkor az eredmény típusa mindenkor a legnagyobb memóriaterüleettel bíró típus lesz a kiértékelés. Explicit konverzió segítségével tudjuk megadni, hogy pontosan milyen típussal kívánunk dolgozni. Szövegek esetén szövegkonverzióról beszélünk, ez akkor jelenik meg, ha egy nem String típusú változót Stringgé akarunk alakítani.

C++-ban az osztálytagokat a „::” operátorral tudtuk elérni. Javaban az adott tagokat „.” használatával tudjuk elérni. Az utasításoknak itt is 2 fajtája lesz. Vannak a kifejezések, ilyen például az értékadás, és a deklarációs-utasítások. A feltételes utasításnak két fajtája van, van a jól ismert egyszerű típusa, az if, és van az összetett típusa, mely switch-case-ekben merül ki. A már jól ismert ciklusok természetesen Javaban is megtalálhatók, ez a for, a while és a do-while ciklus.

IV. rész

Irodalomjegyzék

18.3. Általános

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

18.4. C

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

18.5. C++

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

18.6. 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 UDPORG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEAHCackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésekért és hasznos észrevételeikért.

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