

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Loós Tamás

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY	Bátfai, Norbert ÁCs Loós, Tamás	November 5, 2019

REVISION HISTORY

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

Ajánlás

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

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

DRAFT

Contents

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	11
2.6 Helló, Google!	12
2.7 100 éves a Brun téTEL	16
2.8 A Monty Hall probléma	17
3 Helló, Chomsky!	21
3.1 Decimálisból unárisba átváltó Turing gép	21
3.2 Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	21
3.3 Hivatkozási nyelv	23
3.4 Saját lexikális elemző	24
3.5 l33t.l	25
3.6 A források olvasása	28
3.7 Logikus	29
3.8 Deklaráció	29

4 Helló, Caesar!	31
4.1 double ** háromszög mátrix	31
4.2 C EXOR titkosító	33
4.3 Java EXOR titkosító	36
4.4 C EXOR törő	37
4.5 Neurális OR, AND és EXOR kapu	41
4.6 Hiba-visszaterjesztéses perceptron	46
5 Helló, Mandelbrot!	48
5.1 A Mandelbrot halmaz	48
5.2 A Mandelbrot halmaz a std::complex osztálytalál	51
5.3 Biomorfok	52
5.4 A Mandelbrot halmaz CUDA megvalósítása	54
5.5 Mandelbrot nagyító és utazó C++ nyelven	57
5.6 Mandelbrot nagyító és utazó Java nyelven	57
6 Helló, Welch!	58
6.1 Első osztályom	58
6.2 LZW	61
6.3 Fabejárás	63
6.4 Tag a gyökér	64
6.5 Mutató a gyökér	67
6.6 Mozgató szemantika	68
7 Helló, Conway!	70
7.1 Hangyszimulációk	70
7.2 Java életjáték	74
7.3 Qt C++ életjáték	78
7.4 BrainB Benchmark	79
8 Helló, Schwarzenegger!	80
8.1 Ssoftmax Py MNIST	80
8.2 Ssoftmax R MNIST	82
8.3 Mély MNIST	83
8.4 Deep dream	83
8.5 Robotpszichológia	83

9 Helló, Chaitin!	84
9.1 Iteratív és rekurzív faktoriális Lisp-ben	84
9.2 Gimp Scheme Script-fu: króm effekt	87
9.3 Gimp Scheme Script-fu: név mandala	91
10 Gutenberg	99
10.1 Programozási alapfogalmak	99
10.2 Programozás bevezetés	109
10.3 Programozás	111
III Második felvonás	112
11 Helló, Arroway!	114
11.1 OO szemlélet	114
11.2 Homokozó	115
11.3 Gagyi	119
11.4 Yoda	121
11.5 Kódolás from scratch	122
12 Helló, Liskov!	123
12.1 Liskov helyettesítés sértése	123
12.2 Szülő-gyerek	124
12.3 Anti OO	124
12.4 Hello, Android! Tutorom: Racs Tamás	126
12.5 Ciklomatikus komplexitás	130
13 Helló, Mandelbrot!	132
13.1 Reverse engineering UML osztálydiagram	132
13.2 Forward engineering UML osztálydiagram	134
13.3 Egy esettan	138
13.4 BPMN	138
13.5 TeX UML Tutorom: Rácz András István	140

14 Helló, Chomsky!	144
14.1 Encoding	144
14.2 I334d1c4 5	145
14.3	146
14.4 Full screen	146
14.5 Perceptron osztály	148
15 Helló, Stroustrup!	150
15.1 JDK osztályok & Összefoglaló	150
15.2 Másoló-mozgató szemantika	151
15.3 Változó argumentumszámú ctor	152
16 Helló, Gödel!	154
16.1 Gengszterek	154
16.2 STL map érték szerinti rendezése	155
16.3 GIMP Scheme hack	156
16.4 Alternatív Tabella rendezése	166
17 Helló, !	170
17.1	170
17.2	170
17.3	170
17.4	170
17.5	170
18 Helló, Schwarzenegger!	171
18.1	171
18.2	171
18.3	171
18.4	171
18.5	171
19 Helló, Calvin!	172
19.1	172
19.2	172
19.3	172
19.4	172
19.5	172

20 Helló, Berners-Lee!	173
20.1 1.hét - Python	173
20.2 1.hét - A Java nyelvről C++ programozóknak-1.	174
20.3 2.hét- Öröklődés, osztályhierarchy. Polimorfizmus, metódustúlterhelés. Hatáskörkezelés. A bezárási eszközrendszer, láthatósági szintek. Absztrakt osztályok és interfészek.	175
 IV Irodalomjegyzék	 179
20.4 Általános	180
20.5 C	180
20.6 C++	180
20.7 Lisp	180

List of Figures

4.1	Szemléltetés	31
4.2	EXOR szemléltetése	34
4.3	EXOR titkosítás visszafejtése a kulcs birtokában	36
4.4	Neuralnet-OR	42
4.5	Neuralnet-OR-AND	43
4.6	Neuralnet-EXOR	46
5.1	Mandelbrot halmaz	51
5.2	Biomorph	53
6.1	SUN módszere	61
6.2	Faépítés szemléltetése	62
7.1	Hangyaszimulációk	74
7.2	Hangyaszimulációk	74
7.3	Életjáték	78
9.1	Script-fu konzol	85
11.1	nextGaussian() a jdk-ban	115
11.2	tomcat servlet indítás	118
11.3	az eredmény	118
11.4	Formula	122
12.1	RGB beállítás	127
12.2	Blue	128
12.3	Green	129
12.4	checkstyle	131
13.1	Forrás kiválasztása instant reversehez	133

13.2 Binfa osztálydiagramma	134
13.3 Egy beépített telefon kommunikációs template	135
13.4 Instant Generate használata	136
13.5 A megadott output mappába generált kódok	137
13.6 Pizzarendelés BPMN	139
13.7 Generált kód	140
13.8 OOCWC	143
14.1 Fordítás Cp1250-el	145
14.2 Leet	146
14.3 Teljes képernyőn	147
14.4 Perceptron	149
16.1 Lambda expression- from geeksforgeeks.org	154

DRAFT

List of Tables

12.1 Összehasonlítás	126
----------------------------	-----

DRAFT

Előszó

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

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz alkálni 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 feladatesokrot. minden feladathoz megadja a megoldás forráskódját és forrásokat feldolgozó videókat. Az olvasó feladata, hogy ezek tanulmányozása után maga adja meg a feladat megoldásának lényegi magyarázatát, avagy írja meg a könyvet.

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

Milyen nyelven nyomjuk?

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

Part I

Bevezetés

DRAFT

Chapter 1

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.

Part II

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

Chapter 2

Helló, Turing!

2.1 Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: [100egy.c](#)

A feladatnál három kódra van szükségünk. Első kódunk egy for végtelen ciklust használ, ami folyamatosan fut amíg ki nem lépünk a programból, így 100%-ban dolgoztat meg egy magot. Ezt például a terminálban beírva a htop parancsot, és ott a programot megkeresve tekinthetjük meg.

```
int
main ()
{
    for (;;) ;

    return 0;
}
```

A második kód esetében a sleep (n) utasítást használjuk a végtelen ciklusunkban. (A sleepről a manualban bővebben is olvashatunk.) Ezzel a programunkat mindenkor várakoztatásra sarkalljuk, így lelassítva azt, és nem terheljük annyira a magot.

[100nulla.c](#)

```
main ()
{
    for (;;)
        sleep (1);

    return 0;
}
```

A harmadik kód : [100osszes.c](#)

Ahhoz hogy az összes magot megdolgoztassuk, a 100egy nevű program mintájára írunk egy másik programot, annyi különbséggel, hogy az OpenMp-t használjuk, amivel ki tudjuk használni a számítógép összes magját. Az OpenMP egy compiler kiegészítő, futtatáskor a szokásos gcc után –fopenmp-t írva fordítunk, a forráskódban a mainben #pragma omp parallel-t használjuk.

```
#include <omp.h>

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

2.2 Lefagyott, nem fagyott, akkor most mi van?

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

Megoldás videó:

Megoldás forrása:https://gitlab.com/nbatfai/bhax/tree/master/thematic_tutorials/bhax_textbook 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...

Megnézzük a kódot, és a Lefagy-ban szereplő if argumentumában szereplő kódra vetünk egy pillantást. Ezt, hogy "P-ben van végtelen ciklus", nem tudjuk előre megjósolni, nem tudjuk kód formára hozni.

Ha létezne egy ilyen, működő T1000-es program, és ennek segítségével saját magáról szeretnénk megállapítani, hogy le fog-e fagyni, akkor ellentmondásra jutnánk, tehát ilyen program nem létezik, nem működne.

2.3 Változók értékének felcserélése

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

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

Megoldás forrása:

Egy egyszerű programról van szó,többféle megoldás létezik,kis gondolkodás és a megoldás videó megtekintése után az alábbi matekos megoldást választottam. Két változóm van,a és b,értékük 1 és 2. Ezt ki is íratom a programmal,majd az értéküket a csere után is,hogy látszódjon jól csináltam-e a dolgokat. A csere három lépéssben zajlott le. Első lépéssben a értékéből kivontuk b értékét.(A változók új értékét kikommenteltem,hogy látszódjon mi történik.) Második lépéssben b-hez hozzáadjuk a(új)értékét.Végezetül pedig a új értéke b és a különbsége lesz,így a két változónk értéke megcsere elődött.

A programhoz a courses.h függvénykönyvtár szükséges, ebben szerepelnek a refresh(), clear(), refresh(), mvprintw() függvények. Fordításkor a megszokott gcc módszer után a -lncurses-t kell kapcsolni.

Változócsere

```
int main() {  
  
    int a = 1;  
    int b = 2;  
  
    printf("A változók csere előtt: \n");  
    printf("a = ");  
    printf("%d\n",a);  
    printf("b = ");  
    printf("%d\n",b);  
  
    a = a-b; // a most -1  
    b = b+a; // b most 1  
    a = b-a; // a most 2  
  
    printf("A változók csere után: \n");  
    printf("a = ");  
    printf("%d\n",a);  
    printf("b = ");  
    printf("%d\n",b);  
}
```

A program output:

A változók csere előtt:

```
a = 1  
b = 2  
A változók csere után:  
a = 2  
b = 1
```

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 videókon.)

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

Megoldás az if-es programoz: [pattog_if.c](#)

A megoldás nagyon egyszerű. Először is, szükség van egy területre, amin belül a labdánk pattoghat, majd teknikai okokból egy kezdő értékre, ahonnan elindul a labda. A terület sorokból és oszlopokból áll.

```
int x = 0; // oszlop kezdopont  
int y = 0; // sor kezdopont  
  
int deltax = 1; // hanyasaval lepkedjen  
int deltay = 1;  
  
int mx; // oszlopok  
int my; // sorok
```

Maga a labdapattogtatás egy végtelen for ciklussal történik, amiben a következők mennek végbe:

```
for ( ; ; ) {  
  
    getmaxyx ( ablak, my , mx );  
  
    mvprintw ( y, x, "O" );  
  
    refresh ();  
    usleep ( 100000 );  
  
    clear();  
  
    x = x + deltax;  
    y = y + deltay;  
  
    if ( x>=mx-1) { // elerte-e a jobb oldalt?  
        deltax = deltax * -1;
```

```
        }
        if ( x<=0 ) { // elerte-e a bal oldalt?
            deltax = deltax * -1;
        }
        if ( y<=0 ) { // elerte-e a tetejet?
            deltay = deltay * -1;
        }
        if ( y>=my ) { // elerte-e a aljat?
            deltay = deltay * -1;
        }
    }

}
```

Az mvprintw a "kurzorunk" helyére ír egy O-betűt, amit az x és y koordináták határoznak meg. A refresh() és clear() ahoz kell, hogy kirajzolhassuk majd letörölhessük a képernyőt minden iterációnál. A labda kirajzolódik a nullákkal inicializált kezdőhelyeken, majd x és y értékét növeljük a növekedés értékével (ami 1). Ezután négy if fügvénnyel vizsgáljuk a labdánk pozícióját. Ha elérte területünk akármeilyik szélét, akkor az annak megfelelő sor vagy oszlop koordinátát (-1)-el szorozzuk meg, így negatív irányba fog "mozogni" a labda, nem fog a területen kívülre tovább haladni.

Megoldás forrása: Az udprog-os mappában a pattog.c programot értelmeztem a feltételes, if nélküli megoldáshoz. [pattog.c](#)

Van két fügvényünk, a bitzero, és a rajzol. A bitzero függvény egy karaktert kap argumentumául, aminek a bit értékével fog dolgozni, iterálni. A rajzol két karaktert kér, egy szélességet és egy magasságot.

```
char bitzero(char x) {
    int i;
    char bitt = x&0x1;
    for (i=0; i<8; i++) {
        bitt |= (x>>i)&1;
    }
    return 1-bitt;
}

void rajzol(char width, char height) {
    int i;
    /*magasság*/
    for (i=1; i<=height; i++) {
        printf("\n");
    }
    /*szélesség*/
    for (i=1; i<=width; i++) {
        printf(" ");
    }
    printf("O\n");
}
```

A rajzol függvényben egy-egy for ciklussal haladunk végig a sorokon és oszlopokon. A magasságot vizsgálva haladunk a sorokon, úgy hogy ameddig el nem éri a ciklus a magasságot, addig minden iterációban egy \n-el sortörést iktat be. Az oszlopokon ugyanígy, a szélességet vizsgálva ír " " -el jelölt szóközöket. Amikor befejeződik a két for ciklus ,akkor a printf ("O\n") használva írunk ki egy 0-t, ami a labdánkat jelöli.

A main függvényben egy while ciklust használunk. A ciklus kezdésként egy rendszer hívást hajt végre, ami megtisztítja a terminált, így a programunkat fogjuk látni rajta. Ahhoz, hogy tudjuk, hogy hova is kell kiíratni a labdát, meg kell határoznunk a kezdőértékeket, kezdő koordinátákat, hogy azokhoz igazodhassunk. Ezen lennie kell egy keretnek, amin belül pattoghat a labda, tehát egy maximális magasságának és szélességnek. Ezek az x, y, vx, és vy. A következő műveleteket hajtjuk végre:

```
int main() {
    char x=1, y=1, vx=1, vy=1;
    while(1) {
        system("clear");
        vx-=2*bitzero(79-x);      //balra pattanjon
        vx+=2*bitzero(x);         //
        vy-=2*bitzero(24-y);      //lefelé
        vy+=2*bitzero(y);         //
        x+=vx;
        y+=vy;
        //printf("X: %d Y: %d \n", x, y);           //Koordináták
        //printf("Vx: %d Vy: %d", vx, vy);           //Velocity
        rajzol(x,y);
        usleep(100000);
    }
    return 0;
}
```

A kirajzolás után új értéket kap az x és az y. Meghívjuk az előbb elmagyarázott rajzol-t, majd az usleep () et, amivel várakoztatjuk a program bezárását.

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. Használд ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás videó:

Megoldás forrása:

A Linus Torvalds féle BogoMips: [bogomips.c](#)

A feladatban szereplő while ciklus feje a következő:

```
while ((loops_per_sec <= 1))
```

Ez a program által megkapott loops_per_sec-et bitshifteli. A bitshiftelés azt jelenti, hogy az értéket amit használunk bitenként nézzük, és (jelenleg balra) toljuk el bitenként. A feladatunk most megmondani, hogy mennyi egy int típusú változó mértéke. Tehát a while () -nak o loops_per_sec helyett egy másik, int típusú változót adunk, és megszámoljuk hányszor történik bit shiftelés, ezt pedig egy másik változóban könnyedén számon tarthatjuk.

Megoldás : [szohossz.cpp](#)

2.6 Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Tutorom: Pócsi Máté

A feladat megértéséhez a https://www.youtube.com/watch?v=P8Kt6Abq_rM videót és az alábbi kódöt használtam.

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

void
kiir (double tomb[], int db){

    int i;

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

double
tavolsag (double PR[], double PRv[], int n){

    int i;
    double osszeg=0;

    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

    return sqrt(osszeg);
}

void
pagerank(double T[4][4]){
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény
    double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0}; //ezzel szorzok
```

```
int i, j;

for(;;){

    // ide jön a mátrix művelet

    for (i=0; i<4; i++){
        PR[i]=0.0;
        for (j=0; j<4; j++) {
            PR[i] = PR[i] + T[i][j]*PRv[j];
        }
    }

    if (tavolsag(PR,PRv,4) < 0.0000000001)
        break;

    // ide meg az átpakolás PR-ből PRv-be

    for (i=0;i<4; i++) {
        PRv[i]=PR[i];
    }
}

kiir (PR, 4);
}

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

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

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

    printf("\nAz eredeti mátrix értékeivel történő futás:\n");
}
```

```
    pagerank(L);  
  
    printf("\nAmikor az egyik oldal semmire sem mutat:\n");  
    pagerank(L1);  
  
    printf("\nAmikor az egyik oldal csak magára mutat:\n");  
    pagerank(L2);  
  
    printf("\n");  
  
    return 0;  
}
```

A PageRank,-mint ahogy a feladatunk címe is mutatja,a Google találmánya,a feladata hogy meghatározza a weboldalak "értékét",és azt, hogy keresés közben milyen valószínűséggel találhatják meg az adott weboldalt az emberek.

A Pagerank értékét egy iteratív formulával határozzuk meg. Egy oldal pagerankjának a következő iterációban lévő értéke az adott oldal előző iterációban lévő pagerankjainak és a kimenő linkek hányadosának összege. Egyszerűbben szólva azoknak a többi oldalaknak a Pagerank értékét kell figyelembe venni,amelyek a meghatározni kívánt oldalunkra mutatnak, ,és ezeket elosztani az oldalak kimenő linkjeinek számával.

A legcélravezetőbb az lesz,ha kezdésnek megbeszéljük mi is történik a pagerank függvényünkben.

```
void  
pagerank(double T[4][4]) {  
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény  
    double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0 }; //ezzel szorzok  
  
    int i, j;  
  
    for(;;) {  
  
        // ide jön a mátrix művelet  
  
        for (i=0; i<4; i++) {  
            PR[i]=0.0;  
            for (j=0; j<4; j++) {  
                PR[i] = PR[i] + T[i][j]*PRv[j];  
            }  
        }  
  
        if (tavolsag(PR, PRv, 4) < 0.0000000001)  
            break;  
  
        // ide meg az átpakolás PR-ből PRv-be  
  
        for (i=0;i<4; i++) {  
            PRv[i]=PR[i];  
        }  
    }  
}
```

```

    }
    kiir (PR, 4);
}

```

A 4 weboldalunkat egy 4x4 es tömbbe tudjuk szépen felírni, és aztán dolgozni vele, a pagerank() függvény ezt a tömböt kéri be.

A tömbünk a következőképpen néz ki :

```

{
{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 sorok elő és az oszlopok fölé rendre fel lehetne írni a lapok nevét, ami most legyen A,B,C és D. Ha például az A oszlopot nézzük, azt látjuk, hogy A melyik oldalakra mutat, mint látható itt a B oldalra. Ha például az A sort nézzük, azt látjuk, hogy az A oldalra melyik oldalak mutatnak, ami itt a C oldal.

A formulánkhöz szükség van egy iterációra, hogy a következő iteráció értékét kiszámolhassuk. Ehhez inicializálnunk kell weboldalakat, ehhez az 1 értékünket (ami a nagy egész, az összes) el kell osztanunk négy felé, így minden oldal kezdőértéke 1/4 lesz, ezeket az értékeket a PRv-be tesszük.

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

A mátrixszorzás (amire egy pillanatonbelül rátérünk) utáni új értékeket pedig a PR-be fogjuk tenni, amit kezdésből ki kell nulláznunk hogy a belekerülő új érték helyes legyen.

```
double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
```

A tömbökön mátrixszorzást kell végrehajtani az iterációs formulában leírt módon, ezt a for ciklussal írjuk le :

```

for(;;) {

    for (i=0; i<4; i++) {
        PR[i]=0.0;
        for (j=0; j<4; j++) {
            PR[i] = PR[i] + T[i][j]*PRv[j];
        }
    }
}

```

A for ciklus végigfut a sorokon és az oszlopokon. PR új értékét úgy kapjuk meg, hogy hozzáadjuk a kezdeti(0) értékéhez a mátrixszorzás elvégzésével kapott értéket.

A kódban szerepel két speciális eset is, amihez egy-egy másik tömböt írtunk fel, L1-et és L2-t. L1 esetében a D oldalunk semmire sem mutat, L2 esetében pedig az A oldal csak saját magára.

L1 esetben, amikor D nem mutat sehol, ekkor "elveszik" a pagerank, minden oldal értéke nagyon kicsi lesz a sokadik iteráció után.

L2 esetben amikor A csak magára mutat, akkor "összegyűjt" a pageranket, ami egyszer belekerült az csak önmagába kerül vissza, a sokadik iteráció után A értéke közel egy lesz, a többi oldalé pedig nagyon kevés.

2.7 100 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_stp.r

A Brun téTEL szerint az ikerprímek reciprokösszege egy véges értékhez konvergál. A prím számok olyan természetes számok, amelyeknek egyedüli osztói önmaguk és 1. Az ikerprím számok pedig olyan sorban egymást követő prím számok, amelyek különbségének abszolút értéke kettő. (azért abszolút mert a különbség lehet nagyobból-kisebb és kisebb-nagyobb is).

Ezt a téTEL fogjuk feldolgozni R nyelven, az RStudio segítségével. A forrás:

```
library(matlab)

stp <- function(x) {

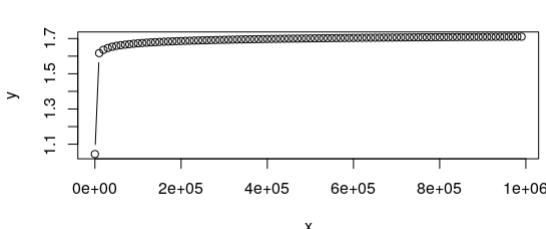
  primes = primes(x)
  diff = primes[2:length(primes)] - primes[1:length(primes)-1]
  idx = which(diff==2)
  t1primes = primes[idx]
  t2primes = primes[idx]+2
  rt1plust2 = 1/t1primes+1/t2primes
  return(sum(rt1plust2))
}

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

Készítünk egy függvényt stp néven. A függvény x megadott értékig fogja számolni az ikerprím számokat a következőképpen: Vektorokkal dolgozunk, ezekbe több elemet is tudunk egymás után pakolni. A primes vektorba kerülnek a prím számok. A diff ("difference", különbség) -be kerülnek a sorban egymás után kiszámított prímszámok különbsége. Nem mindegyik egymást követő prímre lesz igaz hogy 2 a különbségük, pl 7-11, ezért a which et használva kiválasztjuk, és az idx vektorba tesszük az elemeket, amelyekre az lesz igaz, hogy diff==2, vagyis tényleg kettő a különbség. Amint megvan két ikerprím, ott

a kisebbik, sorban egyel előrébb lévő prímet a t1primes vektorba, a nagyobbikat a t2primes -ba tessük. A reciprokösszegükre vagyunk kiváncsiak, ezeket az rt1plust2-be tessük. Az stp függvény visszatérési értéke ez az rt1plust2 összeg lesz.

A plot-ot használva(amihez szükséges telepíteni a matlab library-t) kirajzoltatjuk az eredményt:



2.8 A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R
mh.r](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R mh.r)

Egy másik videó egy youtubertől, ami sokat segített nekem megérteni, és egyébként nagyon érdekes videókat is készítenek: 4:40 től <https://www.youtube.com/watch?v=kJzSzGbfc0k>

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall probléma a következő. Egy televíziós játékban a játékosok 3 ajtó közül kinyithatnak egyet. Az egyik ajtó mögött egy nagy nyeremény vár rájuk, a másik kettő mögött pedig mondjuk semmi. Az hogy mit nyernek, nem lényeges. Az érdekesség az, hogy miután a játékos rámutat egy ajtóra, a játékvezető nem nyitja ki az ajtót, hanem döntés elő állítja a személyt. A másik két ajtó közül kinyit egy üreset(2 között mindig lesz egy üres, és vagy egy másik üres,vagy a nyertes), és megkérdezi a játékos, szeretne-e változtatni a döntésén, és a másik feltáratlan ajtót választani. Na már most, az első gondolatmenetünk a következő lehet. A legelején a választásunk 1/3 eséllyel jó, 2/3 eséllyel rossz. Miután kinyit a játékvezető egy üres ajtót, az esélyünk nyerni és veszteni is 1/2. Számít akkor, hogy megváltoztatjuk-e a döntésünket? Miért?

A helyes válasz az, hogy minden meg kell választani a döntésünket. De miért? Nagyon röviden azért, mert a 3 ajtó két csoportra osztható. Az első az az ajtó, amit alapból választ a játékos, a másik az a két ajtó, amely közül az egyik üres, és azt feltárja a játékvezető. Ha megváltoztatjuk a döntésünket, azzal a második csoportot fogjuk választani. Így az esélyünk arra hagy jó válasszunk, 2/3 lesz 1/3 helyett, azzal a különbséggel, hogy kiderül, az egyik ajtó mögött nincs semmi. De az esélyünk 2/3, statisztikailag ez a jó megoldás!

A program:

```
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
```

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

for (i in 1:kiserletek_szama) {

  if(kiserlet[i]==jatekos[i]){

    mibol=setdiff(c(1,2,3), kiserlet[i])

  }else{

    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))


  }

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


}

nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

  holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvalt[sample(1:length(holvart),1)]


}

valtoztatesnyer = which(kiserlet==valtoztat)

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

A programmal egymillió kísérletet végzünk el. minden egyes kísérlethez és a kísérlethez tartozó játékosnak kisorsolunk egy értéket véletlenszerűen 1-3 között. A műsorvezető a kísérletek számának aktualitását fogja képviselni.

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

Ezután egy forciklust használva végigmegyünk a kísérleteken, azokan az esetekben, ha egy iterációnál megegyezik a kísérlet és a játékos eredménye, akkor a játékos elsőre helyes ajtót választott, nyert. A kimaradt két ajtó, vagyis a nem nyertes ajtók számai bekerülnek a mibol-be. Azokban az esetekben viszont, amikor a játékos nem találja el elsőre a nyertes ajtót, a mibol-be már csak egy erték kerülhet, mert egy nem nyertes ajto már kiesett, és maradt még egy nem nyertes, és egy nyertes.

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

Ezeket nyomon követni azért szükséges, mert a játékvezető csak olyan ajtót nyithat ki a 2 extrában felajánlott közül, amelyik mögött nem rejlik semmi.

Ezután vizsgáljuk azokat az eseteket, amikor a felajánlás után váltott a játékos, vagy nem, és hogy ezeknek mi lesz az eredménye.

```
nemvaltoztatesnyer= which(kiserlet==jatekos)  
valtoztat=vector(length = kiserletek_szama)  
  
for (i in 1:kiserletek_szama) {  
  
    holvart = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))  
    valtoztat[i] = holvart[sample(1:length(holvart),1)]  
  
}  
valtoztatesnyer = which(kiserlet==valtoztat)
```

Lefuttatjuk,

```
> sprintf("Kiserletek szama: %i", kiserletek_szama)
[1] "Kiserletek szama: 10000000"
> length(nemvaltoztatesnyer)
[1] 3331586
> length(valtoztatesnyer)
[1] 6668414
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.499607
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 10000000
```

és látjuk, hogy az esély arra, hogy a döntésünket meg nem változtatva nagyjából 33% esélyünk van nyerni, megváltoztatva viszont 66%.

DRAFT

Chapter 3

Helló, Chomsky!

3.1 Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

A decimális a tizes, az unáris pedig az egyes számrendszer neve. A 10-s számrendszer a köznapi életben megsokkott, mindenki által ismert számrendszer, a számok 0-9 -ig terjednek. Az egyes számrendszerben csak az 1-es szám létezik. A tizes számrendszer számait unárisan X-darab egyesével lehet felírni. A 12-es szám például 12 darab egyes: 11111 11111 11.

A decimálisból unárisba átváltó Turing gép a következő képpen működik. Van egy decimális számrendszer beli számunk, a turing gép a legutolsó számjegyre állítja a gép fejét, ezt olvassa legelőször. Ezt a számjegyet addig csökkenti egyesével, ameddig nulla nem lesz, és minden egyes 1-el való csökkentésért egy tárolóba egy darab 1-est ír. Ha az aktuális számjegyünk nullára csökkent, akkor a nulla helyet egy 9-est írunk a számjegy helyére, majd a fej balra, a következő számjegyre áll. Ezt a számot szintén 1-el csökkenti, majd a fej visszalép az előbb már kinullázott, majd 9-es értéket kapott számra, és azt csökkenti egyesével. Ezt a folyamatot ismétli meg amíg az összes számjegy ki nem nullázódik.

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

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

Megoldás videó:

Megoldás forrása: 30.-32. oldalak [Prog1_1.ppt](#)

A generatív grammatika megalapozója Noam Chomsky. Ennek a nyelvtannak a célja olyan nyelvi modellek készítése, amelyek segítségével végtelen számú mondatot alkothat.

A generatív nyelvek négy részből állnak:

- Nem terminális jelek

- Terminális jelek
- Kezdő jel
- Helyettesítési szabályok

A nem terminális jelek változók, ezeket a nem terminális jelekkel, vagyis a konstansokkal helyettesíthetők. Ezeket a helyettesítéseket szabályok határozzák meg, amiket saját magunk határozunk meg. Példa:

- Nem terminális jelek -> S, X, Y
- Terminális jelek -> a, b, c
- Kezdő jel -> S
- Helyettesítési szabályok -> S → abc, S → aXbc, Xb → bX, Xc → Ybcc, bY → Yb, aY -> aaX, aY → aa

Tehát van egy alap jelünk a nem terminális jelek közül, jelen esetben S, erre alkalmazzuk a megfelelő helyettesítési szabályt, aXbc-t kapunk. Ezután megnézzük mit kaptunk, és addig alkalmazzuk a helyettesítési szabályok valamelyikét, ameddig teljesen konstansokból álló eredményt nem kapunk. Mivel nincs olyan szabály, amiben egy nem terminális jelből egy terminális jelet kapunk (pl X → a), ezért a nyelv környezetfüggő.

A levezetés:

```
S (S → aXbc)
aXbc (Xb → bX)
abXc (Xc → Ybcc)
abYbcc (bY → Yb)
aYbcc (aY -> aaX)
aaXbcc (Xb → bX)
aabXbcc (Xb → bX)
aabXcc (Xc → Ybcc)
aabYbcc (bY → Yb)
aabYbbccc (bY → Yb)
aaYbbbccc (aY → aa)
aaabbccc
```

Egy másik példa:

- Nem terminális jelek -> A, B, C
- Terminális jelek -> a, b, c
- Kezdő jel -> A
- Helyettesítési szabályok -> A → aAB, A → aC, CB → bCc, cB → Bc, C → bc

Levezetés:

A ($A \rightarrow aAB$)
aAB ($A \rightarrow aAB$)
aaABB ($A \rightarrow aAB$)
aaaABBB ($A \rightarrow aC$)
aaaaACBBB ($CB \rightarrow bCc$)
aaaabC_cBB ($cB \rightarrow Bc$)
aaaabCB_cB ($cB \rightarrow Bc$)
aaaabCBBc ($CB \rightarrow bCc$)
aaaabbC_cBc ($cB \rightarrow Bc$)
aaaabbCBcc ($CB \rightarrow bCc$)
aaaabbbCccc ($C \rightarrow bc$)
aaaabbbbcccc

3.3 Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

A BNF, vagyis a Backus-Naur-forma a környezetfüggetlen nyelvek leírását meghatározó nyelv.

```
<utasítás> ::=  
<összetett_utasítás>  
<kifejezés>; (értékkedás pl, num=10)  
if(<kifejezés>) <utasítás>  
else if(<kifejezés>) <utasítás>  
else <utasítás>  
switch (<kifejezés>  
<egész_konstans_kifejezés : <utasítás>  
goto <azonosító>;  
<azonosító> : <utasítás>  
break; continue; return<kifejezés>;  
or(<kifejezés1><kifejezés2><kifejezés3>) <utasítás>  
while(<kifejezés>) <utasítás>  
do <utasítás> while<kifejezés>  
üres utasítás ;
```

3.4 Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetben 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!

Megoldás videó:

Megoldás forrása:

A feladat megoldásához lexet használunk. Olyan (1 végződésű) programot írunk, amelyet odaadunk a lexnek, és ez a programunkból egy C programot készít. A terminálban következőképpen hajtjuk ezt végre:

lex -o realnumber.c realnumber.l

Ezután a C programunkat a megszokott módon fordítjuk le, és még hozzá írunk egy "-lfl" -t.:

gcc realnumber.c -o realnumber -lfl

Miután lefuttattuk a programot, beírhatunk olyan szöveget amit szeretnénk, ebből a szövegből pedig a program kiírja a valós számokat.

```
yhennon@Yhennon-PC:~/Desktop/Masodik Felev/kódok$ ./realnumber
Csodálatos12312 N4p ma ez a t4nulásra,n3m igaz?!444
Csodálatos[realnum=12312 12312.000000] N[realnum=4 4.000000]p ma ez a
t[realnum=4 4.000000]nulásra,n[realnum=3 3.000000]m igaz?!
[realnum=444 444.000000]
```

Az egész lex program így néz ki :

```
%{
#include <stdio.h>
int realnumbers = 0;
%%
digit [0-9]
%%
{digit}*(\.{digit}+) ? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

A programnak 3 fő része van, amit a %% %% választ el. Első rész az első %% jelek előtti rész, második a két %% %% jelek közötti rész, harmadik rész pedig a második %% jelek utáni rész.

Az első rész tehát :

```
%{  
#include <stdio.h>  
int realnumbers = 0;  
%}  
digit [0-9]
```

A %{ %} közötti lévő sorok, tehát az include és a realnumber változó bekerül a majdani c forrásba. A digit [0-9] definícióval azt adjuk meg,hogy a nullától kilencig terjedő számokat vegyük figyelembe.

A második rész a %% %% közötti rész :

```
%%  
{digit}*(\.{digit}+)? {++realnumbers;  
    printf("[realnum=%s %f]", yytext, atof(yytext));}  
%%
```

Ebben a részben a fordítási szabályok szerepelnek. A {digit}* rész azt jelzi hogy a megadott 0-9 számok közül vesz figyelembe akármennyit. Az utáni következő (\.{digit}+)? pedig azt, hogy nem kötelezően, de lehet egy olyan része a számnak, ami a szám után egy pont és legalább egy szám, tehát például .5 . Fontos hogy a {digit}* rész akármennyi számból állhat, tehát nullából is, így ponttal is kezdődhet a szám.

Ha találunk a kritériumoknak megfelelő számot, növeljük a realnumber változónkat, kiíratjuk a talált számot stringként majd doubleként, az atof () függvényt használva,ami stringből doublet készít.

A harmadik rész:

```
int  
main ()  
{  
    yylex ();  
    printf("The number of real numbers is %d\n", realnumbers);  
    return 0;  
}
```

Itt a yylex () függvényel elindítom az elemzést. A végén pedig kiiratjuk a valós számok számát.

3.5 l33t.I

Lexelj össze egy l33t cipher!

Megoldás videó:

Megoldás forrása:

A megoldáshoz az előző feladatban megismert módon a lexert használjuk, és a leet nyelvet, amiről a <https://simple.wikipedia.org/wiki/Leet> weboldalon olvashatunk. A program a beírt szöveget át fogja alakítani leet nyelven.

Kezdésből felépítjük a cipher struktúrát, egy tömböt ami két részből áll az abc minden karakterére, és a 0-9 számokra. Az első rész maga a karakter, egy char, a második rész pedig karakterenként 4 lehetséges opcióniból egyet-egyet fogunk kiválasztani.

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

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

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

{'a', {"4", "4", "@", "/-\\"}}, {'b', {"b", "8", "|3", "|{}"}}, {'c', {"c", "(", "<", "{}"}}, {'d', {"d", "|)", "|]", "|{}"}}, {'e', {"3", "3", "3", "3"}}, {'f', {"f", "|=", "ph", "|#"}}, {'g', {"g", "6", "[", "[+{}"}}, {'h', {"h", "4", "-|", "[-]"}}, {'i', {"1", "1", "|", "!"}}, {'j', {"j", "7", "_|", "_/"}}}, {'k', {"k", "|<", "1<", "|{}"}}, {'l', {"l", "1", "|", "|_"}}, {'m', {"m", "44", "(V)", "|\\|/|"}}, {'n', {"n", "|\\|/", "/\\/", "/V"}}, {'o', {"o", "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"}}

};

%}
```

A következő (a %% közötti) részben megadjuk, hogy egy bármilyen karaktert keresünk, ezt egy ponttal jelöljük, és megadjuk, hogy mi a teendő ha találtunk egy ilyet.

```
%%
.

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);
}
%%
```

Nézzük mi is történik. A talált karaktert az adott karakterhez tartozó tömbsorként kezeljük, a karaktert a tolower() függvény kisbetűssé alakítja, ha alapból nem kisbetű volt.

Erzután egy 1-100-ig terjedő véletlen számot generálunk a karakterhez, és attól függően, hogy milyen értéket kaptunk, a karakterhez tartozó 4 lehetséges helyettesítést egyikét íratjuk ki. Ha 91-nél kisebb a szám, akkor az első lehetőséget választjuk. Ha 91-nél nagyobb, és 95-nél kisebb, akkor a másodikat. Ha 95-nél nagyobb, és 98-nál kisebb, akkor a harmadikat. minden más esetben pedig a negyediket. Ezt minden karakterre elvégzi a program.

A harmadik részben szerepel a main függvényünk, amiben az `srand()` függvénnnyel ténylegesen véletlen szám generátort állítunk be, a `yylex()` függvénnnyel pedig elindítjuk az elemzést.

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, jelkezelo);
```
- ii.

```
for(i=0; i<5; ++i)
```
- iii.

```
for(i=0; i<5; i++)
```
- iv.

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

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

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

```
printf("%d %d", f(a), a);
```
- viii.

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7 Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, https://youtu.be/AJSXOQFF_wk

Tanulságok, tapasztalatok, magyarázat...

3.8 Deklaráció

Vezess be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- `int a;`

a egy egész típusú változó;

- `int *b = &a;`

b az a címére mutató pointer;

- `int &r = a;`

r az a referenciajára,

- `int c[5];`

c egy 5db egészből álló tömb;

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

tr egy tömb értékeinek referenciajára;

- `int *d[5];`

d egy mutató tömb amiben 5 egészre mutató mutató van;

- `int *h();`

h egy egészre mutató mutatót visszaadó függvény;

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

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

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

Megoldás video:

Megoldás forrása: [deklaracio.cpp](#)

Tanulságok, tapasztalatok, magyarázat...

Chapter 4

Helló, Caesar!

4.1 double ** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Lefoglalunk egy double-kból álló háromszögmátrixnak helyet, és ennek a háromszögmátrixnak adunk értéket.

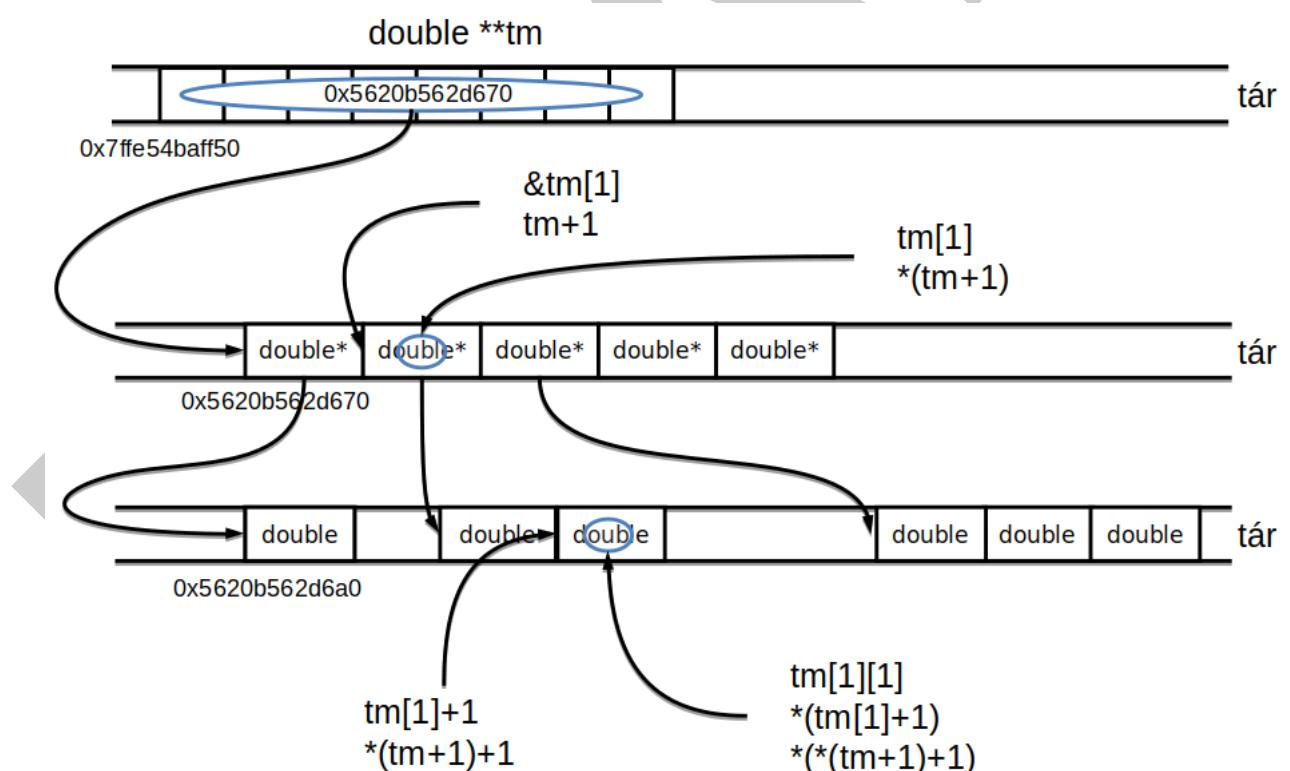


Figure 4.1: Szemléltetés

A double ** tm az egy double mutatóra mutató mutató.Lefoglalunk a memóriában 5 double mutatónak helyet a malloc() függvényel, és a tmet ennek a területnek az első elemére állítjuk.

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

A double tm által címzett öt elemű double mutató tömb öt elemét rendre beállítjuk 1,2...,5 doublenek foglalt helyre.

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

Az első for ciklussal értéket adunk a háromszögmátrixunk elemeinek,másodikkal pedig kiírjuk azt.

```
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");
}
```

Majd 4 különböző szintaktikával adunk értéket a háromszögmátrixunk negyedik sorában lévő elemeknek.Kiírjuk a háromszögmátrixot.

```
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");
}
```

Az öt elemű double * tömb által címzett memóriaterületeket rendre felszabadítjuk ,végül pedig magát a double* tömböt is a free() függvényel.

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

free (tm);
```

```
yhennnon@Yhennon-vm:~/Downloads/yhennnon/enyimé/bhax/thematic_tutorials/ ←
    bhax_textook/codes/caesar_csoport$ ./tm
0x7ffdb9c73650
0x55b034d22670
0x55b034d226a0
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,
```

Az első három sorral nyomon követjük a képen is látható memóriacímeteket, rendre felülről lefelé vannak, ahogy a képen is látszik.

4.2 C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

A feladat egy titkosítót írni EXOR-ral, ehhez először is ismerkedjünk meg azzal, hogy mi is az az EXOR. Egy logikai művelet, -amit a C programozási nyelv ^-ként ismer-, amelynek két értékre (itt: 0 vagy 1) van szüksége, és egy másik értéket ad vissza, attól függően, hogy miket kapott. A művelet 1-et ad vissza, ha a két kapott érték közül pontosan az egyik 1, különben 0-t.

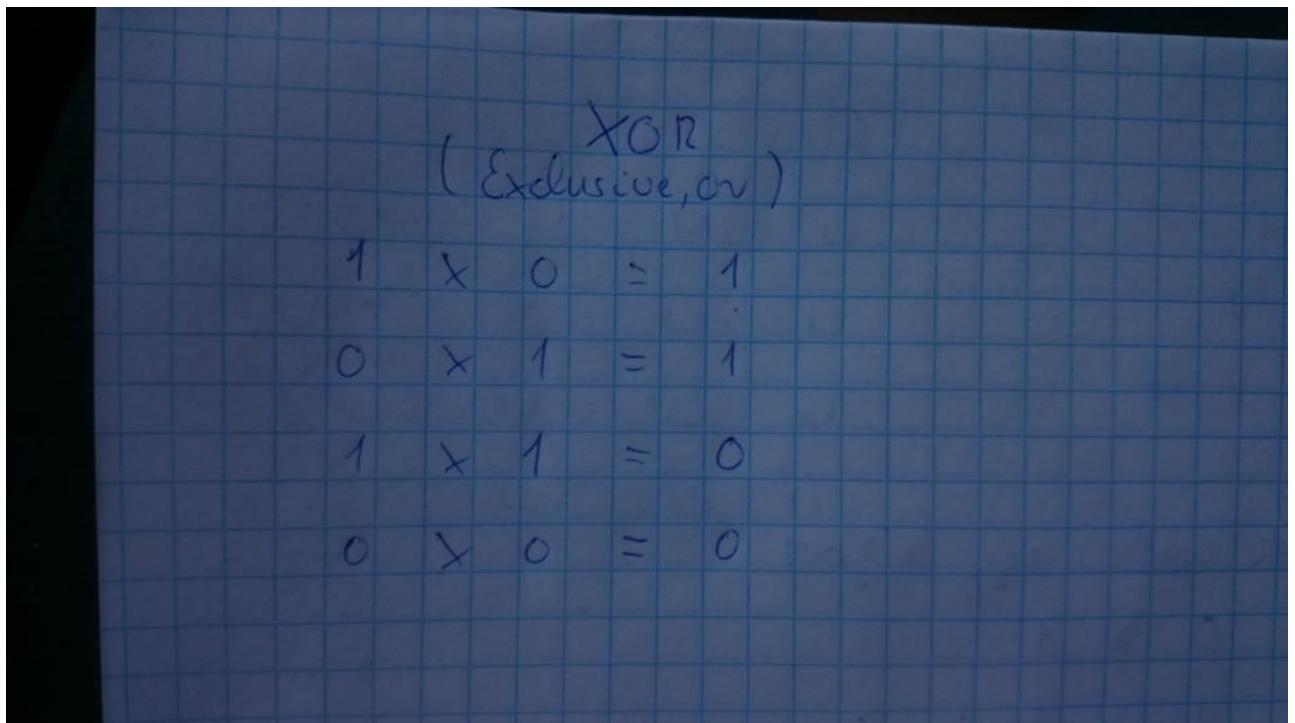


Figure 4.2: EXOR szemléltetése

Ezt a titkosítást úgy lehet elképzelni,hogy van egy szöveg,amit el szeretnénk juttatni valakinek,de nem akarjuk hogy bárki elolvashassa a szöveget,ezért titkosítjuk,méghozzá exoros módszerrel.Ehhez szükségünk van egy kulcsra. Így meglessz a két "értékünk" amit az előbb említettem,úgy,hogy karakterről karakterre,bitenként exorozni fogjuk a szöveget a kulccsal.

Lássuk a kódot,és értelmezzük a sorokat.:

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

    char kulcs[MAX_KULCS];
    char buffer[BUFFER_MERET];

    int kulcs_index = 0;
    int olvasott_bajtok = 0;

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

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

    write (1, buffer, olvasott_bajtok);
}

}
```

A felhasználó megad egy kulcsot,majd a szöveget amit titkosítani szeretne.A titkosított szöveget is tárolunk kell.

```
char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];
```

Ezeknek van egy maximális mérete amit definiáltunk a program elején.

Ahogy haladunk a szövegben,mindig tudnunk kell, hogy hol járunk,és megadunk egy értéket,ami követi majd,hogy a szövegben hány karaktert olvastunk már be.

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

A kulcs méretét beállítjuk:

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

És végül van egy ciklusunk,ami addig fog futni,amíg írunk neki szöveget,és itt megtörténik az exorozás,legvégül pedig kiírja a mostmár titkosított szöveget.

```
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);
}
```

Akihez eljut az üzenet, és a kulcs birtokában van, könnyen vissza tudja fejteni a titkosítást. Az eredményt exorozva a kulccsal megkapjuk az eredeti szöveget.

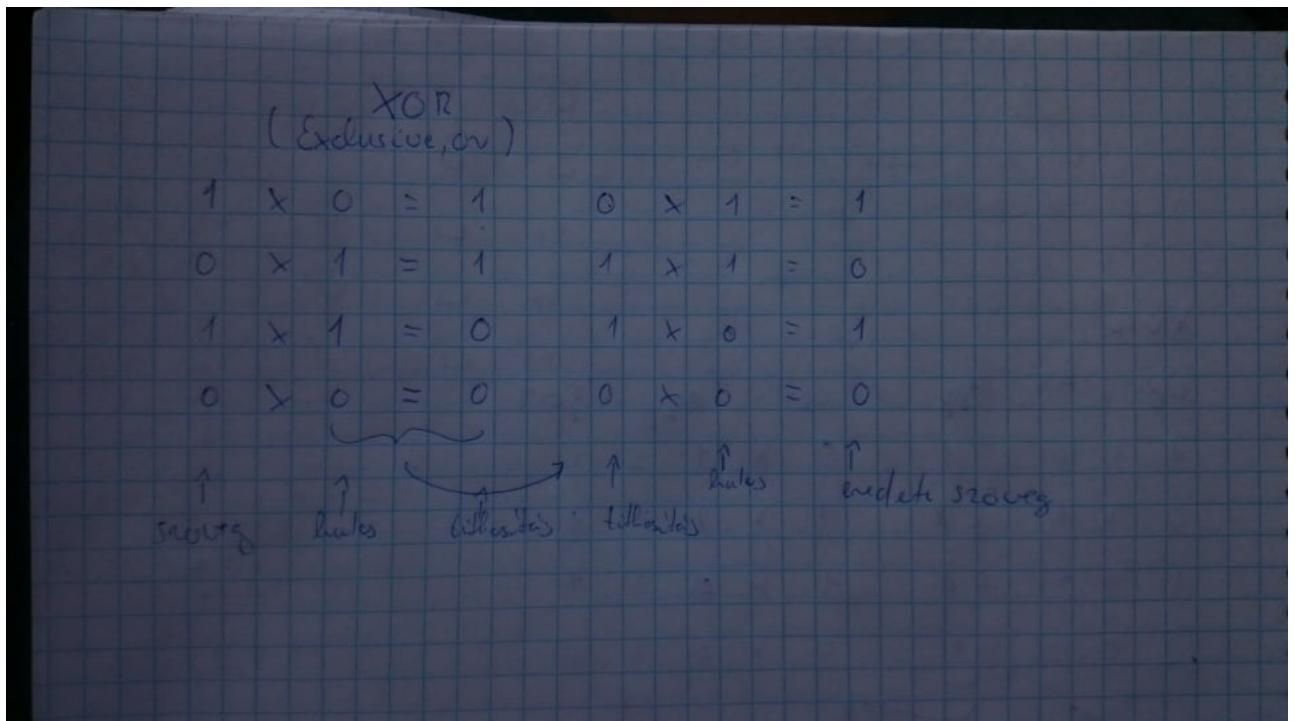


Figure 4.3: EXOR titkosítás visszafejtése a kulcs birtokában

4.3 Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
public class ExorTitkosító {  
  
    public ExorTitkosító(String kulcssSzöveg,  
                         java.io.InputStream bejövőCsatorna,  
                         java.io.OutputStream kimenőCsatorna)  
        throws java.io.IOException {  
  
        byte [] kulcs = kulcssSzöveg.getBytes();  
        byte [] buffer = new byte[256];  
        int kulcsIndex = 0;  
        int olvasottBájtok = 0;  
  
        while((olvasottBájtok =
```

```
bejövőCsatorna.read(buffer) ) != -1) {  
  
    for(int i=0; i<olvasottBájtok; ++i) {  
  
        buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);  
        kulcsIndex = (kulcsIndex+1) % kulcs.length;  
  
    }  
  
    kimenőCsatorna.write(buffer, 0, olvasottBájtok);  
  
}  
  
}  
  
public static void main(String[] args) {  
  
    try {  
  
        new ExorTitkosító(args[0], System.in, System.out);  
  
    } catch(java.io.IOException e) {  
  
        e.printStackTrace();  
  
    }  
  
}  
}
```

4.4 C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Az előző feladatban elmagyaráztuk hogyan lehet visszatörni a kódot a kulcs birtokában, de mi a helyzet, ha nem tudjuk a kulcsot? Ki kell találni!

```
#define MAX_TITKOS 4096  
#define OLVASAS_BUFFER 256  
#define KULCS_MERET 8  
#define _GNU_SOURCE  
  
#include <stdio.h>  
#include <unistd.h>
```

```
#include <string.h>

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

    int kulcs_index = 0;

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

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

    }

}

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

```
return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;

    // 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;

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

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

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

        // ujra EXOR-ozunk, igy nem kell egy masodik buffer
    }
}
```

```
    exor (kulcs, KULCS_MERET, titkos, p - titkos);  
}  
  
return 0;  
}
```

Kezdjük az átlagos_szohossz és a tiszta_lehet függvényekkel. A tiszta lehet függvény meghatározza az adott kulccsal visszafordított szövegből a keresőszavak és az átlag szóhossz alapján hogy az adott kulcs lehetséges megoldás-e. Az átlag szóhossz függvény pedig megnézi hogy az adott kulccsal viszafordított szövegen mennyi a szóhossz.

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

Beolvassunk a bufferból a karaktereket.

Miután beolvastam a karaktersorozatot,a titkos karaktertömbben a titkosított utáni plussz karaktereket nullázom.

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

Ezek a for ciklusok végezik magát a törést,végigmennek a kulcs összes lehetséges értékén,minden kulcs értékre lefuttatja az exor_tores függvényt, amely az exor függvény segítségével az adott kulccsal visszafejt a lehetséges eredeti szöveget. Erre a visszafejtett szövegre lefuttatja a tiszta_lehet függvényt, melynek segítségével megállapítja,hogy az adott kulcs lehetséges megoldás-e.

```
for (int ii = '0'; ii <= '9'; ++ii)  
    for (int ji = '0'; ji <= '9'; ++ji)  
        for (int ki = '0'; ki <= '9'; ++ki)  
            for (int li = '0'; li <= '9'; ++li)  
                for (int mi = '0'; mi <= '9'; ++mi)  
                    for (int ni = '0'; ni <= '9'; ++ni)  
                        for (int oi = '0'; oi <= '9'; ++oi)  
                            for (int pi = '0'; pi <= '9'; ++pi)  
                            {  
                                kulcs[0] = ii;  
                                kulcs[1] = ji;  
                                kulcs[2] = ki;  
                                kulcs[3] = li;  
                                kulcs[4] = mi;  
                                kulcs[5] = ni;  
                                kulcs[6] = oi;  
                                kulcs[7] = pi;  
  
                                if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))  
                                    printf
```

```

("Kulcs: [%c%c%c%c%c%c] \nTiszta szoveg: [%s] \n",
ii, ji, ki, li, mi, ni, oi, pi, titkos);

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

```

4.5 Neurális OR, AND és EXOR kapu

R

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

Megoldás forrása: [nn.r](#)

Tanulságok, tapasztalatok, magyarázat...

Egy gépnek akarjuk megtanítani az OR, AND, és EXOR műveleteket. Ábra:

R szimulációkkal az emberi agy működésének mintájára készítjük el ezeket a műveleteket. A testünk idegsejtjei a következőképpen működnek. A neuron az idegrendszer legkisebb egysége, jeleket továbbítanak. A neuronokat ingerek érik, amiknek ha az erőssége, a "súlya" elér egy bizonyos küszöböt, akkor a neuron "tüzelni" fog, és továbbítja a súly értékét, mint jelet, a hozzáartozó egységhez.

```

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

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

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

plot(nn.or)

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

```

Az a1 és a2 vektorokban tárolunk 0 vagy 1 értékeket, amik logikai HAMIS és IGAZ-nak felelnek meg, úgy hogy minden lehetőséget lefedjünk. Az OR-ba - vagyis a logikai VAGY-ba -,az a1 és a2 fejben, értékenként összpárosított és VAGY logikai műveletet elvégzett értékeit írjuk.

A data.frame-nek paraméterül adjuk az a1, a2 és OR-t, és or.data néven mentjük őket. A data.frame funkció táblázatba rendezi őket. Ezt a táblázatot lehet használni a következő lépésnél.

```

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

```

A neuralnet() funkcióval megalkotjuk a neurális hálót, itt meg kell adni hogy milyen műveletet(OR) akarunk elvégezni milyen értékekkel(a1+a2), hogy hol találhatóak a szükséges adatok,(ord.ata), a rejtegett rétegek számát(hidden, ami jelen esetben 0), a maximális lépések számát(stepmax), és a hibaküszöböt(threshold).

A rejtett rétegek arra szolgálnak, hogy extra információt felhasználva történjenek a számolások, az információ alapján újraellenőrizve, igazítva.

Az OR-t tanítva a következőt kapjuk:

```
> compute(nn.or, or.data[,1:2])
$neurons
$neurons[[1]]
  a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1

$net.result
[,1]
[1,] 0.001334083
[2,] 0.999678030
[3,] 0.999037789
[4,] 1.000000000
```

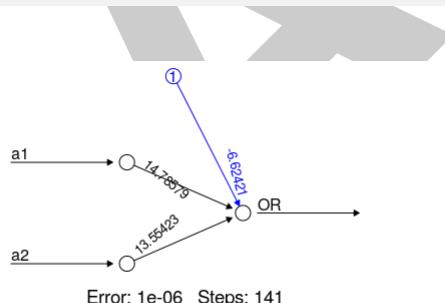


Figure 4.4: Neuralnet-OR

Egyszerre lehet tanítani két dolgot és, most az előző mintájára legyen a logikai VAGY és ÉS :

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

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

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

plot(nn.orand)
```

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

Az eredmény:

```
> compute(nn.operand, operand.data[,1:2])
$neurons
$neurons[[1]]
  a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1

$net.result
 [,1]      [,2]
[1,] 4.070802e-05 1.420913e-09
[2,] 9.999739e-01 9.903470e-04
[3,] 9.999741e-01 1.051658e-03
[4,] 1.000000e+00 9.986403e-01
```

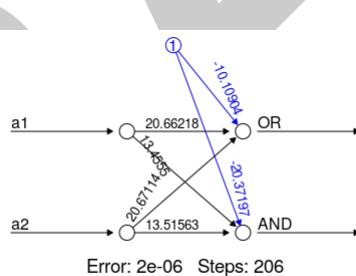


Figure 4.5: Neuralnet-OR-AND

Következő művelet az EXOR. Ahhoz viszont, hogy helyes eredményt kapjunk, az EXOR esetében szükségünk lesz rejtett rétegekre.

```
hidden=c(6, 4, 6)
```

Ha hidden layerek nélkül nézzük a kapott eredményt, látjuk, hogy helytelen, 0,5 közelí értékeket kapunk :

```
> 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])
$neurons
$neurons[[1]]
  a1 a2
[1,] 1  0
[2,] 1  1
[3,] 1  0
[4,] 1  1

$net.result
  [,1]
[1,] 0.5000015
[2,] 0.4999985
[3,] 0.5000015
[4,] 0.4999985
```

Rejtett rétegekkel viszont:

```
> a1      <- c(0,1,0,1)
> a2      <- c(0,0,1,1)
> EXOR    <- c(0,1,1,0)
>
> exor.data <- data.frame(a1, a2, EXOR)
>
> nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ←
  output=FALSE, stepmax = 1e+07, threshold = 0.000001)
>
> plot(nn.exor)
>
> compute(nn.exor, exor.data[,1:2])
$neurons
$neurons[[1]]
  a1 a2
[1,] 1  0
[2,] 1  1
[3,] 1  0
[4,] 1  1
```

```
$neurons[[2]]  
[,1] [,2] [,3] [,4] [,5] [,6] ←  
[1,] 1 0.152926928 0.8837576 0.0807014880 0.0508627071 0.8612823 ←  
0.0518831089  
[2,] 1 0.935261516 0.9984545 0.9262889411 0.0001069123 0.1634753 ←  
0.9387018252  
[3,] 1 0.001091262 0.1382388 0.0006514734 0.9477504873 0.9968738 ←  
0.0001688462  
[4,] 1 0.080392115 0.9316545 0.0853535941 0.0349281047 0.9093926 ←  
0.0451259789  
  
$neurons[[3]]  
[,1] [,2] [,3] [,4] [,5]  
[1,] 1 0.9771435913 1.088089e-02 0.9705293095 1.736090e-03  
[2,] 1 0.0003725196 1.740242e-08 0.0004301389 5.346070e-10  
[3,] 1 0.5118886333 8.733900e-01 0.2741813249 9.774301e-01  
[4,] 1 0.9852046827 1.316129e-02 0.9843106259 2.005140e-03  
  
$neurons[[4]]  
[,1] [,2] [,3] [,4] [,5] [,6] ←  
[1,] 1 0.99183770 0.988007865 0.004624684 0.996310471 0.992437750 ←  
0.002461752  
[2,] 1 0.18334443 0.031978637 0.899293762 0.040031238 0.406654512 ←  
0.917211353  
[3,] 1 0.01232873 0.007340944 0.998889164 0.001026309 0.001852492 ←  
0.999485391  
[4,] 1 0.99228517 0.988937738 0.004306596 0.996633703 0.992818887 ←  
0.002269655  
  
$net.result  
[,1]  
[1,] 0.0005827560  
[2,] 0.9989956314  
[3,] 0.9997439041  
[4,] 0.0005786685
```

Itt már helyes értékeket, a megadott párok esetében a második és harmadik esetben kapunk igaz eredményt, az EXOR művelet akkor ad vissza igaz eredményt(1), ha egymástól eltérő bemenetet kap(0-1,1-0).

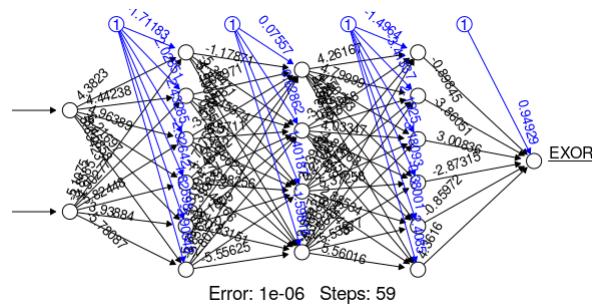


Figure 4.6: Neuralnet-EXOR

4.6 Hiba-visszaterjesztéses perceptron

C++

Megoldás video: <https://www.youtube.com/watch?v=XpBnR31BRJY>

Megoldás forrása: [main.cpp](#) [mandel.cpp](#) [mlp.hpp](#)

Tanulságok, tapasztalatok, magyarázat...

Az előző feladataban elmagyarázott neuralnet mintájára építve dolgozunk, bemenetnek most nem konkrétan számpárokat adunk meg, hanem egy bizonyos mandelbrot halmaz(következő fejezetben részletesen fel lesz dolgozva) képének RGB kódját adjuk meg bemenetnek. Ez a bemenetnek beadás, a feltöltés a main.cpp feladata, így néz ki:

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

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

    Perceptron* p = new Perceptron(3, size, 256, 1);

    double* image = new double[size];

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

    double value = (*p) (image);

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

    delete p;
```

```
delete [] image;
```

```
}
```

Az mlp.hpp tartalmazza (az eredeti, teljes mlp.hpp nem csak a most szükséges perceptron osztállyal : [nahshon](#)) a Perceptron osztályt. A mandel.cpp-t lefuttatva kapunk egy képet, ezt fogjuk inputnak adni a neuralnetnek, aminek a megvalósítása most a perceptronnal fog működni.

```
yhennnon@Yhennon-vm:~/Downloads$ ./mandel mandel.png
```

Lefuttatjuk, és egy értéket kell visszakapnunk.: 

```
yhennnon@Yhennon-vm:~/Downloads$ g++ mlp.hpp main.cpp -o perceptron -lpng -std=c++11
yhennnon@Yhennon-vm:~/Downloads$ ./perceptron mandel.png
0.556892
yhennnon@Yhennon-vm:~/Downloads$
```

DRY

Chapter 5

Helló, Mandelbrot!

5.1 A Mandelbrot halmaz

Megoldás video:

Megoldás forrása: [mandel.c](#)

A Mandelbort halmaz a komplex számok síkján alkotott halmaz, amit úgy tudunk vizualizálni, hogy egy kép minden pixelét a hozzáartozó komplex számnak feleltetjük meg(c), majd megnézzük, hogy a komplex számból alkotott sorozat, amelyet úgy kapunk, hogy ($képlet$). És azt vizsgáljuk, hogy ez a sorozat konvergens-e. Amennyiben konvergens, a Mandelbrot halmaz eleme. Ekkor fekete színt kapunk. Ha pedig nem eleme, akkor a feketétől eltérőt, színe attól függ, hogy milyen hamar derül ki a számsorról hogy nem konvergens. Ha az általunk vizsgált iterációs hatáértéig ez nem derül ki, akkor a számsort konvergensnek tekintjük.

```
#include <stdio.h>
#include <stdlib.h>
#include <png.h>
#include <sys/times.h>
#include <libpng16/png.h>

#define SIZE 600
#define ITERATION_LIMIT 32000

void mandel (int buffer[SIZE][SIZE]) {

    clock_t delta = clock ();
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int width = SIZE, height = SIZE, iterationLimit = ITERATION_LIMIT;

    // a számítás
    float dx = (b - a) / width;
```

```
float dy = (d - c) / height;
float reC, imC, reZ, imZ, newReZ, newImZ;

int iteration = 0;

for (int j = 0; j < height; ++j)
{
    //sor = j;
    for (int k = 0; k < width; ++k)
    {

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

        reZ = 0;
        imZ = 0;
        iteration = 0;

        while (reZ * reZ + imZ * imZ < 4 && iteration < iterationLimit)
        {
            newReZ = reZ * reZ - imZ * imZ + reC;
            newImZ = 2 * reZ * imZ + imC;
            reZ = newReZ;
            imZ = newImZ;

            ++iteration;
        }

        buffer[j][k] = iteration;
    }
}

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

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

}

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

    if (argc != 2)
    {
        printf("Hasznalat: ./mandelpng fajlnev\n");
        return -1;
    }
}
```

```
/Konstruktor bye,bye
FILE *fp = fopen(argv[1], "wb");
if(!fp) return -1;

png_structp png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, ←
NULL,NULL,NULL);

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

png_set_IHDR(png_ptr, info_ptr, SIZE, SIZE,
8, PNG_COLOR_TYPE_RGB, PNG_INTERLACE_NONE,
PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

png_text title_text;
title_text.compression = PNG_TEXT_COMPRESSION_NONE;
title_text.key = "Title";
title_text.text = "Mandelbrot halmaz";
png_set_text(png_ptr, info_ptr, &title_text, 1);

png_write_info(png_ptr, info_ptr);

png_bytеп row = (png_bytеп) malloc(3 * SIZE * sizeof(png_byte));
int buffer[SIZE][SIZE];
mandel(buffer);

for (int j = 0; j < SIZE; ++j)
{
    //sor = j;
    for (int k = 0; k < SIZE; ++k)
```

```

    {
        row[k*3] = (255 - (255 * buffer[j][k]) / ITERATION_LIMIT);
        row[k*3+1] = (255 - (255 * buffer[j][k]) / ITERATION_LIMIT);
        row[k*3+2] = (255 - (255 * buffer[j][k]) / ITERATION_LIMIT);
        row[k*3+3] = (255 - (255 * buffer[j][k]) / ITERATION_LIMIT);
    }
    png_write_row(png_ptr, row);
}
png_write_end(png_ptr, NULL);

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

```

5.2 A Mandelbrot halmaz a std::complex osztályal

Megoldás videó:

Megoldás forrása: [mandelpngt.cpp](#)

Mandelbort

Figure 5.1: Mandelbrot halmaz

A Mandelbort halmaz a komplex számok síkján alkotott halmaz, amit úgy tudunk vizualizálni, hogy egy kép minden pixelét a hozzáartozó komplex számnak feleltetjük meg(c), majd megnézzük, hogy a komplex számból alkotott sorozat, amelyet úgy kapunk, hogy (képlet). És azt vizsgáljuk, hogy ez a sorozat konvergens-e. Amennyiben konvergens, a Mandelbrot halmaz eleme. Ekkor fekete színt kapunk. Ha pedig nem eleme, akkor a feketétől eltérőt, színe attól függ, hogy milyen hamar derül ki a számsorról hogy nem konvergens. Ha az általunk vizsgált iterációs hatáértékig ez nem derül ki, akkor a számsort konvergensnek tekintjük.

Az std::complex osztály igénybevételével a komplex számokat egy változóban tudjuk tárolni, két paraméterrel.

```

int szelesseg = 1920;
int magassag = 1080;
int iteraciosHatar = 255;

```

A kép felbontása(szélesség x magasság) az általunk vizsgált 4x4-es oldalhosszúságú négyzet részeit feleltetjük meg az általunk megadott kép pixeleivel.

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

Készít egy képet.

```
for ( int j = 0; j < magassag; ++j )
{
    for ( int k = 0; k < szelesseg; ++k )
    {
        reC = a + k * dx;
        imC = d - j * dy;
        std::complex<double> c ( reC, imC );

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

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

            ++iteracio;
        }

        kep.set_pixel ( k, j,
                        png::rgb_pixel ( 64, iteracio%255, (iteracio* -->
                                         iteracio)%255 ) );
    }

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

Ezzel a két ciklussal iterálunk végig a kép pixelein. A koordináták alapján meghatározzuk a komplex szám valós és imaginárius részét. A while ciklussal megvizsgáljuk, a számsorozat első iterációs határnyi elemét, melyek alapján eldöntjük, hogy az adott komplex szám eleme-e a Mandelbrot-halmaznak. Ha a sorozatról kiderül hogy konvergens, azaz a beállított iterációs határunkig nem lép ki a ciklusból, akkor 255 lesz az értéke, így 0-t kapunk vissza, és a pixelt feketére színezzük. Ha nem jutunk el 255 ig, akkor a sorozat nem konvergens, és csak egy 255 nél kisebb értéket kapunk, ennek a pixelnek a színe ettől fog függeni.

5.3 Biomorfok

Megoldás videó: <https://youtu.be/IJMbqRzY76E>

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

A biomorf algoritmus segítségével sokféle és összetett formákat készíthetünk, amelyek egyszerű organizmusokra emlékeztetnek. Az algoritmus komplex számos fraktálokat használ, mint amilyen tipikusan a Mandelbrot fraktál is. A biomophokra a Júlia halmazok kapcsán talált rá Clifford Pickover 1986-ban, mikor egy programot írt a halma kirajzolására. Meg volt győződve róla, hogy felfedezte a Természet Törvényeit, amelyek meghatározzák az élő organizmusok alakját. Sokféle complex függvényt és paramétert kipróbálva, Pickover egy egész "állatkertnyi" képet készített ezekről a primitív élőlényekről.

Miközben Júlia halmazokat akart kirajzolatni, a programjában egy hibát vétett Pickover, és így, véletlen talált rá ezekre a biomorfokra.

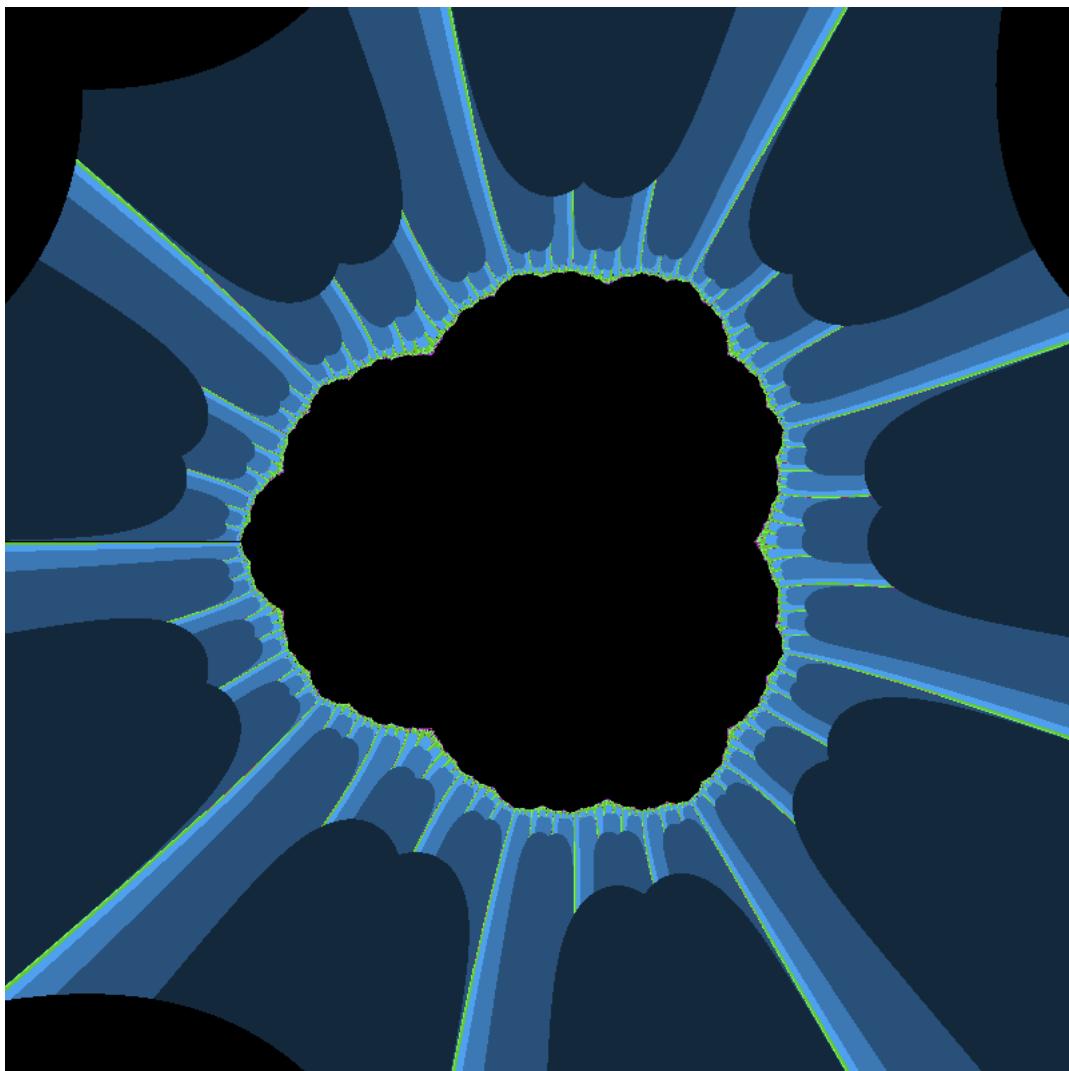


Figure 5.2: Biomorph

Ahhoz hogy mi is ilyen képeket tudjunk alkotni, használhatjuk az előző feladatban használt programkódot, pár fontos változtatással.

A komplex számunk -amivel iterálunk- imaginárius része (C) változó helyett állandó lesz, ez volt a hiba amivel rátalált Pickover a biomorfokra.

```
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;
    }
}
```

5.4 A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

A CUDA-s változat a Mandelbrot halmaz szálasított, a számításokat így gyorsabban elvégző változatát mutatja be.

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

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

#define MERET 600
#define ITER_HAT 32000

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

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

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

    // c = (reC, imC) a rács csomópontjainak
    // megfelelő komplex szám
    reC = a + k * dx;
    imC = d - j * dy;
    // z_0 = 0 = (reZ, imZ)
    reZ = 0.0;
    imZ = 0.0;
    iteracio = 0;
    // z_{n+1} = z_n * z_n + c iterációk
    // számítása, amíg |z_n| < 2 vagy még
    // nem értük el a 255 iterációt, ha
    // viszont elértek, 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 < iteracionsHatar)
{
    // z_{n+1} = z_n * z_n + c
    ujreZ = reZ * reZ - imZ * imZ + reC;
    ujimZ = 2 * reZ * imZ + imC;
    reZ = ujreZ;
    imZ = ujimZ;

    ++iteracio;

}
return iteracio;
}

/*
__global__ void
mandelkernel (int *kepadat)
{
    int j = blockIdx.x;
    int k = blockIdx.y;

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

}
*/

__global__ void
mandelkernel (int *kepadat)
{
    int tj = threadIdx.x;
    int tk = threadIdx.y;

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

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

void
cudamandel (int kepadat[MERET][MERET])
{
    int *device_kepadat;
    cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

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

```
// mandelkernel <<< grid, 1 >>> (device_kepadat);

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

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

}

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

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

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

    int kepadat [MERET] [MERET];

    cudamandel (kepadat);

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

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

    kep.write (argv[1]);
}
```

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

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

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

}
```

5.5 Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

5.6 Mandelbrot nagyító és utazó Java nyelven

Chapter 6

Helló, Welch!

6.1 Első osztályom

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

Megoldás videó:

Megoldás forrása: [PolárGenerátor.java](#) [polargenerator.cpp](#)

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

```
public class PolárGenerátor {  
  
    boolean nincsTárolt = true;  
    double tárolt;  
  
    public PolárGenerátor() {  
  
        nincsTárolt = true;  
  
    }  
  
    public double következő() {  
  
        if(nincsTárolt) {  
  
            double u1, u2, v1, v2, w;  
            do {  
                u1 = Math.random();  
                u2 = Math.random();  
  
                v1 = 2*u1 - 1;  
                v2 = 2*u2 - 1;  
            } while(v1*v1 + v2*v2 >= 1);  
            w = Math.sqrt(1 - v1*v1 - v2*v2);  
            tárolt = v1/w;  
            nincsTárolt = false;  
        }  
        return tárolt;  
    }  
}
```

```
w = v1*v1 + v2*v2;

} while(w > 1);

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

tárolt = r*v2;
nincsTárolt = !nincsTárolt;

return r*v1;

} else {
    nincsTárolt = !nincsTárolt;
    return tárolt;
}
}

public static void main(String[] args) {

    PolárGenerátor g = new PolárGenerátor();

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

}
}
```

A polártranszformációs algoritmus lényege, hogy nem szükséges minden számot tárolni, csak minden páratlanadikat. Ezt a nincsTárolt logikai változónkban tároljuk, amit igazként inicializáltunk. Emellett van egy tárolt nevű double változónk. Lépéseként a számainak véletlenszám generálással kapjuk, de nem mindig kell számolnunk. A páratlan eseteknél a megelőző lépésnél kapott fel nem használt számot használjuk.

```
yhennnon@Yhennon-vm:~/Downloads$ java PolárGenerátor
-0.06702469778212108
1.6788135465046043
-1.0163579873139807
-1.0420280674183082
-2.0468631587032804
-0.6448879344191917
-0.9438274401627531
-1.3611656658003801
-0.08860533420892555
1.3534404152533206
```

C++ beli megvalósítás:

```
#include <cstdlib>
```

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

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

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

            tarolt = r * v2;
            nincsTarolt = !nincsTarolt;

            return r * v1;
        }
        else
        {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }
private:
    bool nincsTarolt;
    double tarolt;
};

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

```
PolarGen pg;

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

return 0;
}
```

A javas megoldás a SUN programozói is így készítették:

```
synchronized public double nextGaussian() {
    // See Knuth, ACP, Section 3.4.1 Algorithm C.
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1; // between -1 and 1
            v2 = 2 * nextDouble() - 1; // between -1 and 1
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}
```

Figure 6.1: SUN módszere

6.2 LZW

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

Megoldás videó:

Az LZW mozaikszó nevében szereplő L és Z betűk az algoritmus alkotóira, Abraham Lempelre és Jakob Zivre utalnak, a W betű pedig Terry Welchre, aki továbbfinomította az Abraham és Jakob által megalkotott algoritmus egy változatát. Ez a finomított változat igen népszerű, tömörítésre használható.

Tegyük fel, hogy van egy szövegünk, tele karakterekkel. minden egyes betű, szám megfelel egy char-nak. A karakterek 8 bitból állnak, tehát 8 darab nulláknak és egyeseknek a kombinációjából. Ezeket az egyeseket és nullásokat épít fel a következő program egy fa mintában.

A megoldás forráskódja: [binfa.c](#)

A fa lényege, hogy egy kezdeti csomópontot kijelölve, amit nevezünk gyökérnek, majd ahhoz igazítva, elkezdjük beolvasni karaktereket, lebontva egyesekre és nullákra, azokkal pedig felépítjük a fát. A nullások a fa bal ágain szerepelnek, az egyesek pedig a jobb oldalon, egy algoritmust követve. Kezdésként a gyökérből indulunk ki, és ha a következő bit 0, akkor a fa bal oldalán készítünk egy csomópontot, ha egyes, akkor a jobboldalára. Így egy új "mélység" keletkezik, ami szépen megmutatja hogy hol van a csomópont. Miután új csomópontot hozunk létre, visszalépünk a gyökérre, és onnan kezdve vizsgáljuk tovább a bitsorozatot. Ha nullás értéket következik, és Kezdésként a gyökérből indulunk ki, és elkezdjük beolvasni a bitsorozatot.

Ha nullás értéket kapunk, és még nem létezik nullás csomópont(a fa egy ága), akkor létrehozzuk a fa bal oldalán, majd visszalépünk a gyökérre. Egyes érték esetén ugyanezt hajtjuk végre, csak a jobb oldalon. Ez a ciklus ismétlődik újra meg újra, amíg el nem fogynak a bitjeink. Ha már létezik olyan csomópont amilyen értéket kapunk, akkor egyszerűen arra a csomópontra lépünk, és itt értékelődik ki a következő bitre, hogy van-e már megfelelő csomópontja. Szemléltetés papíron:

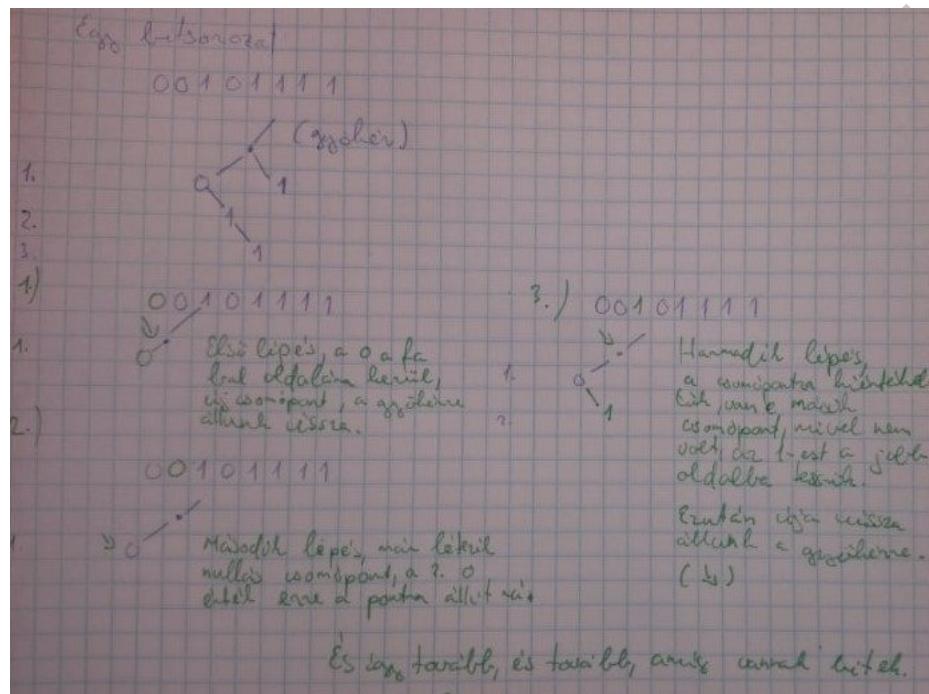


Figure 6.2: Faépítés szemléltetése

Magyarázat a programhoz:

```

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

        if (b & 0x80)

            c='1';
        else
            c='0';
        b <<= 1;

        if (c == '0')

        {
            if (fa->bal nulla == NULL)
            {
                fa->bal nulla = uj_elem ();
                fa->bal nulla->ertek = '0';
                fa->bal nulla->bal nulla = fa->bal nulla->jobb_egy = NULL;
                fa = gyoker;
            }
        }
    }
}

```

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

A while ciklussal beolvassuk a szöveget, a karakterek bitsorozatait maszkolva az 10000000 bitsorozatból álló karakterrel, logikai ÉS-t végrehajtva, amely két egyes esetén egyest,(azaz 128-at, ami 2^8 -on), bármely más esetben pedig nullát ad vissza. Ezután, az if ciklusunk nullás esetén a fentebb leírt és papíron ábrázolt folyamatot hajtja végre. Jól látható, hogy új elemet, új csomópontot hoz létre, ha még nincs, az értéket belerakja, a fa nevű mutatókat pedig a gyökérre állítja.

Végül, de egyáltalán nem utolsó sorban, nagyon fontos felszabadítani a malloc függvényel a binfának lefoglalt memóriaterületet felszabadítani.

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

Először felszabadítjuk a jobboldali részeket, majd a baloldaliakat, és végül magát az elemet, a csomópontot. Ez egy rekurzív folyamat, és nagyon fontos, ezt elhanyagolva memóriaszivárgást okoz a program, ugyanis a manuálisan, malloc-al lefoglalt memória nem szabadulna fel, nem lehetne hozzáférni.

6.3 Fabejárás

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

Megoldás videó:

Az előző, [binfa.c](#) megoldást kell minden össze egy kicsit megváltoztatni.

A fa kiíratása itt történik, a faépítés után:

```
void kiir(BINFA_PTR elem)
{
    if (elem != NULL) {
        ++melyseg;
        kiir(elem->jobb_egy); //JOBB
        //=====
        for (int i = 0; i < melyseg; ++i)
            printf(" --- ");
        printf("%c\n", elem->ertek);
        //=====
        kiir(elem->bal nulla); //BAL
        --melyseg;
    }
}
```

A program mainrészre előtt megírt `kiir` függvényt hívjuk meg, miután a fát megépítettük. Amíg vannak elemeink, addig írjuk a fát, méghozzá olyan módon, hogy a jobb oldalon lévő ágakat írjuk ki először, majd a gyökeret, és végül a bal ágakon lévő elemeket. Ha elfordítanánk a számítógépünk kijelzőjét oldalra, a papíron bemutatott apró mintával megegyező, inorder módon kiíratott fát láthatnánk.

Ahhoz hogy preorder vagy posztorder módon írassuk ki a fát, csupán a `for` ciklusunk helyzetét kell megváltoztatnunk, preorder esetén a jobb és baloldal kiíratása elő, postorder esetén a jobb és balérték kiírása után helyezzük el a `for` ciklust.

6.4 Tag a gyökér

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

Megoldás videó:

Ahhoz, hogy az előző C kódot átültethessük C++ba, egy pár dolgot módosítani kell. C++ban `class`ok, magyarul osztályok vannak `struct` úrák helyett. A feladatban kiutalt Tree és Node osztály nevei az LZWBinFa és a Csomópont osztályok. A [z3a7tagagyoker.cpp](#) kódot használjuk fel ehhez, amiben az osztályba való átírásban kívül más dolgok is hozzáadásra kerültek. A mélységet, a szórást, és az átlagot is kiiratjuk a fa után.

```
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;
            szorasossszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}
```

A feladatban címe, hogy "Tag a gyökér". Ez azt jelenti, hogy az LZWBinFa osztály `protected` részében egy Csomopont gyokeret hozunk létre, a gyökér szoros része a fának, kompozícióban állnak.

```
.
.
protected:
Csomopont gyoker;
```

A programot lefordítva, majd lefuttatva az

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

alábbi módon egy infile-t benne szöveggel, és egy outfile kimeneti célként megadva a következő láthatjuk az outfile ban:

```
yhennon@Yhennon-PC:~/enyimé/udprog-code/source/vedes/első$ cat outfile
-----1(3)
-----1(2)
-----0(3)
----1(1)
-----1(4)
-----0(5)
----1(3)
-----0(2)
-----1(5)
-----1(4)
-----0(3)
--/(0)
-----1(4)
-----1(3)
-----0(4)
-----0(5)
-----0(6)
-----1(2)
-----0(3)
-----0(4)
-----0(1)
-----1(4)
-----1(3)
-----1(5)
-----0(4)
-----0(2)
-----1(4)
-----0(3)
-----1(5)
-----0(4)
depth = 6
mean = 4.36364
var = 0.924416
```

6.5 Mutató a gyökér

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

Megoldás videó:

A megoldás forráskódja: [z3a7masolat.cpp](#)

Tutorom: Petraskó Adrián

Ebben a változatban mutatókat használunk, a kódon több helyen is apró változtatásokat kell végrehajtani ehhez. A fő eltérés, hogy a gyökér egy csomópontra mutató mutató.

```
protected:
```

```
    Csomopont *gyoker;
```

A C programmal ellentétben C++ban a memóriaterület felszabadítását a destruktortvalósítjuk meg. Jele a ~, az adott osztályban szükséges megírni. Miután az osztály egy elemét használtuk, az meghívja a destruktort, így felszabadul a lefoglalt hely.

```
~LZWBinFa ()  
{  
    szabadit (gyoker->egyesGyermek ());  
    szabadit (gyoker->>nullasGyermek ());  
    szabadit (gyoker);  
}  
. . .  
~Csomopont ()  
{  
};  
. . .
```

6.6 Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: [12binfa.cpp](#)

A mozgató konstruktor és értékadás:

```
LZWBinFa ( LZWBinFa && regi ) {  
    std::cout << "LZWBinFa mozgató konstruktor" << std::endl;  
  
    gyoker = nullptr;  
    *this = std::move (regi);  
  
}  
LZWBinFa & operator= ( LZWBinFa && regi)  
{  
    std::cout << "LZWBinFa mozgató értékadás" << std::endl;  
    std::swap (gyoker, regi.gyoker);  
  
    return *this;  
}
```

A mozgató konstruktörben a gyökeret kinullázzuk, majd a `*this` el az aktuális fának a mutatóját a regi-re állítjuk. Így nem kell az összes elemet átmásolni, az új fa a régire és annak értékeire fog mutatni. A `root` egy lvalue, a `regi` pedig egy nem konstans rvalue.

Az értékadás során igénybe vesszük az előző mozgató konstruktort. A `move` tulajdonképpen semmit sem mozgat, a régiből jobbérték(rvalue) referenciát készít, ezzel kiváltja, kikényszeríti a mozgató értékadás hívását, ez a valódi mozgatás így meg végbe.



Chapter 7

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/nbatfai/bhax/tree/master/attention_raising/Myrmecologist

Tanulságok, tapasztalatok, magyarázat...

A vizuális megjelenítéshez a Qt Creator-t használjuk, amiben megírjuk a nekünk szükséges header fájlokat és cppket, összeállítjuk őket, és futtatjuk a programot.

Kezdjük a header fájlokkal :

Első az 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;
};
```

Létrehozzuk az Ant osztályt. Két koordinátája van, az x és az y. Ezek segítségével határozza meg a hangya a helyét a sejttérben. X a sorokat, y az oszlopokat jelöli. Ezen kívül van még egy irány, ez a dir, "direction". A dir-t a qrand() randomszámgenerátor állítja be, 8 lehetséges értéke van. Ez a nyolc irány a négyzetekre felosztott sejttérben egy hangyat körülvevő nyolc négyzet egyike.

antthread.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 &);

};
```

Ebben a headerben több minden történik. Includeoljuk a <QThread< és az előbb megalkotott ant.h headereket. Majd létrehozzuk az AntThread osztályt a QThread alosztályaként, így AntThread minden olyan tulajdonsággal rendelkezni fog, ami szerepel QThreadben. Az AntThread konstruktora argumentumknt megkap rengeteg változót, amit minden taglalunk is. Ebben az osztályban mennek vége a hangyák számításai.

Elkészítünk egy konstruktort is, valamint a program futásához szükséges több függvényt és az osztály private részeként számos változót is, valamint egy step() szignált, amihez szükséges a Q_OBJECT makró, ezt meg is adtunk.

Harmadik header az **antwin.h**.

```
#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCcloseEvent>
#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 ( QCcloseEvent *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 &);

};
```

Kezdésnek, includeolunk több dolgot is, ezekre szükség lesz a többi osztály megalkotásakor, ezek "gyermekeiként" fogjuk esetekben elkészíteni az osztályokat.

Megalkotjuk az AntWin osztályt, ami egy Qt ablak lesz, ehhez szükséges a <QMainWindow<. AntWin konstruktorként megadjuk az ablak méretét, a kezdő hangyák számát(numAnts), a feromon szintet(pheromon), a feromonok elpárolgásának gyorsaságát(evaporation), az iteráció sebességét(delay), és hogy mennyi hangya lehet egyszerre egy cellában(cellAntMax).

Mivel a programban lehetőséget akarunk nyújtani arra, hogy az élet folyásának szemlélése közben meg tudjuk állítani az iterálást, felvehessünk egy képet, vagy bármi más, ezért ezeket eseményekként meg kell írni(event-ek).

Az eventekhez a Qt eventeit vesszük segítségül, ezeket kezelik az eseményeket, a megfelelő headereket használni kell hozzájuk.

```
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();  
}
```

Billentyűzenet a "P" gomb lenyomására szüneteltetjük a programot. A "Q" vagy "Escape" lenyomására bezárjuk a programot, meghívódik a `close()`, amit a következő kezel:

```
void closeEvent ( QCloseEvent *event ) {  
  
    antThread->finish();  
    antThread->wait();  
    event->accept();  
}
```

A `paintEvent`-re is szükségünk lesz, ezzel fogjuk a hangyákat kirajzolni az ablakra.

És végül nem feledkezünk el itt is egy destruktort elkészíteni.

A cpp forrásokban az antthread és antwin headerek még nem definiált függvényeit valósítjuk meg. Ezek a függvények szükségesek a számításokhoz és az elején kifejtett játékszabályok alkalmazásához. A függvények nevei (bár angolul) beszédesek, megszámolják a cellák szomszédait(sumNbhs), az irányokat,irány változtatásokat és az irányokból eredő utat(newDir,detDirs),stb.

A program addig fut futni, amíg a definiált gomblenyomás események(Q és Esc) nem tesznek arról, hogy abbamaradjon. A `run()` fogja vizsgálni a beállított `delay()`, vagyis késleltetés időközönként, hogy fut-e a program. Tehát amíg nem kapcsoltuk ki, vagy nem szüneteltetjük a programot, vagyis amíg a `running = true`, iterálni fog, és elvégzi a `timeDevel-t`, az időhaladást. Ebben a haladásban mozgatja a hangyákat, elvégzi a szabályokat, a párolgást, beállítja az irányokat, minden.

A program futtatva:

Hangya1

Figure 7.1: Hangyaszimulációk

Hangya2

Figure 7.2: Hangyaszimulációk

7.2 Java életjáték

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

Megoldás videó:

Megoldás forrása: <https://bitstorm.org/gameoflife/>

Sejtautomata.java

A John Conway féle életjáték egy pár szabályt követ, amit egy cellákra felosztott felületen alkalmazunk.

- minden olyan benépesített cella, amelynek nincs, vagy csak egy szomszédos benépesített cellája van, elhal "elszigeteltségen".
- minden olyan benépesített cella, amelynek négy vagy több benépesített szomszédja van, elhal "túlnépesedésben".
- minden olyan benépesített cella, amelynek kettő vagy három benépesített szomszédja van, "túlél".
- minden üres, benépesítetlen cella, amelynek három benépesített szomszédja van, szintén benépesedik.

Ezen szabályokat alkalmazva újra és újra iteráljuk a programot, és így szemlélhettük az élet menetét. Ezeket az időFejlődés-ben írjuk le :

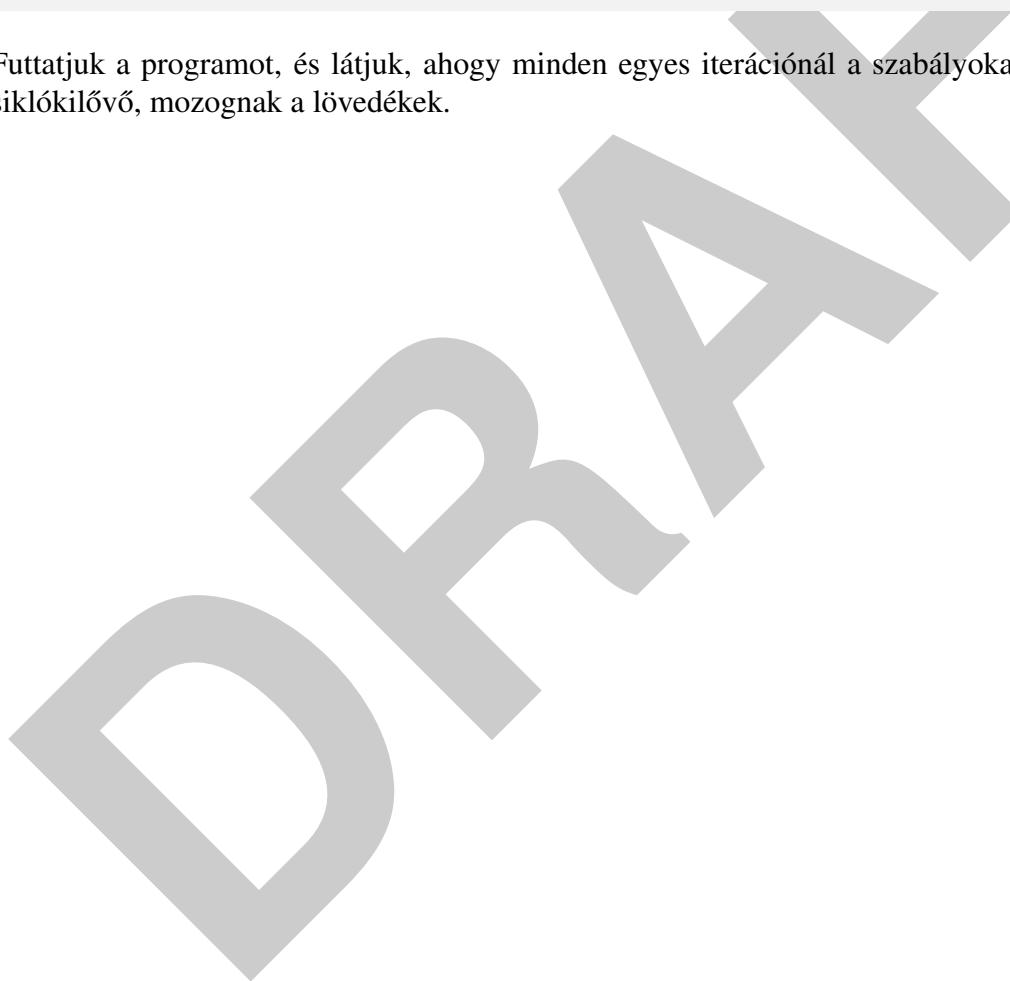
```
public void időFejlődés() {  
  
    boolean [][] rácsElőtte = rácsok[rácsIndex];  
    boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];  
  
    for(int i=0; i<rácsElőtte.length; ++i) { // sorok  
        for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok  
  
            int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);  
  
            if(rácsElőtte[i][j] == ÉLŐ) {  
                /* Élő élő marad, ha kettő vagy három élő  
                szomszedja van, különben halott lesz. */  
                if(élők==2 || élők==3)  
                    rácsUtána[i][j] = ÉLŐ;  
                else  
                    rácsUtána[i][j] = HALOTT;  
            } else {  
                /* Halott halott marad, ha három élő  
                szomszedja van, különben élő lesz. */  
                if(élők==3)  
                    rácsUtána[i][j] = ÉLŐ;  
                else  
                    rácsUtána[i][j] = HALOTT;  
            }  
        }  
    }  
    rácsIndex = (rácsIndex+1)%2;  
}
```

Van egy sejtünk, aminek vannak oszlopai és sorai, ezek megadásával egyes sejteket "életrére" lehet kelteni. Ilyen módon sokféle ábrát, mintát lehet megalkotni, mi most John Conway "sikló kilövő"-jét készítjük el.

```
public void sikló(boolean [][] rács, int x, int y) {  
  
    rács[y+ 0][x+ 2] = ÉLŐ;  
    rács[y+ 1][x+ 1] = ÉLŐ;  
    rács[y+ 2][x+ 1] = ÉLŐ;  
    rács[y+ 2][x+ 2] = ÉLŐ;  
    rács[y+ 2][x+ 3] = ÉLŐ;  
  
}  
public void siklóKilövő(boolean [][] rács, int x, int y) {  
  
    rács[y+ 6][x+ 0] = ÉLŐ;  
    rács[y+ 6][x+ 1] = ÉLŐ;  
    rács[y+ 7][x+ 0] = ÉLŐ;  
    rács[y+ 7][x+ 1] = ÉLŐ;  
  
    rács[y+ 3][x+ 13] = ÉLŐ;  
  
    rács[y+ 4][x+ 12] = ÉLŐ;  
    rács[y+ 4][x+ 14] = ÉLŐ;  
  
    rács[y+ 5][x+ 11] = ÉLŐ;  
    rács[y+ 5][x+ 15] = ÉLŐ;  
    rács[y+ 5][x+ 16] = ÉLŐ;  
    rács[y+ 5][x+ 25] = ÉLŐ;  
  
    rács[y+ 6][x+ 11] = ÉLŐ;  
    rács[y+ 6][x+ 15] = ÉLŐ;  
    rács[y+ 6][x+ 16] = ÉLŐ;  
    rács[y+ 6][x+ 22] = ÉLŐ;  
    rács[y+ 6][x+ 23] = ÉLŐ;  
    rács[y+ 6][x+ 24] = ÉLŐ;  
    rács[y+ 6][x+ 25] = ÉLŐ;  
  
    rács[y+ 7][x+ 11] = ÉLŐ;  
    rács[y+ 7][x+ 15] = ÉLŐ;  
    rács[y+ 7][x+ 16] = ÉLŐ;  
    rács[y+ 7][x+ 21] = ÉLŐ;  
    rács[y+ 7][x+ 22] = ÉLŐ;  
    rács[y+ 7][x+ 23] = ÉLŐ;  
    rács[y+ 7][x+ 24] = ÉLŐ;  
  
    rács[y+ 8][x+ 12] = ÉLŐ;  
    rács[y+ 8][x+ 14] = ÉLŐ;  
    rács[y+ 8][x+ 21] = ÉLŐ;  
    rács[y+ 8][x+ 24] = ÉLŐ;  
    rács[y+ 8][x+ 34] = ÉLŐ;  
    rács[y+ 8][x+ 35] = ÉLŐ;
```

```
rács[y+ 9][x+ 13] = ÉLŐ;  
rács[y+ 9][x+ 21] = ÉLŐ;  
rács[y+ 9][x+ 22] = ÉLŐ;  
rács[y+ 9][x+ 23] = ÉLŐ;  
rács[y+ 9][x+ 24] = ÉLŐ;  
rács[y+ 9][x+ 34] = ÉLŐ;  
rács[y+ 9][x+ 35] = ÉLŐ;  
  
rács[y+ 10][x+ 22] = ÉLŐ;  
rács[y+ 10][x+ 23] = ÉLŐ;  
rács[y+ 10][x+ 24] = ÉLŐ;  
rács[y+ 10][x+ 25] = ÉLŐ;  
  
rács[y+ 11][x+ 25] = ÉLŐ;  
  
}
```

Futtatjuk a programot, és látjuk, ahogy minden egyes iterációt alkalmazva "dolgozik" a siklókilővő, mozognak a lövedékek.



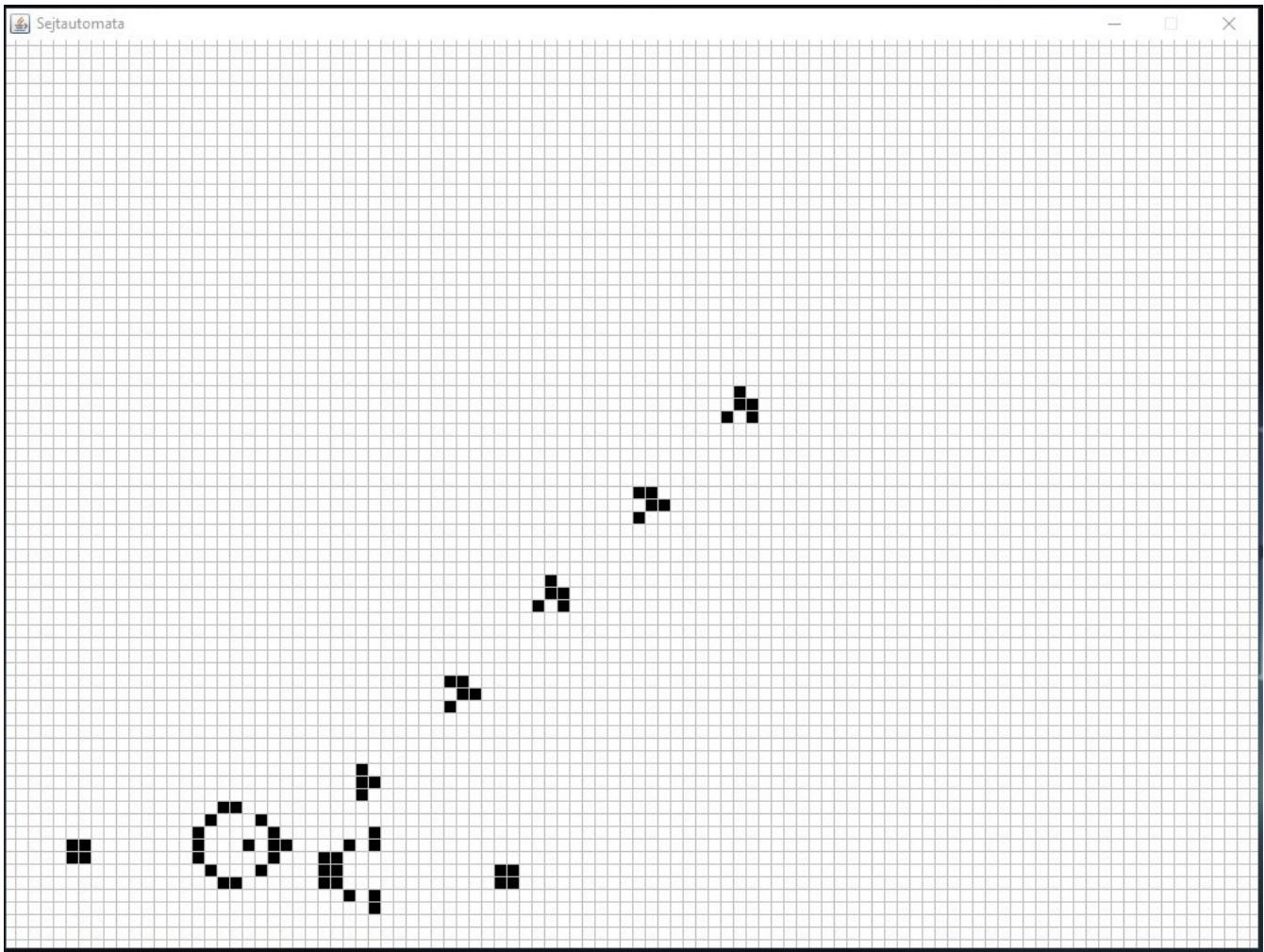


Figure 7.3: Életjáték

Többféle lehetőség is implementálva van a forrásban. Egérgombot lenyomva a sejttérben életre kelthetünk sejteket. Az s-t lenyomva a billentyűzeten felvételt csinálhatunk. Az n-et lenyomva növeljük a sejtméretet, i-vel növelni az iteráció sebességét, g-vel lassítani, k val a sejtméretet lehet csökkenteni.

7.3 Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://bitstorm.org/gameoflife/>

`bhax/thematic_tutorials/bhax_textbook/codes/nyolcadik_het/build-myrmecologist
Qt_5_12_2_GCC_64bit-Debug/myrmecologist`

Az életjáték Java-beli és C++ beli megoldása csak szemantikában tér el,

Képek a futó programról:

7.4 BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

Chapter 8

Helló, Schwarzenegger!

8.1 Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása: Progpater: https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol?token=ce0dc57jTe4WMZcvZc

[softmax.py](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat hasonlít arra, amikor neuralnetnek tanítottuk az AND, OR és EXOR-t. Most számok felismerésére fogjuk tanítani, és pythont használunk R helyett. A MNIST adatbázis rengeteg számot tartalmaz, vagyis azoknak a fényképeit, amiket kézzel írtak. Ezeket lehet jól használni gépek tanítására. Egy virtuális környezetet fogunk használni("venv"-> virtual environment), és a TensorFlow-ot. A TensorFlowot a google fejlesztette ki, segítségével gráfokat készítünk a számításokról. Én itt olvastam róla (de persze sok másol is lehet): <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html>

Szokásosan, a forrás elején importáljuk a megfelelő könyvtárakat, köztük a tensorflowt is, amit a sok későbbi használat megkönnyítése céljából tf néven hívunk be.

Fontos beállítanunk, hogy vezessünk egy naplót, amiben a hibákat fogjuk elkönyvelni, azért hogy majd látassuk a tanulás során a hibaárányt, és fejleszthessük a tanulást.

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

import argparse

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

```
import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

import matplotlib.pyplot
```

Mint ahogyan a neurális hálónál a logikai műveletek tanításánál, megadjuk a bemeneti értékeket és a várt eredményt, plusz azt, hogy 784 pixeles képeket kap bemenetként(*placeholder*-be). A várt eredmény, *y* értéke 0-1 között van.

```
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
```

Az előbb említett rossz válaszokat, azoknak arányát a *cross_entropy*-ban tároljuk. Az *y* várt értéke és a gép által megbecsült érték közti különbséget szeretnénk minimalizálni.

A tanítás lépésekben fog így állni, az eredménybeli eltérést is lépésről lépésre egyre jobban akarjuk kiküszöbölni legalább egy minimumig, egy beépített függvényt használunk fel, a *GradientDescentOptimizer*-t.

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

sess = tf.InteractiveSession()
```

Mostmár egy inicializálás után futtathatjuk a programunkat, és tesztelhetünk képeket.:

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

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

```
print("-- A MNIST 42. tesztképenek felismerése, mutatom a számot, a ←
      továbblepeshez csukd be az ablakat")

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

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

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

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

#print("-- A saját kezi 8-asom felismerése, mutatom a számot, a ←
      továbblepeshez csukd be az ablakat")
print("-- A MNIST 20. tesztképenek felismerése, mutatom a számot, a ←
      továbblepeshez csukd be az ablakat")
# img = readimg()
# image = img.eval()
# image = image.reshape(28*28)
img = mnist.test.images[20]
image = img
matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm. ←
    binary)
matplotlib.pyplot.savefig("20.png")
matplotlib.pyplot.show()

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

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

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

8.2 Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3 Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4 Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5 Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Chapter 9

Helló, Chaitin!

9.1 Iteratív és rekurzív faktoriális Lisp-ben

Megoldás video: <https://www.youtube.com/watch?v=z6NJE2a1zIA>

Megoldás forrása:

A Lisp (vagyis List Processing) egy programozási nyelvcsalád, többféle "dialektusa" van. A lisp adatstruktúrája láncolt listákból épül fel, és igen érdekes, hogy a megszokott infix alak helyett prefix alakot használ. Egy összeadásnál például az összeadásjel(az operandus) nem a két változónk között, hanem azok előtt szerepel.

- Infix összeadás: $2+2$
- Prefix összeadás: $+ 2 2$

Másik érdekes dolog, hogy a LISP-es nyelvekben a kódok rekurzívan futnak le. Feladatunk most, hogy a faktoriális függvényt mind rekurzív, mind iteratív módon valósítsuk meg egy LISP-es nyelven. A Scheme dialektust fogjuk ehhez használni, amit a GIMP médiaszerkesztő alkalmazásban script-fu konzolban el is érhetünk.:)

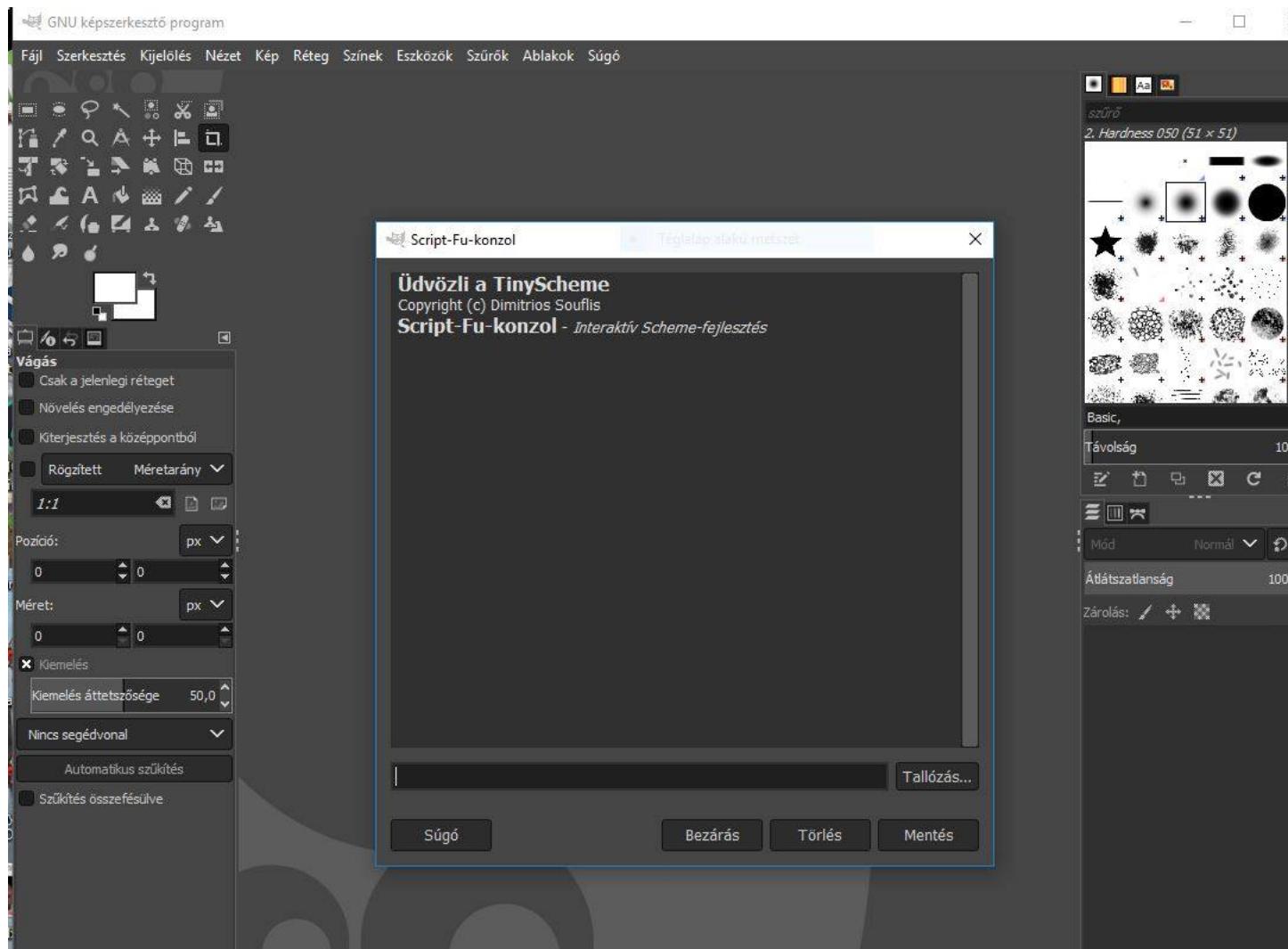
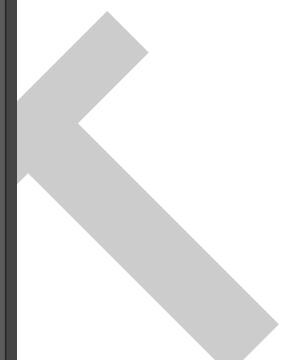


Figure 9.1: Script-fu konzol

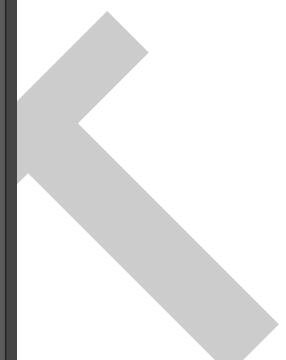
Saját függvényeket a `define`-al tudunk definiálni. Maga a faktoriális függvény igen egyértelmű, azthiszem itt most nem kell elmagyarázni. A Scheme-es rekurzív faktoriális függvény így néz ki :



```
Script-Fu-konzol
> (define (fakt n) (if (< n 1) 1 (* n(fakt(- n 1)))))
fakt
> (fakt 12)
479001600
```

Megadjuk a define után milyen néven szeretnénk megalkotni a függvényt. Egy `if`-et használunk. Ami legelőször azt nézi, hogy a beírt `n` számunk kisebb e 1-nél, ha igen , akkor befejeződik a függvény, ha nem kisebb, akkor pedig `n`-et szorozzuk egyel csökkentett változatával. Ez addig megy amíg `n` kisebb mint 1 nem lesz.

Ebben a nyelvben nincsenek iterációs ciklusok,mint amilyen pl. a `for`. Ha iteratív megoldást szeretnénk, azt magunknak kell megalkotni, de ettől még rekurzívan zajlanak a dolgok:



```
Script-Fu-konzol
> (define (fact_iter n) (do (( i 1 (+ i)) (num 1 (* i num))) ((> i n) num)))
fact_iter
> (fact_iter 10)
3628800
```

9.2 Gimp Scheme Script-fu: króm effekt

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

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome_bhax_chrome3.scm

Tanulságok, tapasztalatok, magyarázat...

A feladatban az előző feladathoz hasonlóan GIMP-et használunk. A forrás:

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)
; (color-curve)

(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-wh text font fontsize)
(let*
  (
    (
      (text-width 1)
      (text-height 1)
    )
    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
      PIXELS font)))
    (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
      fontsize PIXELS font)))
    (list text-width text-height)
  )
)
```

```
)  
)  
  
;(text-width "alma" "Sans" 100)  
  
(define (script-fu-bhax-chrome text font fontsize width height color ←  
    gradient)  
(let*  
    (  
        (image (car (gimp-image-new width height 0)))  
        (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←  
            LAYER-MODE-NORMAL-LEGACY)))  
        (textfs)  
        (text-width (car (text-wh text font fontsize)))  
        (text-height (elem 2 (text-wh text font fontsize)))  
        (layer2)  
    )  
  
;step 1  
(gimp-image-insert-layer image layer 0 0)  
(gimp-context-set-foreground '(0 0 0))  
(gimp-drawable-fill layer FILL-FOREGROUND)  
(gimp-context-set-foreground '(255 255 255))  
  
(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←  
    ))  
(gimp-image-insert-layer image textfs 0 0)  
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←  
    height 2) (/ text-height 2)))  
  
(set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←  
    LAYER)))  
  
;step 2  
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)  
  
;step 3  
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)  
  
;step 4  
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)  
  
;step 5  
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))  
(gimp-selection-invert image)  
  
;step 6  
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←  
    LAYER-MODE-NORMAL-LEGACY)))  
(gimp-image-insert-layer image layer2 0 0)
```

```
;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT- ←
    LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
        3)))
;

;step 8
(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ←
    0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 '(255 0 0) " ←
    Crown molding")

(script-fu-register "script-fu-bhax-chrome"
    "Chrome3"
    "Creates a chrome effect on a given text."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 19, 2019"
    ""
    SF-STRING      "Text"          "Bátf41 Haxor"
    SF-FONT        "Font"          "Sans"
    SF-ADJUSTMENT  "Font size"   '(100 1 1000 1 10 0 1)
    SF-VALUE       "Width"         "1000"
    SF-VALUE       "Height"        "1000"
    SF-COLOR       "Color"         '(255 0 0)
    SF-GRADIENT    "Gradient"     "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
    "<Image>/File/Create/BHAX"
)
```

Egy beírt szövegre egy krómhatást szeretnénk alkalmazni, ami a következőképpen néz ki:



siker!

Ez a színezés:

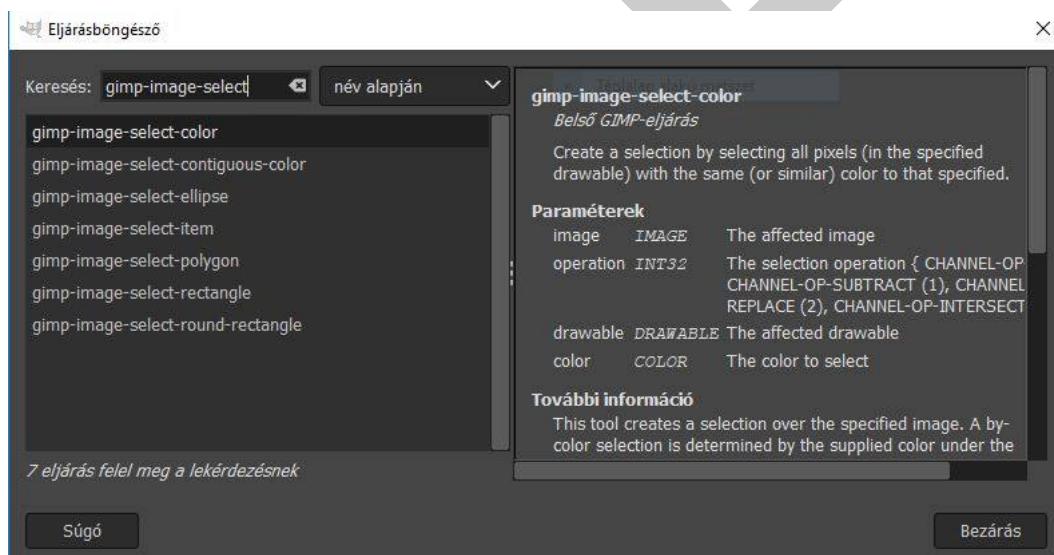
```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
```

```
(aset tomb 7 190)
tomb)
)
```

Egy program egy kiterjesztés a GIMP-hez, bemásolva a scriptek közé, a File-Create alatt egy új, "BHAX"-ként elnevezett lehetőség válik elérhetővé, így állítottuk be.

```
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```

A programban saját függvényeket is alkotunk a `define-al`, és a GIMP és a nyelvezet beépített függvényeit is használjuk. Ezeket a beépítetteket az Súgóból az eljárásböngészőt megnyitva megtekinthetjük, és leírást is kapunk róluk, pl.:



A kiterjesztést lehet a beállított File-Create-BHAX ról, és script-fu terminálba írva is használni. Script-fu:

```
(script-fu-bhax-chrome "Siker!" "Sans" 120 1000 1000 ' (255 0 0) "Crown ←
molding")
```

9.3 Gimp Scheme Script-fu: név mandala

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

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala_bhax_mandala9.scm

Tanulságok, tapasztalatok, magyarázat...

A mandala egy szanszkrit eredetű szó, aminek a jelentése kör. A hinduizmust gyakorlók körében évezredek óta használt segédeszköz, és emellett szépsége miatt díszítőművészeti formában is elterjedt.

Elkészítése egy pár érdekes lépésből áll. Többféle betűtípusral és rövidebb-hosszabb szöveggel dolgozhatunk, mindenféleképpen nagyon szép mandalákat alkothatunk. Most is a GIMP-et használjuk, a szöveget középre leírjuk, vízszintesen tükrözük és a tükrözöttet a szöveg tetejére illesztjük, a kettőt összeillesztjük. Az összeillesztett alakzatot lemásoljuk, középre illesztjük, és elforgatjuk 90 fokkal. Ezeket megint összeillesztjük, másoljuk, forgatjuk 45 fokkal, és ugyanez megegyezik, a végén 30 fokkal elforgatva. Egy kört kapunk, ezt még bekeretezzük, és kész a mandala.

Ezt a már előző feladatban megemlített függvényekkel megalkotjuk, integráljuk a script-et GIMP-be.

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  text-width
)
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  ;;;
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  ;;; ved ki a lista 2. elemét
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
  ;;;
  (list text-width text-height)
)
)
```

```
; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
    gradient)
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
        LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-layer)
    (text-width (text-width text font fontsize)))
  ;;;
  (text2-width (car (text-wh text2 font fontsize)))
  (text2-height (elem 2 (text-wh text2 font fontsize)))
  ;;;
  (textfs-width)
  (textfs-height)
  (gradient-layer)
  )
  (gimp-image-insert-layer image layer 0 0)

  (gimp-context-set-foreground '(0 255 0))
  (gimp-drawable-fill layer FILL-FOREGROUND)
  (gimp-image-undo-disable image)

  (gimp-context-set-foreground color)

  (set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
  (gimp-image-insert-layer image textfs 0 -1)
  (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
    height 2))
  (gimp-layer-resize-to-image-size textfs)

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
  (set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
  (gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
  (set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
    -LAYER)))

  (set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
  (gimp-image-insert-layer image text-layer 0 -1)
```

```
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
-LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
-LAYER)))

(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car(gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car(gimp-drawable-height textfs)) 100))

(gimp-layer-resize-to-image-size textfs)

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ←
"gradient" 100 LAYER-MODE-NORMAL-LEGACY)))

(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ (/ ←
width 2) (/ textfs-width 2)) 8) (/ height 2))

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)
```

```
(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
    )))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
    height 2) (/ text2-height 2)))

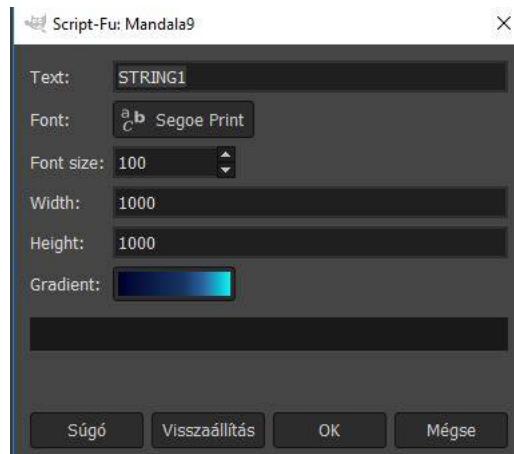
;(gimp-selection-none image)
;(gimp-image-flatten image)

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Ruge Boogie" 120 1920 ←
 1080 '(255 0 0) "Shadows 3")

(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"          "Bátf41 Haxor"
  SF-STRING      "Text2"         "BHAX"
  SF-FONT        "Font"          "Sans"
  SF-ADJUSTMENT  "Font size"    '(100 1 1000 1 10 0 1)
  SF-VALUE       "Width"         "1000"
  SF-VALUE       "Height"        "1000"
  SF-COLOR       "Color"         '(255 0 0)
  SF-GRADIENT    "Gradient"     "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

Az alul elhelyezkedő rész felelős azért, hogy ne kelljen script-fu konzolt használni, a felhasználó kiválaszthatja a létrehozásnál a Mandala9 névvel illetett opciót, és egy interaktív felületet fog látni maga előtt amivel a szöveget, a betűtípus, méreteket, színezést, átmenetet ki tudja választani:

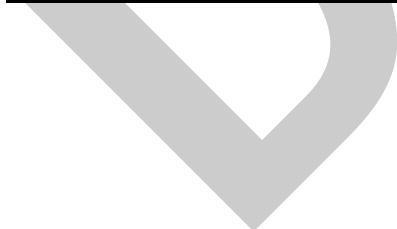
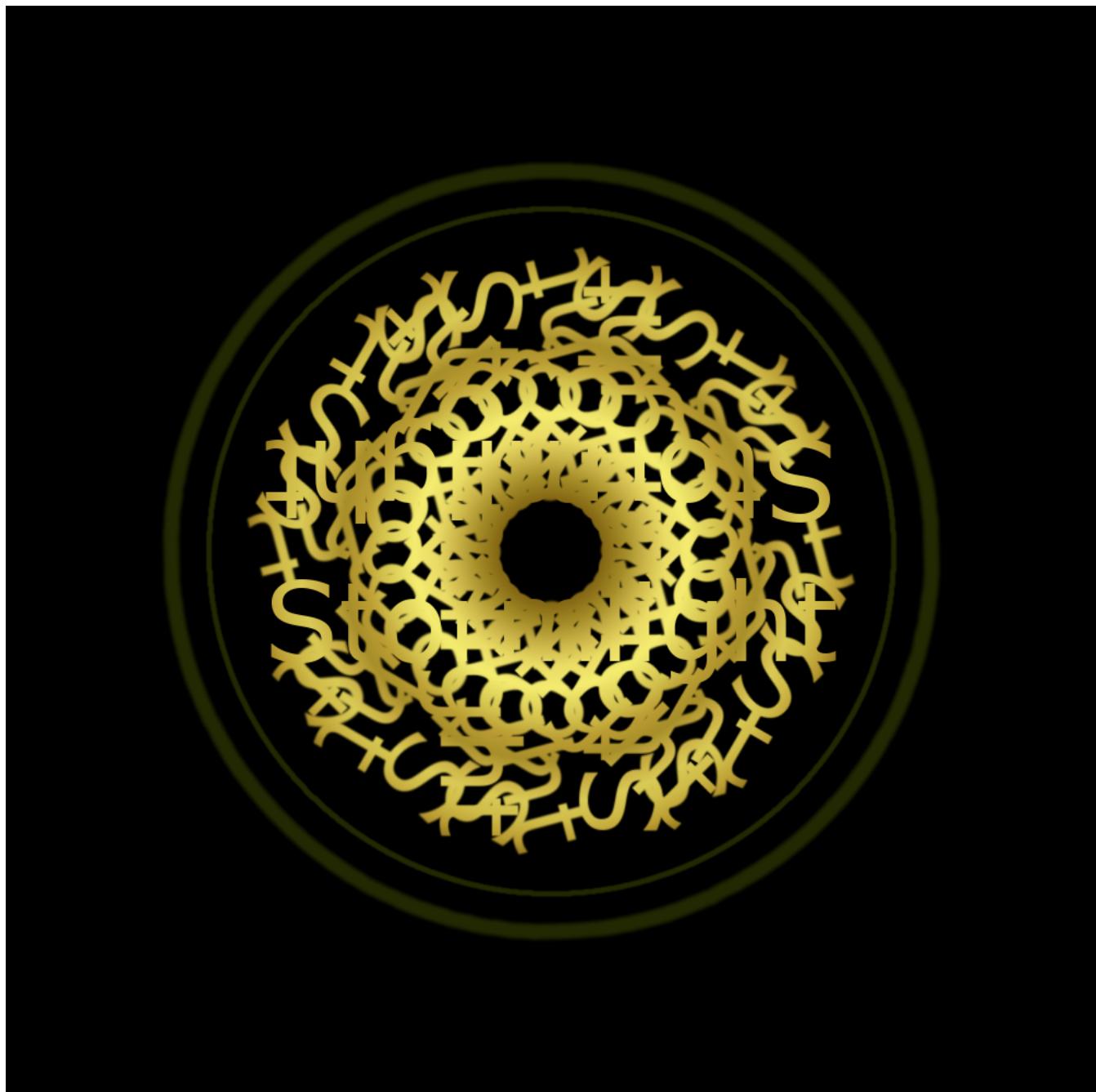


Storm feliratból mandala:

DRAFT



Stormlight feliratból mandala:



Chapter 10

Gutenberg

10.1 Programozási alapfogalmak

PICI

1.2 Alapfogalmak

A programozási nyelveknek három szintje létezik, magas szintű nyelv, assembly nyelv, és gépi nyelv. Ahhoz hogy használni tudjunk egy programot, először meg kell írni, (ehhez használunk egy magas szintű nyelvet) majd olyan formára hozni amit a gép már olvasni tud, tehát gépi nyelvre. Az assembly nyelv közel áll a gépi nyelvhez, de azt még nem tudja olvasni, ezt a kódot is le kell fordítani gépi kódra.

A választott magas szintű nyelvvel írt kódot forráskódnak nevezzük. Ennek a nyelvnek megvannak a maga szabályai, a szintaktikai szabályok -amelyekre gondolhatunk nyelvtani szabályokként-, és a szemantikai szabályok, amelyek kifejezik a leírt dolgok tartalmát, értelmét. minden programnyelvnek a hivatkozási nyelve írja le ezeket a szabályokat részletesen. A forráskódot a gép nem tudja olvasni, ezért le kell fordítani gépi nyelvre. Ezt tehetjük egy fordítóprogram segítségével, vagy interpretációval.

A fordítóprogram olyan program, amely forráskódból tárgykódot készít. Fordítás közben a program a forráskódon szemantikai, szintaktikai, és lexikális elemzést végez, majd ha minden rendben talál, legenerálja a tárgykódot. Így a kód már gépi nyelven van, ezt még futtathatóvá kell tenni, amit a kapcsolatszerkesztő végez el

Az interpreter esetében nem készítünk tárgykódot, hanem az interpreter a forráskódot fogja értelmezni majd végrehajtani, és egyből lefut a program.

1.3. A programnyelvek osztályozása

A nyelveket többféleképpen lehet osztályozni. Vannak imperatív, deklaratív és egyéb olyan nyelvek, (-és ezek alcsoportjai) amelyek nem esnek bele az előző két kategóriába. Az imperatív nyelv esetében utasításokat írunk, algoritmusokat, és ezt értelmezi a gép mikor végrehajtjuk a programot. A deklaratív nyelv nem algoritmikus, esetükben megfogalmazunk valamit amit végre szeretnénk hajtatni, a megfelelő módon megírjuk a kódot, és a program elkészíti a megoldást.

2.4 Adattípusok

Miközben programot írunk, gyakran elvonatkoztatunk a valóságtól, absztrakt módon közelítjük meg ezt a valóságot, így tudunk értelmezni olyan dolgokat, amiket különben teljes egészükben tekintve túl nagy

feladat lenne. Amiről most részletesebben beszélni fogunk, az ennek az absztraktiónak a programozásban egy eszköze, az adattípus. A nyelveknek megvan a saját adattípusai, és vannak olyan nyelvek, amelyek megengedik, hogy a programozó saját típusokat hozzon létre, és kedve szerint alkossa meg azt, pontosabban leírva mit is szeretne modellezni. Egy adattípusnak van tartománya, művelete, és reprezentációja.

A tartományban tartalmazza azokat az értékeket, amiket az adott adattípussal rendelkező eszköz fel tud venni.

Az adattípus művelete (vagy műveletei) a megadott tartománybeli értékekkel tud dolgozni.

A reprezentáció pedig azt jelenti, hogy az adattípus szemléletesen hogyan van jelen a kódban. Meghatározza, hogy az egyes elemek milyen bitkombinációra képződnek le a tárban.

2.5 A nevesített konstans

A nevesített konstansokról— A nevesített konstansoknak van neve, típusa, és értéke, mint az átlagos változóknak, de van különbség, az előbbieknek minden kötelező értéket adni mikor megalkotjuk őket, minden deklarálni kell. Ezeket a konstansokat a program elején lehet deklarálni a következőképpen: `#define név literál Célja`, hogy már így az elején megadva a típusukat, értéküket, és a nevüket, a programban a nevüket használva minden helyettesítve lesznek, az értékükhez csak itt lehet hozzányúlni, de így kényelmessé teszi, az alapértéket átírva minden előfordulásában ugyanaz az érték fog szerepelni

Viszont tudni kell, hogy nem minden programozási nyelv nyújt lehetőséget saját nevesített konstans létrehozására, vagy egyáltalán nem tartalmaznak nevesített konstanst. Az előző példa C nyelvben használatos.

2.6. A változó

A változóknak négyféle tulajdonsága van. Név, attribútumok, cím, és érték. A programban a változóra a nevével tudunk hivatkozni, hasonlóan az előbb megbeszélt nevesített konstansokhoz.

A név jelöli minden tulajdonságát a változónak.

Attribútumból több is van, ezek a tulajdonságok írják le a változó működését a futó programban. Egyik legfontosabb attribútum a változó típusa, ami jelöli, hogy milyen féle értéket vehet fel. (Ilyen típus például az integer, karakter, lebegő pontos, stb.)

A változó címe megmutatja, hogy hol helyezkedik el a tárban, memóriában, és ezt a helyet többféleképpen lehet allokálni:

- Programozó által vezérelt memóriakiosztás
- Statikus memóriakiosztás
- Dinamikus memóriakiosztás

Statikus kiosztásnál előre meghatározott a cím, betöltődik futás alatt a tárba, és közben nem változik. Dinamikus kiosztásnál a futás közben a rendszer címezí meg a változót, a cím változhat futás közben, és akkor van tényleg címe, amikor aktív a programegység amelyben a változó szerepel. Mikor ezeken a módokon tárhelyet foglalunk le, gondolunk kell rá hogy fel is kell szabadítani ezt a foglalást, különben „szivárogni” fog a memória.

A változók értékét meg lehet adni inicializálásuk során, de később szükség lehet az értékük megváltoztatására, ekkor ezt egy értékkadó utasítással megtehetjük.(Értéket adni pedig kötelező, különben nem tudjuk használni a változót.) Az utasítás jele „=”. Az egyenlőség jel bal oldalán a változó neve szerepel, a jobb oldalon az a kifejezés, amit új értékül szeretnénk adni. Fontos figyelembe venni, hogy egyes nyelvek

megkövetelik a típusegyenértékűséget, ekkor a kifejezés típusa megegyező kell legyen a változó típusával. Ha nem követeli meg a nyelv, akkor használhat típuskényszerítést, ilyenkor a kifejezés típusát a változó típusához igazítja.

2.7. Alapelemek az egyes nyelvekben

Vannak aritmetikai típusok, amin belül az integrális típusokhoz tartozik az egész(int), karakter(char), és a felsorolásos típus, a valós típusokhoz pedig a float, double, és long double, amelyek különböző bit méretű tört számok. Léteznek még származtatott típusok, ezek a tömb, függvény, mutató, struktúra, union. Végül pedig az üreset visszadó void típus.

3. KIFEJEZÉSEK

A kifejezések szintaktikai eszközök. Egy már definiált változónak lehet vele új értéket adni különböző módokon. A kifejezéseknek három része van, az operandus, az operátor, és az opcionális kerek zárójelek.

Az operandus alatt változókat, konstansokat, literálokat és függvényhívásokat értünk, ezeknek az értékét. Az operátorok a műveleti jelek (például "=", "+", "/", és sok más). A kerek zárójelek segítségével a műveletek sorrendje tetszés szerint változtatható

Egyoperandusú, kétoperandusú és háromoperandusú operátorok is vannak, ez azt jelöli, hogy operandussal dolgozik az operátor. Például egyoperandusú a negáció(!), kétoperandusú az összeadás (+), háromoperandusú a feltételes operátor (?).

A kifejezéseknek lehet prefix, infix, és postfix alakja. Prefix esetén az operátor az operandus(ok) előtt áll, postfix esetén az oprandus(ok) után, infix esetén pedig közöttük.

A végrehajtási sorrend, amelyben elvégezzük a kifejezések által meghatározott műveletet szintén többféle lehet. Lehet felírás sorrendjével megegyező, azazl balról jobbra haladva, felírás sorrendjével ellenkező, jobbról balra, és balról jobbra egy precedencia táblázat szabályait követve.

A folyamatot amely során elvégezzük a műveletet, kifejezés kiértékelésnek nevezzük. Ennek során minden a kifejezés értéke, minden a típusa kiértékelésre kerül.

Azért lehet szükség precedencia táblára, amely leírja, hogy milyen sorrendben kell elvégezni az operátorokat, mert nem mindegyik alak egyértemű (ilyen az infix alak). A precedencia táblázatban egymás alatt szerepelnek sorok, a fentebb lévő sorok operátorai előbb elvégezendőek, és van egy kötési irány (itt szintén, lehet balról jobbra, és fordítva is), ami megmutatja hogy az azonos sorban szereplő operátorok között is melyik élvez elsőbbséget. Tehát ha precedencia táblázatot használunk egy infix kifejezés kiértékelésére, akkor azt a következő képpen tesszük. A kötési irányt követve, ha egy operátor van, akkor elvégezzük azt, ha több, akkor összehasonlítjuk az első két operátort, és ha az első operátor erősebb a másodiknál, vagy azonos erősségűek, akkor elvégezzük az első operátor által kijelölt műveletet, majd ha vannak még operátoraink, akkor lépünk a következőre, és ismétünk amíg el nem fogynak. Így az elsőnek elvégezendő művelet pontosan meghatározható

Ahhoz hogy el tudjuk végzeni a műveleteket, meg kell határozni az operandusok értékét, a C nyelvben ez tetszőleges, és vannak olyan nyelvek, amelyek szabályt követnek. Mivel az infix alak nem egyértelmű, ezért itt lehet szükség a zárójelekre, amelyek segítségével -az előbb megismert kifejezés kiértékelés menetét és a precedencia táblázat szabályait figyelembe véve- tetszőleges műveleti sorrendet tudunk kialakítani.

Azok a kifejezések, amelyekben logikai operátor szerepel, speciálisak, mert az operátortól függően nem feltétlen kell végig csinálni a kiértékelést. Vannak olyan operátorok, amelyek olyan műveleteket jelölnek, amelyekben már az első operandus értéke eldönti a végkimenetelt. Ilyen a logikai ÉS, amely igaz értéket ad vissza, ha minden operandus értéke igaz, különben pedig hamisat, tehát ha az első, vagy bármelyik más

érték hamis, már nem számít a többi operandus értéke, tudjuk a végeredményt. Vagy ilyen a logikai VAGY, amely hamis értékek esetén hamisat, különben pedig igaz értéket ad vissza, tehát az első igaz érték után tudjuk, hogy hamis lesz az eredmény.

A nyelvek ilyen esetek kiértékelésénél megintcsak többféleképpen járhatnak el:

- Teljes kiértékelés, ekkor a speciális esetekben is végig viszi a kiértékelést.
- Rövidzár kiértékelés, csak addig folytatja a kiértékelést, ameddig biztosan el nem dől az eredmény.
- A programozó dönti el a kiértékelés módját, lehetnek rövidzár operátorok és nem rövidzár operátorok is.
- A kiértékelés módjának beállítására futás közben ad lehetőséget.

A kifejezések értékét így mebeszéltük, most pedig jöjjön a kifejezések típusa. A nyelvek kétféle elvet követhetnek, a típuskényszerítést és a típusegyenértékűséget megkövetelőt.

A típusegyenértékűséget megkövetelő nyelvek esetében, ha kétoperandusú kifejezés kiértékelést végünk, akkor az operandusok típusának azonosnak kell lenniük. Az eredmény típusa az operandusok típusával megegyező lesz, vagy az a típus, amit az operátor ad eredményül. Fontos meghatározni, hogy mikor azonosak, egyenértékűek a típusok.:

- Ha a kifejezések azonos típusnévvel vannak deklarálva, azonos utasításban, akkor deklaráció egyenértékűség áll fenn.
- Ha a kifejezések azonos típusnévvel vannak deklaráva, de nem feltétlenül azonos deklarációs utasításban, akkor név egyenértékűség áll fenn.
- Ha a kifejezések azonos típusúak, és szerkezetük is azonos, akkor struktúra egyenlőség áll fenn.

A típuskényszerítést követő nyelvek esetén a kiértékelésben szereplő operandusok típusai többfélék is lehetnek, de ekkor konvertálódnak, mert a műveleteket csak azonos típusú operandusokkal lehet elvégezni. A kiértékelés megszokottan a leírt módon történik, azzal a különbséggel, hogy az eredményül kapott kifejezés típusa az operátor által meghatározott eredménytípus lesz. Ha több kifejezés és operátor van, akkor minden művelet után kiértékeli a részkifejezés típusát, és azokkal az értékekkel végzi el a kiértékelést.

Vannak olyan nyelvek, amelyek megengedik a típuskényszerítés egy speciális esetét a számokkal való műveletek esetében, akkor is, ha nem típuskényszerítő elvet vallanak. Ezek az esetek a szűkítés és a bővítés. Ha a konvertálandó típus minden eleme szerepel a céltípus lehetséges elemei között, akkor a konvertálás értékesztés nélkül történik, ez a bővítés. Ha pedig nem igaz ez a konvertálandó típus elemeire, akkor is elvégezhető a művelet, de kerekítés, vagy akár értékesztés is történhet, ez a szűkítés, aminek a szabályait külön meg kell határoznia a programozónak.

4. UTASÍTÁSOK

Az utasítások azok az egységek, amelyeknek a segítségével írjuk meg a kód algoritmusait, és instrukciókat adunk a fordítóprogramnak, amiknek a segítségével tárgykódot tud generálni. Két nagy csoportja van az utasításoknak, a deklarációs és a végrehajtható utasítások.

A deklarációs utasítások a fordítóprogramnak szólnak, felhasználandó értékeket, üzemmód beállításokat, szolgáltatások kérését fogalmazza meg. Ezeket fogja felhasználni a fordítóprograma tárgykód generálásához. Ezek az utasítások maguk nem generálnak tárgykódot, vagyis nem kerülnek lefordításra, de ahoz szükségesek, hogy a végrehajtható utasításokból a fordítóprogram elkészíthesse a tárgykódot. A végrehajtható utasításoknak több csoportja van.:

- 1. Értékadó utasítás
- 2. Üres utasítás
- 3. Ugró utasítás
- 4. Elágaztató utasítások
- 5. Ciklusszervező utasítások
- 6. Hívó utasítás
- 7. Vezérlésátadó utasítások
- 8. I/O utasítások
- 9. Egyéb utasítások

A felsorolt utasítástípusok közül a 3.-7. a vezérlési szerkezetet megvalósító utasítások. Nem minden nyelv tartalmazza minden egyik utasítástípust, a C ben például nincsenek egyéb utasítások.

4.1 Értékadó utasítás

Beállítja egy vagy több változó értékét a program futása közben bármikor.

4.2. Üres utasítás

Az eljárásorientált nyelvekre jellemző, hogy tartalmaznak üres utasítást, és vannak olyan nyelvek ahol a szintaktika egyenesen megköveteli a használatukat. Az üres utasítást jelölő szó két utasításjel, amelyek között nincs semmi, vagy egyes nyelveknél van külön szava.

4.3. Ugró utasítás

Az ugró utasításokkal a kód egy részéről a kód egy másik részére ugorhatunk, és az ott lévő utasítás kapja meg a vezérlést. A korai programozó nyelvek idejében a GOTO kifejezés jelölte az ugró utasítást, és alkalmazásuk nélkül nem volt kivitelezhető a programozás.(A mai nyelvek is tartalmazzák a GOTO utasítást, de ritkán szükséges a használata, és ismeret nélküli használata összekuszálhatja a kódot.)

4.4 Elágaztató utasítások

4.4.1 Kétirányú elágaztató utasítás (feltételes utasítás)

Ezzel az utasítástípussal kétféle feladat közül választhatunk, attól függően, hogy az utasítás melyik feltétele teljesül. Alakja a következő: *IF feltétel THEN tevékenység [ELSE tevékenység]* .(A C nyelvben a feltétel zárójelek között áll, és a THEN szó nem szerepel, de ugyanúgy működik az utasítás.) Az alakból látszik, hogy az "else" rész opcionális. Fontos megjegyezni, hogy a tevékenység részt nem minden nyelv ugyanúgy kezeli. Összetett utasítások esetében van amelyik megköveteli az utasítás zárójeleket, melyek neve BEGIN és END. Az ezek között megadott utasításokat formálisan egy utasításként kezelik az ezen szintaktikát megkövetelő nyelvek. Más nyelvek megengedik az összetett utasítások tetszőleges használatát, megint más nyelvek pedig egyetlen utasítást, vagy egy utasítás blokkot fogadnak el.

Ha az elágaztató utasításban szerepel az opcionális ELSE tag, akkor hosszú utasításnak nevezzük, ha pedig nem szerepel, akkor rövidnek.

Az utasítás maga a következőképpen fordul le. A program megállapítja, hogy igaz-e az IF feltétele, ha igaz, akkor a THEN részt veszi figyelembe, az abban a részben szereplő utasításokat követi, majd azok

végeztével visszalép az IF részre. Ezt a folyamatot addig végzi el, amíg a feltétel hamis nem lesz. Ha a feltétel hamis, akkor az ELSE részre lép, és végzi el, majd halad tovább a programkód következő részére. Abban az esetben, ha nincs ELSE ág, akkor az egy üres utasítás.

Az IF utasítások egymásba rendezhetőek, egy IF utasítás ágaiban szerepelhetnek más IF utasítások. Ezeket "csatlakozó ELSE"-nek nevezzük, az utasítás nehezen olvashatóvá válhat, és nehéz lehet eldönteneni hogy melyik ágról van szó adott esetben. Ennek a problémája kiküszöböltető, ha például minden hosszú IF-et használunk, és abban az ágban, amit nem szándékozunk használni, üres utasítást használunk.

4.4.2. Többirányú elágaztató utasítás

A többirányú elágaztató utasítás esetében egymást kölcsönösen kizárt lehetőségek közül választunk, egy kifejezés értékei szerint. C nyelven a szintaxis és a szemantika a következő:

```
SWITCH (kifejezés) {  
CASE egész_konstans_kifejezés : [ tevékenység ]  
[ CASE egész_konstans_kifejezés : [tevékenység] ] ...  
[ DEFAULT: tevékenység ]  
};
```

A kifejezésnek típusának egy egész számnak, vagy egész számra konvertálható típusnak kell lennie. Több eset ág van, ezek nem egyezhetnek meg (Kölcsönösen kizárt lehetőségek), az ágakban szereplő tevékenységek pedig végrehajtható utasítások. A DEFAULT ág az alapértelmezett ág, ez akkor fut le ha egyik CASE(eset) sem következik be. A DEFAULT ág opcionális, de ha egyik eset sem következik be, akkor a SWITCH üres utasítás lesz. Az utasítás első lépésében kiértékelődik a kifejezés, majd ez az érték sorban összehasonlításra kerül a CASE ágakban szereplő értékekkel. Ha megegyeznek, akkor CASE-ben szereplő tevékenység végrehajtásra kerül.

4.5. Ciklusszervező utasítások

A ciklusszervező utasítások lehetővé teszik egy tevékenység akár végtelen sokszori megismétlését. Általában három részből áll, ezek a fej, a mag, és a vég. Az, hogy mi alapján ismételjük meg a tevékenységet, a fejben fogalmazódik meg. A mag tartalmazza magát a tevékenységet, amit el szeretnénk végezni. Sokféle ciklusfajta van: feltételes, előírt lépésszámú, felsorolásos, végtelen, és összetett ciklus. Van két különálló eset is, az üres ciklus és a végtelen ciklus. Az első esetében a ciklus egyszer sem fut le, a másodikéban pedig sohasem áll le.

4.6. Ciklusszervező utasítások az egyes nyelvekben

C-beli ciklusok:

Kezdőfeltételes ciklus:

WHILE(feltétel) végrehajtható_utasítás

Ha a feltétel értéke nullától különbözik, akkor ismétlésre kerül a ciklus. A feltétel típusa integrális.

Végfeltételes ciklus:

DO végrehajtható_utasítás WHILE(feltétel);

Ha a feltétel értéke nullától különbözik, akkor a ciklus ismétlésre kerül.

FOR-ciklus:

FOR([kifejezés1]; [kifejezés2]; [kifejezés3]) végrehajtható_utasítás

A kifejezés1-ben inicializáló kifejezés szerepel, ez lesz a kezdeti érték a ciklusban. Az inicializációt el lehet végezni itt, vagy a ciklus előtt. A kifejezés2-ben határozzuk meg, hogy mertől meddig fussen a ciklus. Ha nincs kifejezés2, akkor végtelen ciklussá válik, mert nincs ami meghatározza meddig ismétlődjön a ciklus. A kifejezés3 pedig minden ciklusban kiértékelésre kerül. Ha végtelen ciklusunk van, abból a BREAK vezérlő utasítással lehet kilépni.

4.7. Vezérlő utasítások a C-ben

Három vezérlőutasítás van:

- CONTINUE;
- BREAK;
- RETURN [kifejezés];

A CONTINUE; a ciklus magjában szerepelhet, hatására a magban a maradék utasításokat nem végzi el, hanem az ismétlődés feltételeit újra megvizsgálja, és ettől függően vagy újabb cikluslépésbe kezd, vagy kilép a ciklusból és a program halad a következő sorral.

A BREAK; a ciklus magjában szerepelhet, vagy többszörösen elágazó utasításokban. Hatására a ciklus szabályosan befejeződik.

A RETURN; függvényekben szerepel, befejezteti azt, majd a vezérlés visszakerül a hívóhoz.

Nem lokális ugrások

Nem lokális ugrásokat a `setjmp()` és a `longjmp()` függvények használatával tudunk kivitelezni. Ez az ugrás a 4.3.-ban leírt GOTO. A `setjmp()`-al egy másik függvény előre meghatározott helyére állíthatunk be egy pontot, ahol a majdani `longjmp()` végrehajtása után a vezérlés kerülni fog.

5. A PROGRAMOK SZERKEZETE

Az eljárásorientált nyelvekben a forráskód több, egymástól független részből, programegységekből állhat. Ezek a programegységek az alprogramok, blokkok, csomagok, és taszkok.

A nyelvek nem ugyanúgy kezelik a programegységeket, ezért különböző nyelvenél felmerülnek kérdések.

A programot egyben, vagy akár részenként is lehet kezelní? Egyes nyelveken megírt programoknál a szöveg részenként fordítható, ezek fizikailag is önálló részek, nincs szerkezetileg mélységük. Vannak nyelvek, ahol nem lehet részenként fordítani a szöveget, csak egyben, ilyenkor a szövegnek felépítésének van mélysége, a programegységek fizikailag nem önállóak. Más nyelvenél pedig az előző kettő kombinációja áll fenn, a szöveg fizikailag önálló részeiből állhat, és a részeknek lehet struktúrális mélysége.

Kérdés még, hogy ha a részek külön fordíthatók, mi alkothat egy önálló fordítási egységet? Milyen programegységek léteznek? Milyen a programegységek viszonya? A programegységek hogyan kommunikálnak egymással? A következőkben ezekre a kérdésekre adunk választ.

5.1. Alprogramok

Az alprogramok nagyon fontos absztraktiós eszközök, segítségükkel általános formában tudunk meghatározni feladatakat, előre nem tudjuk, hogy pontosan mi fog szerepelni az alprogramban, de van egy forma amit meghatároz, ezt megadva az alprogramnak a program során később, többször is felhasználható, tehát az alprogramok az újrafelhasználás eszközei. Használatukhoz az alprogramot az adott helyen meg kell hívni, hivatkozni kell rá, az általános paraméterek helyére formális paraméterek kerülnek.

Az alprogramok formálisan három részből épülnek fel, ezek a fej(specifikáció), a törzs(implementáció), és a vég. Négy komponense van, név, formális paraméter lista, törzs, és környezet.

A név és a formális paraméter lista a fej részben helyezkedik el, az alprogramot hívni ennek a kettőnek a segítségével -és egyes nyelveken egy hívószóval, ami általában a CALL- lehet, a következőképpen:

eljárások esetében:

[alapszó] eljárásnév(aktuális_paraméter_lista)

vagy függvények esetében:

függvénynév(aktuális_paraméter_lista)

Azért az aktuális paraméter listával hívjuk meg, mert a formális paraméter lista csak az általános eszközöket nevezi meg, az aktuális pedig a programban már deklarált eszközöket használja. Látható, hogy a paraméter lista kerek zárójelek között áll. Ha a zárójelek között nem áll semmi, akkor az egy üres, paraméter nélküli alprogram.

Az alprogram törzsében szerepelhetnek deklarációs és végrehajtható utasítások. Az itt deklarált eszközököt lokális eszközöknek nevezünk, neveiket lokális neveknek, ezek az alprogramon kívül rejte maradnak. Hasonló módon a nem egy alprogramban deklarált eszközöket globális eszközöknek nevezünk, ezekre szabadon lehet hivatkozni bárhol.

Az alprogram környezete a globális eszközök együttese.

Az alprogramokból kétféle létezik, az eljárás és a függvény.

Az eljárás egy tevékenységet végez el, és eredményt ad vissza. Ez a tevékenység a törzsben van megírva, ezt elvégezve megváltoztathatja a paramétereinek értékét, vagy a környezetét.

A függvény egy értéket határoz meg, és adja vissza eredményként. Az eredmény típusa tetszőleges, de előre, a név részében meghatározott. Ha a függvény megváltoztatja a paramétereit vagy a környezetét, azt mellékhatásnak nevezünk. Viszont ha ez a mellékhatás nem szándékos, akkor ez egy negatív mellékhatás.

5.2. Hívási lánc, rekurzió

Egy programegység bármikor meghívhat egy másik programegységet, az egy újabb programegységet, és így tovább. Így kialakul egy hívási lánc. A hívási lánc első tagja minden fiprogram. A hívási lánc minden tagja aktív, de csak a legutoljára meghívott programegység működik. Szabályos esetben minden az utoljára meghívott programegység fejezi be legeliször a működését, és a vezérlés visszatér az it meghívó programegységebe. A hívási lánc futás közben dinamikusan épül föl és bomlik le. Azt a szituációt, amikor egy aktív alprogramot hívunk meg, rekurziónak nevezünk.

5.4. Paraméterkiértékelés

Egy alprogram hívásakor paraméterkiértékelés történik. A kiértékeléskor a fej formális, és az alprogram aktuális paraméterei egymáshoz rendelődnek.

5.5. Paraméterátadás

A paraméterátadás az alprogramok és más programegységek közötti kommunikáció egy formája. A paraméterátadásnál minden van egy hívó, ez tetszőleges programegység, és egy hívott, amelyik minden alprogram. Kérdés, hogy melyik irányban és milyen információ mozog. A nyelvek a következő paraméterátadási módokat ismerik: – érték szerinti – cím szerinti – eredmény szerinti – érték-eredmény szerinti – név szerinti – szöveg szerinti Az érték szerinti paraméterátadás esetén a formális paramétereknek van címkomponensük a hívott alprogram területén. Az aktuális paraméterek rendelkeznie kell értékkomponenssel a hívó

oldalon. Ez az érték meghatározódik a paraméterkiértékelés folyamán, majd átkerül a hívott alprogram területén lefoglalt címkomponensre. A formális paraméter kap egy kezdíértéket, és az alprogram ezzel az értékkel dolgozik a saját területén. Az információáramlás egyirányú, a hívótól a hívott felé irányul. A hívott alprogram semmit sem tud a hívóról, a saját területén dolgozik. Mindig van egy értékmásolás, és ez az érték

5.6. A blokk

Nem minden nyelvben léteznek blokkok, szerepük az egyes részek hatásköreinek meghatározása. A blokkok programegységek alegységei, kizárolag ott helyezkedhetnek el. Van egy feje, törzse, és vége, szerepelhetnek a törzsben deklarációs és végrehajtandó utasítások, típustól függően a hozzáartozó részben. A blokkoknak nincsenek paramétereik, nevük sem minden nyelvben.

5.6. Hatáskör

A hatáskör -másnéven láthatóság- alatt a program azon részét értjük, ahol az aktuális név ugyanarra a programozási eszközre terjed ki, tehát olyan programrészek, amelyeknek a tulajdonságai megegyeznek. Amikor egy név hatáskörét megállapítjuk, hatáskörkezelést végzünk.

Ennek két formája van, statikus és dinamikus.

A statikus hatáskörkezelés fordítási időben történik, a fordítóprogram által. A programegységen definiált név a programegység lokális neve. A nem a programegységen definiált, de onnan látható nevek globális nevek. Ez a lokális/globális mivolta egy névnek relatív, attól függ hogy melyik programegységet nézzük. A hatáskör mindig befelé szűkül, kifelé nem terjed. Hatáskör kezelés közben a fordítóprogram ha egy szabad(nem a programegységen definiált, de ott meghivatkozott) nevet talál, akkor megnézi hogy az adott programegység lokális neve e, ha nem, akkor a programegységtől egygyel kintebb lévő programegységnél végzi el ezt a vizsgálatot. Ha így, teljesen a program legkülső részéhez ér, akkor két dolog történhet:

- Abban az esetben, ha a nyelvben minden nevet deklarálni kell, de mi ezt mégsem tettük meg, compile-time errort kapunk.
- Azoknál a nyelveknél, amelyek ismerik az automata deklarálást, a megfelelő szabályok szerint deklárnival fogja a szabad változót, így a legkülső szint lokális neve lesz.

A dinamikus hatáskörkezelés futásidőben zajlik, a hívási láncot használja fel. Itt a szabad nevet a hivási lánc segítségével keresi vissza, amíg meg nem találja a hivatkozás helyét, vagy vissza nem ér a lánc legelejére. Ilyenkor az előző esettel megegyezően, nyelvtől függően runtime-errorröt kapunk, vagy automatikusan deklárája a futtató rendszer a változót.

5.8. Fordítási egység

Az eljárásorientált nyelvek programjai fordítási egységekből állnak. Ezek az egységek egymástól elkülöníthető, külön lefordítható részek. Nyelvenként az alakjuk nagyon eltérő lehet. Gyakran hatásköri és élettartam definiáló egységek ezek.

13. INPUT/OUTPUT

Az input/output sokféle lehet. Az az eszköszrendszer, amely a perifériák közötti kommunikálást valósítja meg.

az I/O középpontjában egy állomány áll, ez egy absztrakt állományfogalomnak felel meg. Egy logikai állomány egy programozási eszköz, amelynek van neve, állományjellemzői. És vannak fizikai állományok, ezeknek az adatai azok, amelyek megjelennek a periférián.

Egy állományt osztályhatunk funkciója szerint:

- input állomány: a feldolgozás előtt már léteznie kell, és a feldolgozás sorából változatlan marad, csak olvasni lehet belőle
- output állomány: a feldolgozás előtt nem létezik, a feldolgozás hozza létre, csak írni lehet bele
- input-output állomány: általában létezik a feldolgozás előtt és étezik a feldolgozás után is, de a tartalma megváltozik, olvasható és írható

Az I/O-n megjelenő adatoknak kétféle átviteli módja van, a folyamatos és a bináris módú. A folyamatos mód esetén konverzió van, a binárisnál nincs.

Folyamatos mód:

A memóriában és a periférián a megjelenése az adatoknak eltérő. A periférián lévő adatokat egy karaktersorozatként kezelik a nyelvek, a memóriában pedig bitsorozatként. Az adatátvitelhez 3 eszközrendszeret használnak a nyelvek: (PICI könyből:)

- formátumos módú adatátvitel: minden egyes egyedi adathoz a formátumok segítségével explicit módon meg kell adni a kezelendi karakterek darabszámát és a típust
- szerkesztett módú adatátvitel: minden egyes egyedi adathoz meg kell adni egy maszkot, amely szerkeszti és átviendi karakterekből áll. A maszk elemeinek száma határozza meg a kezelendi karakterek darabszámát, a szerkeszti karakterek megadják, hogy az adott pozícióon milyen kategóriájú karakternek kell megjennie, a többi karakter változtatás nélkül átvitelre kerül.
- listázott módú adatátvitel: itt a folytonos karaktersorozatban magában vannak a tördelést végző speciális karakterek, amelyek az egyedi adatokat elhatárolják egymástól, a típusra nézve pedig nincs explicit módon megadott információ.

Bináris adatátvitel:

Az adatok a memóriában és a periférián megegyeznek megjelenésben. Háttértárakkal kommunikálnak, az átvitelt a rekord alapozza meg.

(PICI könyv:)

- Deklaráció: A logikai állományt mindenkor deklarálni kell az adott nyelv szabályainak megfelelően. El kell látni a megfelelő névvel és attribútumokkal. minden nyelv definiálja, hogy milyen állományfogalommal dolgozik. Egyes nyelvek azt mondják, hogy a funkció is attribútum, tehát a deklarációtól eldil.
- Összerendelés: Ennek során a logikai állománynak megfeleltetünk egy fizikai állományt. Ezt a megfeleltetés vagy a program szövegében, nyelvi eszközzel történik (a fizikai állomány csak itt jelenik meg), vagy a program szövegén kívül, operációs rendszer szinten végezzük azt el. Innentől kezdve csak a logikai állománynévvel dolgozunk, de a tevékenység mindenkor a mögötte álló fizikai állományra vonatkozik.
- Állomány megnyitása: Egy állománnyal csak akkor tudunk dolgozni, ha megnyitottuk. Megnyitáskor operációs rendszer rutinok futnak le, ellenirizve, hogy a logikai állomány attribútumai és a fizikai állomány jellemzii megfelelnek-e egymásnak. Egy állomány funkciója a megnyitásnál is eldilhet bizonyos nyelvekben (pl. „inputra nyitunk”). Ekkor a program futása folyamán ugyanazt az állományt más-más funkcióra is megnyithatjuk.

- **Feldolgozás:** Ha az állományt megnyitottuk, akkor abba írhatunk, vagy olvashatunk belőle. Az olvasást realizáló eszköznél meg kell adni a logikai állomány nevét és folyamatos módú adatátvitelnél egy tetszéges változólistát. Ekkor a felsorolt változók értékkomponensüket az adott állományból kapják meg. Formátumos átvitelnél minden változóhoz egy formátumot, szerkesztettnél egy maszkot meg kell adni. Listázott átvitelnél a konverziót a változók típusa határozza meg. Bináris átvitelnél általában egy (ritkán több) változó adható meg, melynek rekord típusúnak kell lenni. A kiíró eszközrendszerben a logikai állomány neve mellett egy kifejezéslistát kell szerepelteni. A kifejezések kiértékelődnek, és ezen értékek kiírásra kerülnek. A kifejezésekhez itt is egyenként szükségesek a formátumok, illetve a maszkok. Listázottnál a kifejezés típusa a meghatározó. Bináris átvitelnél a kifejezésnek rekordot kell szolgáltatnia.
- **Lezáras:** A lezáras ismét operációs rendszer rutinokat aktivizál. Azért nagyon fontos, mert a könyvtárak információinak aktualizálása ilyenkor történik meg. Output és input-output állományokat le kell zární, input állományokat pedig illik lezárni. A lezáras megszünteti a kapcsolatot a logikai állomány és a fizikai állomány között. A nyelvek általában azt mondják, hogy a program szabályos befejeződésekor az összes megnyitott állomány automatikusan lezáródik.

10.2 Programozás bevezetés

[KERNIGHANRITCHIE]

Megoldás videó: <https://youtu.be/zmfT9miB-jY>

3. fejezet: Vezérlési szerkezetek

3.1. Utasítások és blokkok

A kifejezésekben akkor válnak utasítások, ha egy pontosvesszővel zártuk le őket(;). Ennek a pontosvesszőnek a neve az utasításlezáró jel.

3.2. Az if-else utasítás

Az if-else, magyarul ha-különben utasítással több lehetőség között választhatunk. Az utasítás alakja: if (kifejezés) 1.utasítás else 2.utasítás Kiértékelődik a az if-et követő kifejezés, és ha a kifejezés igaz, azaz nem nulla értéket ad vissza, akkor az 1.utasítás kerül végrehajtásra. Ha nem igaz a kifejezés, akkor az else ágban szereplő 2.utasítás. Az else rész ugyanekkor opcionális, ha nincs else ág de a kifejezés hamis, akkor ez a kifejezés üres, nem hajt végre semmit.

Az if utasítások egymásba ágazhatóak, viszont ilyenkor könnyen zavaros lehet, hogy melyik ág mikor következik be. Az ilyen egymásba ágazott if-else utasításokat éppen ezért "csellengő else"-nek nevezik. Talán a legegyszerűbb mód ennek a nehézségnak a kiküszöbölésére az, ha az utasításokat körülzárt jelek közé tesszük. Ha ilyen többféléles lehetőségek közül szeretnénk választani, a többszörösen egymásba ágazott if-ek helyett elegánsabb megoldás lehet a switch utasítást használni.

3.4. A switch utasítás

```
SWITCH (kifejezés) {  
CASE egész_konstans_kifejezés : [ tevékenység ]  
[ CASE egész_konstans_kifejezés : [tevékenység ] ]...  
[ DEFAULT: tevékenység ]  
};
```

A switchel több, egymást kizáró lehetőségek közül tudunk egyértelműen választani a kifejezésünk szerint. A case-eket a neveik, majd egy kettőspont, és azután a tevékenység követ. Fontos a tevékenyég után egy break; utasítást adni, ami lezárja a switchet. A case ágakon kívül van egy opcionális, de fontos default; utasítás, amelyben szereplő tevékenységre akkor kerül sor, ha egyik case ágra sem teljesül a kifejezésben megfogalmazott feltétel.

3.5. A while és a for utasítás

A két ciklus szintaktikája különböző ugyan, de szemantikája megegyezik. Amíg a ciklusban szereplő kifejezés igaz, tehát nullától különböző, addig a ciklus újra meg újra le fog futni. Igaz, hogy megegyezik a szemantikájuk, de vannak esetek amikor egyik ciklus kézenfekvőbb mint a másik. Ha például nincs inicializált kezdőértékünk, vagy léptető értékünk, a while használata talán előnyösebb.

A for ciklus elején megadjuk a kezdőértéket, a feltételt, és a léptetést.

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

Pontosvesszőkkal vannak egymástól elválasztva, és a részek elhagyhatóak, de a pontosvesszőt akkor is ki kell tenni.

3.6. A do-while utasítás

A for és a while ciklusok esetében lehetőség van arra, hogy a ciklus befejeződését aktiváló kitételt nem a ciklus végén vizsgálja, a do-while ezt a vizsgálatot viszont a ciklustörzs végén végzi el, ez garantálja hogy a ciklus legalább egyszer lefusson.

```
do  
utasítás  
while (kifejezés);
```

A do után szereplő utasítás végrehajtásra kerül, majd a while ban szereplő kifejezés kiértékelődik, és a megszokott módon, amíg a kifejezés értéke nem nulla, addig újra végrehajtásra kerül a ciklus. Ezt a ciklusfajtát sokkal ritkábban szokták használni mint a for és a while ciklust.

3.7. A break utasítás

A switch-nél már volt szó a break utasításról, amivel ki tudunk lépni a ciklusból. Hasonlóan a for a while és a do ciklusokból is azelőtt ki lehet így lépni, hogy a ciklus végére érnénk.

3.8. A continue utasítás

A continue utasítás az ismétlődő ciklusokban szerepelhet(while, for, do), hatására a for ciklus esetében az újraértékelés rész ismétlődik meg, a while és do-while esetében pedig a continue után újra kiértékelődik a kifejezés. Gyakorlati haszna, hogy kihagyhatunk bonyolult részeit egy feladatnak, amelyekre nincs szükségünk.

3.9. A goto utasítás; címkek

A goto utasítással ugrást hajthatunk végre, és egy másik programrészre léphetünk. A goto-ra nagyon ritkán van szükség, de megvan a létjogosultsága. Ha egy többszörösen egymásba ágazott for ciklusból szeretnénk teljesen kilépni például, gondolhatunk a break utasítás használatára, de nem feltétlen fogunk tudni vele kilépni az egész ciklusból, csak egy ágából. Ilyenkor jól jöhét a goto utasítás. Alakja a következő:

goto példa; példa:

A goto a példa nevű címkére mutat, a címke alakja pedig megegyezik a változónevekkel, és egy kettőspont követi őket.

10.3 Programozás

[BMECPP]

Hatodik fejezet, Operátorok és túlterhelésük.

A programozásban az operátorok többféle jelntőséggel bírnak, és az argumentumoktól függően különböző, meghatározott folyamat megy végbe. A "+"-jel, az összeadás például használható integerek, de lebegőpontos változók, vagy akár mátrixok, komplex számok esetén is. Ahhoz, hogy komplex számokra alkalmazható legyen, a C++ nyelv lehetőséget nyújt, erre való az operátorok túlterhelése.

6.1. Az operátorokról általában

Az operátorok tulajdonságait a Pici könyben már részleteztük, ezt újra nem kell megtárgyalnunk. A C++-ban a C-hez képest több operátor áll rendelkezésünkre, sőt még saját operátorokat is megalkothatunk. Ilyen extra operátor a C++-ban a (::) és a (.* és ->*), a hatáskör és a pointer tag operátorok.

6.2. Függvényszintaxis és túlterhelés

A C++ nyelv támogatja az operátorok függvényszintaxissal történő használatát. A függvények és az operátorok közötti különbség a kiértékelés szabályában tér el. Példa a BME-s Levendovszky könyv 94-94. oldalairól:

```
++i // hagyományos, operátor-írásmód
operator++ (i); //C++: a függvényszintaxis is megengedett

//masik pelda

c = a + b; // hagyományos, operátor-írásmód
c = operator+(a, b); // az = operátor-, a + függvényszintaxissal
operator=(c, operator+(a, b)); // függvényszintaxis
```

Az ilyen féle függvényszintaxissal definiált operátorokat a saját magunk által kreált típusokra tervezzük, a beépített típusokkal ez nem működne, ugyanis ennek a lehetőségnek nem célja hogy a már létező, bevált operátorokat megváltoztassuk. A függvényt érdemes az osztályunkon belül implicit,tagfüggvényként definiálni. Tehát létrehozhatunk egy függvényszintaxist a +operatorra, de annak(legalább egyik) argumentumaként egy általunk megírt osztálybeli tagot kell kapjon.

Part III

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

Chapter 11

Helló, Arroway!

11.1 OO szemlélet

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

Source : [PolarGenerator.java](#) [polargenerator.cpp](#)

A feladat egyszer már szóba került a könyv írása során, most Javaban és c++ -ban is megírjuk a kódot.

Egy polárgenerátor(PolarGenerator) osztályt írunk meg. Lesz egy logikai változónk, a nincsTarolt, ami mindenkor megmondja hogy éppen nincs, vagy van e tárolt értékünk, és egy double értékünk, a tarolt, amiben mindenkor legyen tároljuk az értéket, ha van. A program legfontosabb része a kovetkezo () függvény, amely megnézi, hogy van-e már tárolt értékünk (nincsTarolt) vagy nincs, ha nincs, akkor lefuttatja az algoritmust, ha nincs, akkor felhasználja a tarolt értéket. Mindkét eset végén fontos, hogy a tarolt érték felhasználása után ellenőrizze a nincsTarolt-at,

A programmal egy igazi véletlenszámgenerátort készítünk. Ehhez felhasználjuk a Math.random() függvényt, ami egy pseudo random számot fog generálni, de nem igazit. Elvégezve pár egyszerű műveletet megkapjuk az értékeinket.

Ezután a program main részében példányosítjuk a PolarGenerator osztályt, és egy for ciklussal kiíratunk 10 random számot.

A Sun JDK-jában láthatjuk hogy ugyanez a módszer van megírva, ez a nextGaussian() függvény.

```

synchronized public double nextGaussian() {
    // See Knuth, ACP, Section 3.4.1 Algorithm C.
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1; // between -1 and 1
            v2 = 2 * nextDouble() - 1; // between -1 and 1
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}

```

Figure 11.1: nextGaussian() a jdk-ban

11.2 Homokozó

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

Források:[BinaryTree.java](#) [Main.java](#) Ahogy a feladat is mondja, és ahogy az olvasónaplóban leírtuk, Java nyelvben nem használunk mutatókat, mert minden egy referencia. A megadott c++ os binfa forrást átalakítjuk, töröljük a mutatókat, referenciaikat, és Java szintaxist használunk.

C++-os minta, mutatók, gyökerek vannak(*, &)

```

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

```

Java-s minta:

```
if (newItem == '1') {
    if (current.right() == null) {
        current.setRight(new Node('1'));
        setCurrent(root);
    } else {
        setCurrent(current.right());
    }
}
```

Ezután letöljük az Apache Tomcat-et a weboldaláról, aminek segítségével létrehozunk egy szervletet, ami a Binfát fogja használni a majdani böngészők kérésünkhöz, a címsorba fogjuk a bináris számsorozatot írni az eddigी infile helyett.

A tomcat webapps mappájában készítettem egy Binfa nevű mappát, aminek a forrásaként megadtam a 2 source kódomat, ezt fogja majd futtatni a servlet.A Main forrásban írjuk meg a servletkészítéshez szükséges dolgokat, és a doGet ben leírjuk hogy hogyan kell megadni a számsort és hogy hogyan fogja ezt kezelni(a BinaryTree forráskód értékeit használva).

```
public void doGet(HttpServletRequest request, ←
                    HttpServletResponse response) throws IOException, ←
                    ServletException {
    data = request.getParameter("data");
    tree = new BinaryTree();
    response.setContentType("text/html; charset=UTF-8");
    for (int i = 0; i < data.length(); i++) {
        tree.addItem(data.charAt(i));
    }
    PrintWriter out = response.getWriter();
    try {
        out.println("<!DOCTYPE html>");
        out.println("<html><head>");
        out.println("<meta http-equiv='Content-Type' content ='text/ ←
                    html; charset= UTF-8'>");
        out.println("<body>");
        tree.writeOut(tree.getRoot(), out);
        out.println("<br></br>");
        out.println("<p>Elemszam atlaga " + tree.getElemszamAtlag() + " ←
                    </p>");
        out.println("<p>Atlag " + tree.getAtlag() + "</p>");
        out.println("<p>Melyseg " + tree.getMelyseg() + "</p>");
        out.println("<p>Szoras " + tree.getSzoras() + "</p>");
        out.println("</body>");
        out.println("</html>");
```

Ehhez szükség van még a Binfa mappában elhelyezni egy xml fájlt, ebben megadjuk a servlet nevét, és azt hogy egy get kérést használhatunk.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun. ←
    com/xml/ns/javaee/web-app_3_0.xsd">

<!-- To save as <CATALINA_HOME>\webapps\helloservlet\WEB-INF\web.xml ←
    -->

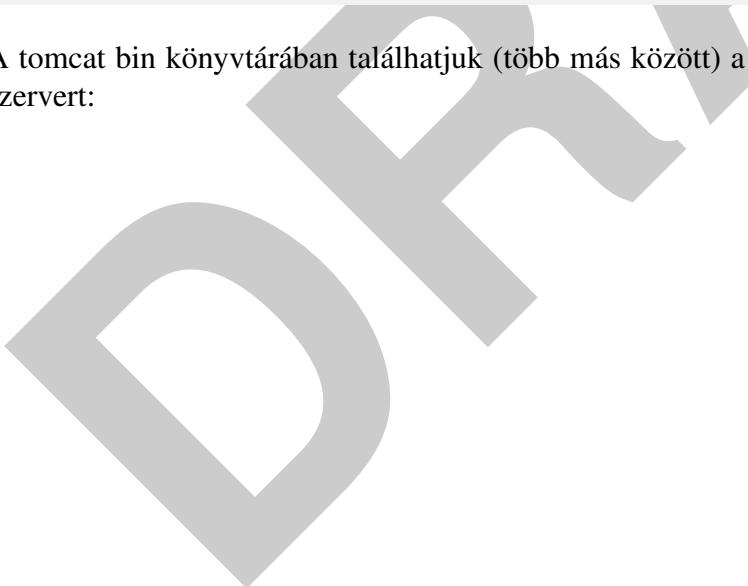
<servlet>
  <servlet-name>Binfá</servlet-name>
  <servlet-class>com.company.Main</servlet-class>
</servlet>

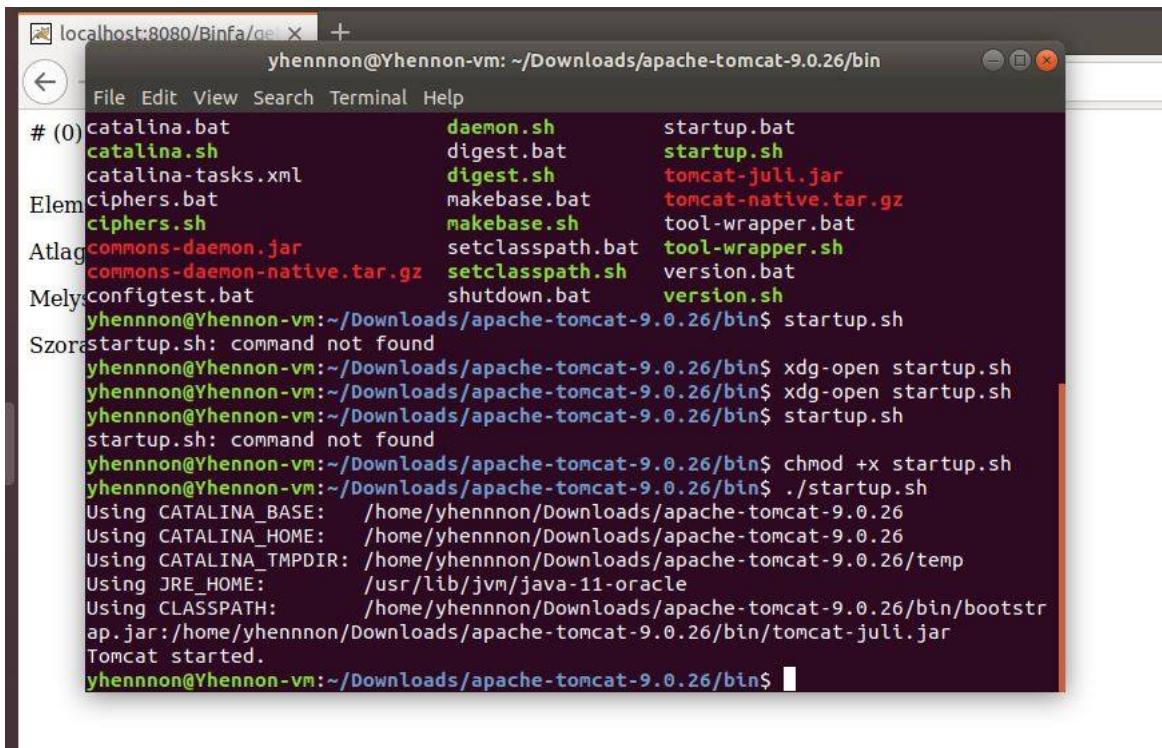
<!-- Note: All <servlet> elements MUST be grouped together and
      placed IN FRONT of the <servlet-mapping> elements -->

<servlet-mapping>
  <servlet-name>Binfá</servlet-name>
  <url-pattern>/get</url-pattern>
</servlet-mapping>

</web-app>
```

A tomcat bin könyvtárában találhatjuk (több más között) a startup.sh fájlt, amit futtatunk hogy elindítsuk szervert:





```
localhost:8080/Binfa/get X +  
yhennnon@Yhennnon-vm: ~/Downloads/apache-tomcat-9.0.26/bin  
File Edit View Search Terminal Help  
# (0)catalina.bat           daemon.sh      startup.bat  
# (0)catalina.sh            digest.bat    startup.sh  
catalina-tasks.xml        digest.sh     tomcat-juli.jar  
Elemciphers.bat           makebase.bat  tomcat-native.tar.gz  
ciphers.sh                makebase.sh   tool-wrapper.bat  
Atlagcommons-daemon.jar   setclasspath.sh  tool-wrapper.sh  
commons-daemon-native.tar.gz setclasspath.sh version.bat  
Melyconfigtest.bat        shutdown.bat  version.sh  
yhennnon@Yhennnon-vm:~/Downloads/apache-tomcat-9.0.26/bin$ startup.sh  
Szorastartup.sh: command not found  
yhennnon@Yhennnon-vm:~/Downloads/apache-tomcat-9.0.26/bin$ xdg-open startup.sh  
yhennnon@Yhennnon-vm:~/Downloads/apache-tomcat-9.0.26/bin$ xdg-open startup.sh  
yhennnon@Yhennnon-vm:~/Downloads/apache-tomcat-9.0.26/bin$ startup.sh  
startup.sh: command not found  
yhennnon@Yhennnon-vm:~/Downloads/apache-tomcat-9.0.26/bin$ chmod +x startup.sh  
yhennnon@Yhennnon-vm:~/Downloads/apache-tomcat-9.0.26/bin$ ./startup.sh  
Using CATALINA_BASE:  /home/yhennnon/Downloads/apache-tomcat-9.0.26  
Using CATALINA_HOME:   /home/yhennnon/Downloads/apache-tomcat-9.0.26  
Using CATALINA_TMPDIR: /home/yhennnon/Downloads/apache-tomcat-9.0.26/temp  
Using JRE_HOME:        /usr/lib/jvm/java-11-oracle  
Using CLASSPATH:       /home/yhennnon/Downloads/apache-tomcat-9.0.26/bin/bootstrap  
ap.jar:/home/yhennnon/Downloads/apache-tomcat-9.0.26/bin/tomcat-juli.jar  
Tomcat started.  
yhennnon@Yhennnon-vm:~/Downloads/apache-tomcat-9.0.26/bin$
```

Figure 11.2: tomcat servlet indítás

A shutdown.sh-val leállíthatjuk a servletet.

Végül pedig elérhetjük a servletet a localhost:8080 címén, a Binfa funkcióját tudjuk használni, és ahogyan az ehhez tartozó xml ben leírtuk, get kérést tudunk használni a Main-ben leírt módon, így : localhost:8080/Binfa/get?data=1010101001011111100. A data= után írjuk be a bináris sorozatot, és megkapjuk az eredményt.

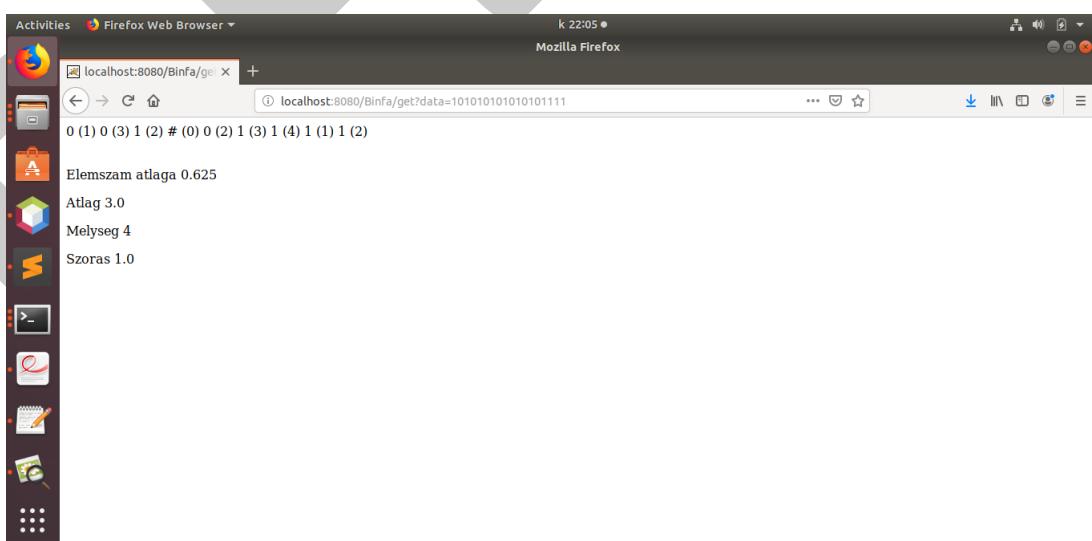


Figure 11.3: az eredmény

11.3 Gagyi

Feladat: Az ismert formális „while ($x \leq t \&& x \geq t \&& t \neq x$);” tesztkérdéstípusra adj a szokásosnál (miszerint x , t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referencia) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x , t értékekkel meg nem! A példát építsd a JDK Integer.java forrására 3, hogy a 128-nál inkluzív objektum példányokat poolozza!

Source: [gagyi.java](#)

A feladat egy felmerült tesztkérdés továbbgondolása, illetve pontos válaszadás a kérdésre. Adott egy `while(x <= t && x >= t && t != x)` ciklus, és két érték pár, egyszer $x = -128$, $t = -128$, majd $x = -129$, $t = -129$. Ami érdekes, az az, hogy -128-as értékekkel a while ciklust használva és az értékeket kiíratva a két érték megjelenik, -129-es értékeket használva pedig végtelen ciklust kapunk.

-128-al:

```
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←
  codes/third_semester_codes/1_arroway$ java gagyi
```

-128

-128

```
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←
  codes/third_semester_codes/1_arroway$
```

-129-el:

```
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←
  codes/third_semester_codes/1_arroway$ javac gagyi.java
```

```
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←
  codes/third_semester_codes/1_arroway$ java gagyi
```

```
^Cyhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←
  codes/third_semester_codes/1_arroway$
```

A különbség megértéséhez az Integer.java forrását kell megvizsgálnunk és egy részét értelmezniink: <https://hg.openjdk.java.net/jdk11/file/1ddf9a99e4ad/src/java.base/share/classes/java/lang/Integer.java?fbclid=IwAR2KBKQc63izndqEB>

A forrásban meg van írva, hogy egy tömböt(cache) készítenek az IntegerCache osztály részeként, ami elősegíti az autoboxolását a -128 és 127 közötti int értékeknek. Egy ranget is beállít az osztályban a primitív értékeknek, amiket autoboxolhat, ez -128 tól egy konfigurálható értéig(high-ig) terjed. Az autoboxolás például az, amikor a compiler egy int típusú primitív értéket egy Integer wrapper-osztálybeli objektummá alakít. A Java compiler a következő esetekben alkalmazza az autoboxolást:

- Amikor a primitív érték egy olyan függvény paramétereként van megadva, ami egy (a primitív value-val) megegyező wrapper-osztálybeli objektumot vár.
- Amikor a primitív érték a vele megegyező wrapper-osztálybeli változóhoz van hozzárendelve.

Ennek a visszafele történő alkalmazása az unboxing.

```
cache = new Integer[(high - low) + 1];
```

```
int j = low;

for(int k = 0; k < cache.length; k++)

    cache[k] = new Integer(j++);

// range [-128, 127] must be interned (JLS7 5.1.7)

assert IntegerCache.high >= 127;
```

Amikor egy új int-et hozunk létre, akkor lefut a következő rész:

```
public static Integer valueOf(int i) {

    if (i >= IntegerCache.low && i <= IntegerCache.high)

        return IntegerCache.cache[i + (-IntegerCache.low)];

    return new Integer(i);

}
```

Megnézi a program, hogy az új int értéke a low és high között van-e, amennyiben igen, az IntegerCache-ben példányosítja, és az int referenciája az lesz, amit a cache az első lefuttatáskor történő inicializáláskor kapott. Amennyiben pedig ezen a rangon kívül adunk értéket az int változónak, akkor az egy új memóriacímre fog referálni.

```
* The cache is initialized on first usage.
..
..
* Returns an {@code Integer} instance representing the specified
* {@code int} value.
..
..
* This method will always cache values in the range -128 to 127,
* inclusive, and may cache other values outside of this range.
```

Így tehát, mivel az első esetben -128at megadva az előre létrehozott rangon belül választott értéket, a while(`x <= t && x >= t && t != x`) ciklusnak nem teljesül az utolsó feltétele mert mind t és mind x ugyanannak az objektumnak a referenciaját hordozzák.

Amikor pedig -129-et használunk, az összes feltétel teljesül, és sosem lépünk ki a while-ból, így végtelen ciklushoz jutunk.

11.4 Yoda

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

Ha valaki ismeri a Star Wars univerzumát, az biztosan ismeri és hallotta már beszélni Yoda mestert. Yoda egy nem megszokott sorrendben alkotja meg a mondatokat, eltér az általunk használt Subject-Verb-Object mondatalkotástól, és Object-Subject-Verb -et használ. Például, magyarul egy helyes mondat úgy hangzik, hogy :’Én szeretek zenét hallgatni’. Ha Yoda mestert utánozva mondánám ugyanezt a mondatot, a Yoda conditions-t használva, akkor pedig így hangzik a mondat: ’Zenét hallgatni Én szeretek’. A feladatunk most pedig az, hogy olyan programot írunk, ami működik, ha tartjuk magunkat a Yoda feltételekhez, és ha nem felelünk meg ennek, akkor NullPointerException -nel álljon le.

Javaban lehet használni egy speciális null értéket különböző célokra, főleg arra, hogy megmondjuk egy változóról, hogy nincs hozzárendelve érték. NullPointerException akkor fordul elő, amikor a program olyan objektum referenciát akar használni, aminek az értéke null.

Source: [yoda.java](#)

Legyen egy egyszerű if() függvényünk, amiben két string értékéről nézzük meg, hogy egyenlőek e. (ezt a stringek equals() metódusával tesszük) Az egyik string értéke legyen null (tehát semmit nem rendelünk hozzá!), a másiké egy általunk választott bármilyen szó, most a tubeOfCheese. Az, hogy mit csináljon a program igaz feltétel esetén, most lényegtelen.(De egy egyszerű kiiratással tesztelhető; ha igaz, írjuk is ki, ha pedig hamis, akkor valami más írunk.) Megszokott módon így néz ki a feltétel:

```
if (wordOne.equals(wordTwo)) {  
    ...  
}
```

Lefuttatva a programunkat ezt a feltételt használva, ezt láthatjuk:

```
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←  
    codes/third_semester_codes/1_arroway$ java Main  
Exception in thread "main" java.lang.NullPointerException  
    at Main.main(yoda.java:9)  
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←  
    codes/third_semester_codes/1_arroway$
```

Nézzük, mi is történik. Tehát, a wordOne stringhez nem rendeltünk értéket, csak a speciális null-t, ami ezt indikálja, a wordTwo -hez pedig a "tubeOfCheese"-t. Mivel a wordOne hoz nem lett semmi hozzárendelve, ezért nincs referenciaja, nincs amit össze lehetne hasonlítani mással, ezért kapunk NullPointerException-t futtatáskor.

Ha fordítva írjuk a feltételt, azaz a második szót hasonlítjuk az elsőhöz, így eleget téve a yoda feltételnek, akkor pedig a kódot lefuttatva ezt látjuk:

```
if (wordTwo.equals(wordOne)) {  
    System.out.println("They are equals.");  
} else {
```

```
System.out.println("sziasztok");  
}
```

Ebben az esetben már a wordTwo létező értékét hasonlítjuk a wordOne -éhoz, amihez nincs hozzárendelve semmi, így nem egyenlőek, és az if függvény hamis ága fog lefutni.

```
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←  
    codes/third_semester_codes/1_arroway$ java Main  
sziasztok  
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/ ←  
    codes/third_semester_codes/1_arroway$
```

11.5 Kódolás from scratch

Feladat: Induljunk ki ebből a tudományos közleményből: <http://crd-legacy.lbl.gov/~dhbailey/dhbpapers/bbp-alg.pdf> és csak ezt tanulmányozva írjuk meg Java nyelven a BBP algoritmus megvalósítását!

Forrás: [PiBBP](#)

A Bailey-Borwein-Plouffe algoritmus a Pí kiszámítására a következő formulát használja, amit 1995-ben fedeztek fel;

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

Figure 11.4: Formula

Ennek a formulának a különlegessége, hogy a Pí bináris és hexadecimális számjegyeit nem csak a tizedestől kezdve, hanem egy tetszőleges kezdőponttól kiszámolhatjuk. Erre akkor jött rá Borwein és Plouffe, amikor a log2 formula kiszámításánál észrevették, hogy bárhonnan kezdhetik az érték kiszámolását.

Chapter 12

Helló, Liskov!

12.1 Liskov helyettesítés sértése

Feladat: Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megsérti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. (számos példa szerepel az elv megsértésére az UDPORG repóban, lásd pl. source/binom/Batfai- Barki/madarak/)

A Liskov helyettesítési elv a következőként szól: Ha S osztály a T osztály leszármazottja, akkor S típust be lehet helyettesíteni minden olyan környezetben(pl. paraméter, változó) ahol T típust kérnek. Erről szól a poliformizmus, ami az olvasónaplóban is szerepel, ezért itt nem részletezném újra.

Ezt az elvet sérti meg a kör-ellipszis probléma, vagy a négyzet-téglalap probléma, és még sok minden más az életben, például ami a feladat leírásában is szerepel, az, hogy a madarak tudnak repülni, de a pingvin hiába madár, mégsem tud.

Én a kör-ellipszis példát fogom most használni a helyettesítési elv hibalehetőségének szemléltetésére. Forrás: [EllipszisKor1.java](#)

A kör egy olyan speciális ellipszis, amelynek minden két tengelye ugyanakkora. Az ellipszisnek tehát lehet két különböző tengelye, a körnek viszont nem. Létrehozunk egy Ellipszis osztályt, aminek van egy nagyobb és egy kisebb tengelye, és egy-egy metódust, amivel állíthatjuk ezeket a tengelyeket. Ebből az osztályból készítünk egy Kör gyermekosztályt, ahol szintén elvégezhetőnek kellene lennie az előző két metódusnak, hisz a Kör is egy ellipszis. A valóságban ez nem így van, ha változtatunk a "kör egyik tengelyén" (vagyis a sugarán, a körnek nincsenek eltérő hosszúságú tengelyei) akkor a "másik tengelye" is ugyanúgy változni fog. Most beállítjuk a kisebb tengelyt 10-re, a nagyobbat 20ra, amit enged a program, de a valóságban nem lehetséges.

```
yhennnon@Yhennon-vm:~/Documents/third_semester/2$ java EllipszisKor1
10.0
20.0
```

12.2 Szülő-gyerek

Feladat: Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ōsön keresztül csak az ōs üzenetei küldhetőek!

Források: [Uzenet.java](#) [Uzenet.cpp](#)

[Ugyancsak az eheti(2.hét) olvasónaplóban is szerepel] Legyen egy osztályunk, amit most Mage-nek hívunk, ami tud bizonyos dolgokat, itt a magic-et. Ebből az osztályból kiterjesztünk egy ElementalMage osztályt, ekkor az ElementalMage a Mage osztály gyermekje, az pedig az ElementalMage ōse lesz.

```
class Mage{  
    ..  
    ..  
}  
class ElementalMage extends Mage{  
    ..  
    ..  
}
```

A gyermek osztály rendelkezni fog az ōsosztály tagjaival, mert belőle lett származtatva, és lehetnek ezen felül saját változói, metódusai is. A feladat rámutatni arra, hogy a gyermek tudja használni az ōse tagjait, de fordítva ez már nem igaz. Ezért lefuttatjuk a gyermekkel a saját metódusát, és az ōsét, majd az ōssel a sajtját, és a gyermekét is. Ekkor viszont hibát kapunk fordításkor, mert olyan metódust akarunk használni az ōben amit nem lát.

```
yhennnon@Yhennon-vm:~/Documents/third_semester/2$ javac Uzenet.java  
Uzenet.java:23: error: cannot find symbol  
        mage.elementalmagic();  
                  ^  
       symbol:   method elementalmagic()  
      location: variable mage of type Mage  
1 error
```

Ha ugyanúgy az előzőeket futtatjuk le, kivétel az utolsót, tehát az ōs nem probálja meg a gyermek metódusát használni, akkor nem lesz ilyen hiba a kódban.

```
yhennnon@Yhennon-vm:~/Documents/third_semester/2$ javac Uzenet.java  
yhennnon@Yhennon-vm:~/Documents/third_semester/2$ java Uzenet
```

12.3 Anti OO

Feladat: A BBP algoritmussal a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10^6 , 10^7 , 10^8 darab jegyet határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket!

Források: [PiBBPBench.java](#) [pi_bbp_bench.c](#) [PiBBPBench.cs](#) [PiBBPBench678.cpp](#)

Az Arrowayes fejezet utolsó feladatában megismertetettük a BBP algoritmust, most azt különböző nyelvekre átírva nézzük meg hogyan teljesítenek, mennyi idő alatt futnak le, és ezeket vetjük össze egymással.

A 10^6 , 10^7 , és a 10^8 hatványat számoljuk ki a PI-nek, ezt minden iterálásnál beállítjuk :

```
for(int d=1000000; d<1000001; ++d) { // 10 hatványait állíthatjuk ←  
    itt
```

Java

A delta segítségével fogjuk megnézni az eltérést, és a currentTimeMillis() -ot használjuk, ami a jelenlegi időt adja vissza milliszekundumokban. A delta-ban eltároljuk az aktuális időt mikor elkezdjük futtatni a számolást, a végeztével pedig átírjuk a végénél lévő időre, amiből kivonjuk a kezdetnél mért időt, így megkapjuk mennyi ideig tartott a programnak hogy lefusson. Azért milliszekundumot mérünk, és osztunk vissza 1000 -el a végén, hogy pontosabb eredményt kapjunk mintha egyből csak másodperct kérnékn.

```
..  
..  
long delta = System.currentTimeMillis();  
..  
..  
delta = System.currentTimeMillis() - delta;  
System.out.println(delta/1000.0);
```

C#, C és C++ változatban ugyanazt csináljuk mint a Javas program esetén, a szintaktika más. C és C++-nál a clock () függvényt használjuk, ami visszaadja hogy mennyi processzor időt igényelt a program, és a végén CLOCKS_PER_SEC -el osztunk.

```
..  
..  
clock_t delta = clock ();  
..  
..  
delta = clock () - delta;  
printf ("%f\n", (double) delta / CLOCKS_PER_SEC);
```

C#-ban pedig a System.DateTime -ot használjuk fel, szintén az előzőek mintájára..

```
..  
..  
System.DateTime kezd = System.DateTime.Now;  
..  
..  
System.TimeSpan delta = System.DateTime.Now.Subtract(kezd);
```

```
System.Console.WriteLine(delta.TotalMilliseconds/1000.0);
```

Végül lefuttatjuk az összes kódot, mind a 3 hatvánnyal, és megnézzük az eredményeket.

Hatvány	Java	C	C++	C#
6	2.539	2.577198	2.52635	
7	28.971	31.934323	29.0169	
8	330.809	323.348317	333.667	

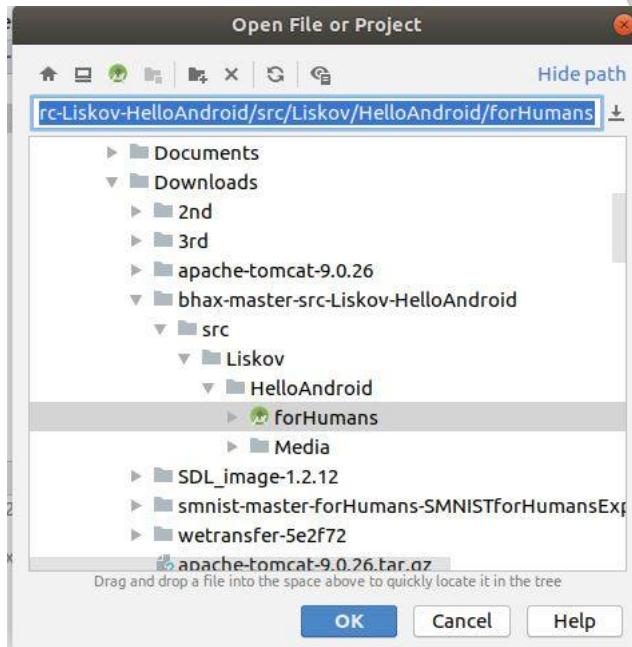
Table 12.1: Összehasonlítás

12.4 Hello, Android! || Tutorom: Racs Tamás

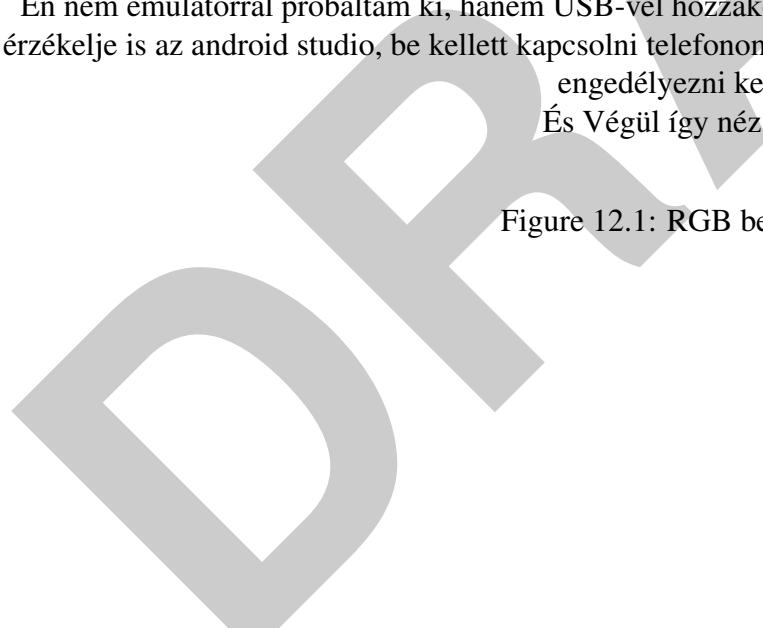
Feladat: Élesszük fel az SMNIST for Humans projektet! Apró módosításokat eszközölj benne, pl. színvilág.

A projekt elindításához legelőször szükségesem volt az Android Studiora, ezt letöltöttem és telepítettem. Indításkor végigmegyünk pár beállításon, többek között ki lehet választani, hogy saját telefonon vagy egy telefon emulátoron akarom kipróbálni az alkalmazást.

Ezután egy üres andorid stúdiós projectet hoztam létre, és a megadott forrást nyitottam meg.



Mostmár csak az van hátra hogy megváltoztassuk a színezést, ezt én kék és zöld között váltakozóra állítottam be, megadtam az rgb kódot hozzá a SMNISTSurfaceView fájlban.



A screenshot of the Android Studio IDE interface. The title bar shows "Activities" and "Android Studio". The status bar indicates the time as "sze 17:38". The main window displays the project structure under "1: Project" and the code editor for "SMNISTSurfaceView.java". The code editor shows the following snippet:

```
private float[] semValueDots = new float[2 * NUM_OF_DIGITS];  
private float[] digitsCoords = new float[2 * NUM_OF_DIGITS];  
  
private int successCnt = 0;  
private boolean armed = true;  
private long armedTime = 0;  
  
int[] bgColor =  
{  
    android.graphics.Color.rgb(red: 0, green: 0, blue: 255);  
    android.graphics.Color.rgb(red: 0, green: 255, blue: 0);  
};  
int bgIdx = 0;  
  
private int decisionTimeDelaySum = 0;  
private int semValueSum = 0;
```

The "Build" tab in the bottom left shows a successful build log:

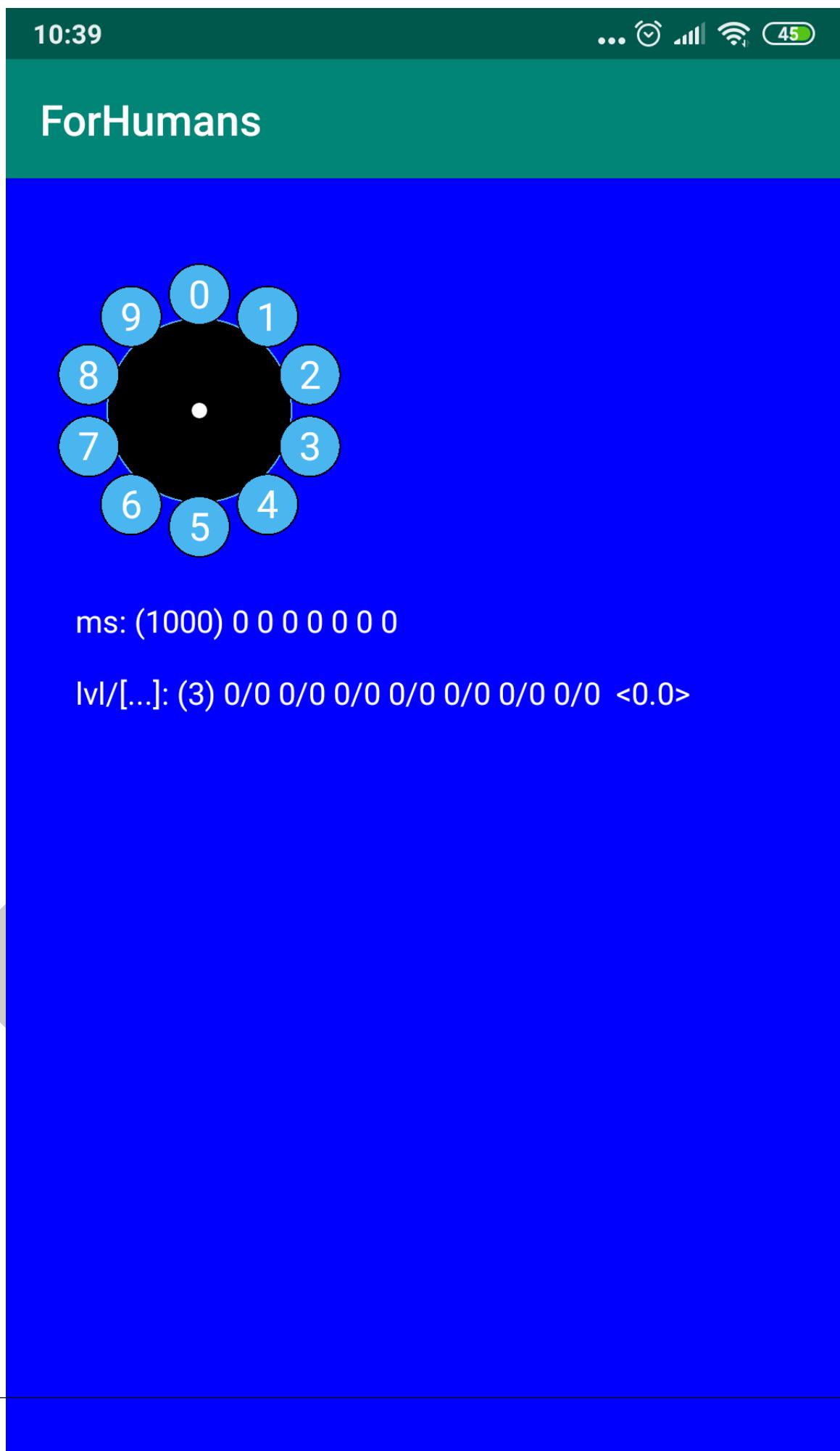
```
Build: completed successfully at 2019.10.02. 17:20  
Starting Gradle Daemon  
Run build /home/yhennnon/Downloads/bhax-master-src-Liskov-HelloAndroid/src/Liskov/HelloAndroid/ForHumans  
Load build  
Configure build  
Calculate task graph  
Run tasks
```

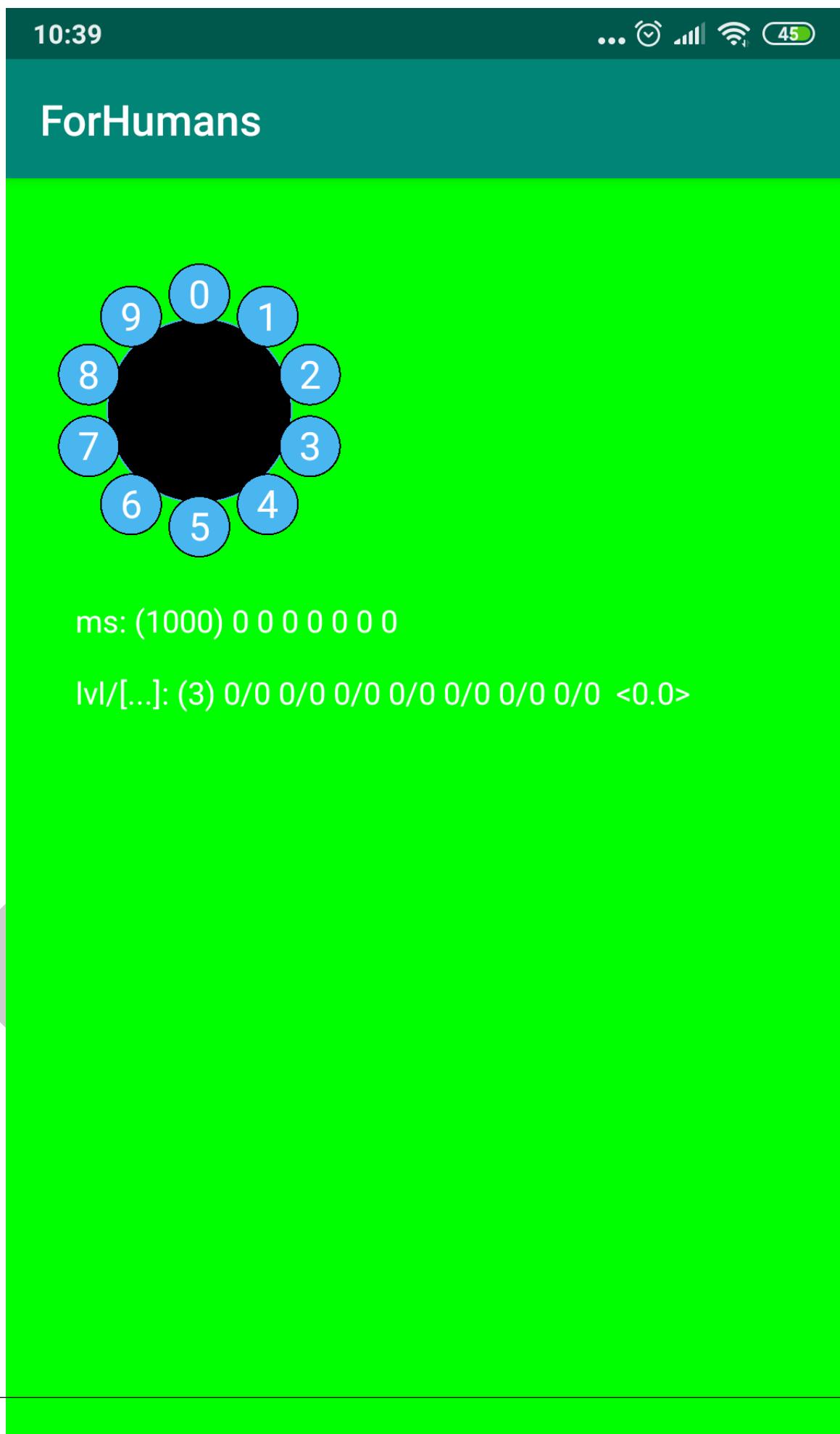
The status bar at the bottom right shows "U+000A" and "265 : 13".

Én nem emulátorral próbáltam ki, hanem USB-vel hozzákötöttem a géphez a telefonomat. Ahhoz hogy érzékelje is az android studio, be kellett kapcsolni telefonon a fejlesztői módot és az USB-Debuggolást is engedélyezni kell.

És Végül így néz ki!

Figure 12.1: RGB beállítás





12.5 Ciklomatikus komplexitás

Feladat: Számoljuk ki valamelyik programunk függvényeinek ciklomatikus komplexitását!

Források: [PiBBP Bench.java](#) [check.xml](#)

A ciklomatikus komplexitás egy szoftvermetrika, amivel egy program komplexitását lehet mérni. A szoftvermetrika olyan mérési eszköz ami a program olyan karakterisztikáit nézi, amik számszerűsíthetőek vagy megszámolhatóak. A szoftvermérés több okból is fontos, ideértve a szoftver teljesítményének mérését, a munkadarabok megtervezését és más felhasználási módokat is.

A ciklomatikus komplexitás a forráskódban az elágazásokból felépülő gráf pontjai, és a köztük lévő élek alapján számítható ki a következő képlettel:

A képlet:

$$M = E - N + 2P \text{ ahol}$$

- E: A gráf éleinek száma
- N: A gráfban lévő csúcsok száma
- P: Az összefüggő komponensek száma

Ezt akár kézzel is kiszámolhatnánk, de vannak rá programok, fölösleges nekünk papíron ezt számolni. A BBP java kódjának a komplexitása:

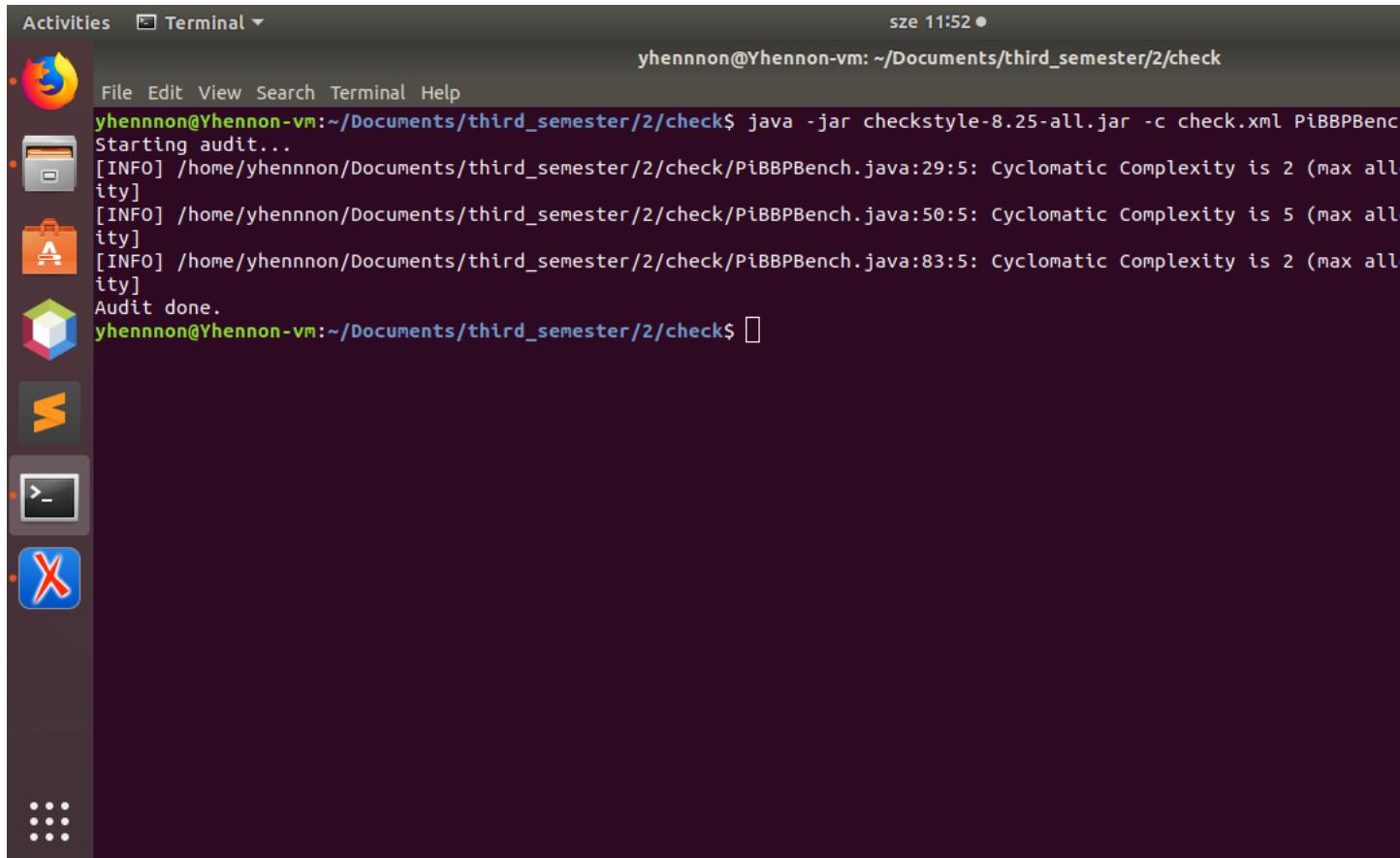


Figure 12.4: checkstyle

A check.xml -ben beállítottam hogy a 0-nál komplexebb részeket dobja ki, így megtudjuk nézni mindeneknek az értékét.

```
..  
..  
<module name="CyclomaticComplexity">  
<property name="max" value="0"/>  
..  
..
```

Chapter 13

Helló, Mandelbrot!

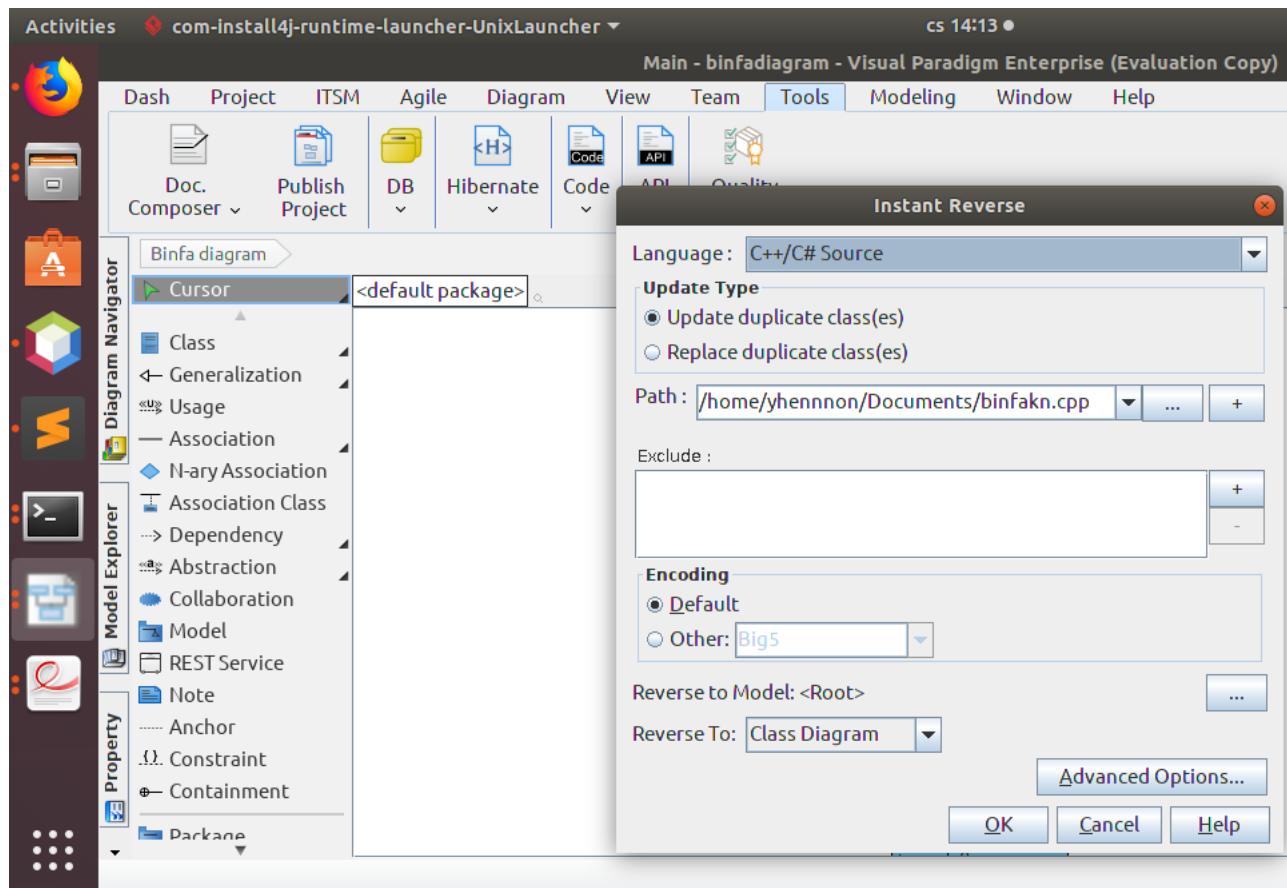
13.1 Reverse engineering UML osztálydiagram

Feladat: UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagrammon.

Az UML a Unified Modelling Language mozaikszava, ami magyarul egységesített modellezési nyelvet jelent. Segítségével a szoftverrendszerünk grafikus modelljét tudjuk vele szemléltetni, könnyen láthatóan ábrázolja az osztályok közötti hierarchiát és kapcsolatokat. Ez a modellező nyelv sokféle beépített diagramtípushoz tartalmaz, amit arra terveztek, hogy jól pontosíthassák, felépíthessék és dokumentálhassák a szoftverrendszer felépítését és működését, és akár üzleti modellekét is. Főképpen üzleti környezetben használatos, a megrendelőknek szokás prezentálni ezeket az UML diagrammokat, hogy valamelyen szinten elmagyarázhassák a rendszer működését, és legyen egy elképzelés arról, hogy az hogyan fog működni. Jó eséllyel a forráskódokat hiába látná a legtöbb ember, az nem annyira tisztán átlátható mint egy osztálydiagram.

A feladat a binfa c++ változatából egy ilyen osztálydiagrammot készíteni, de most nem előre, nem azért hogy a megalkotás előtt látható legyen, hogy hogyan fog felépülni, hanem a már kész kód ből készítjük el. Ehhez segítségül vehetünk rengeteg programot, például a Visual Paradigm nevezetűt. A Visual Paradigm rengeteg lehetőséget nyújt diagramok készítésére, többek között van egy instant reverse funkciója is, pont amire szükségünk van. Diagram készíthető java, c, c++, xml és más nyelveken írt forrásból is, mi kiválasztjuk a binfa c++ forrását, és végigmegyünk a lépésekben. A Visual Paradigm segítségnyújtó dokumentációjában lépésről lépésre le van írva, hogyan is lehet ezt csinálni, és a különböző opciók is ki vannak fejtve.

<https://www.visual-paradigm.com/support/documents/>



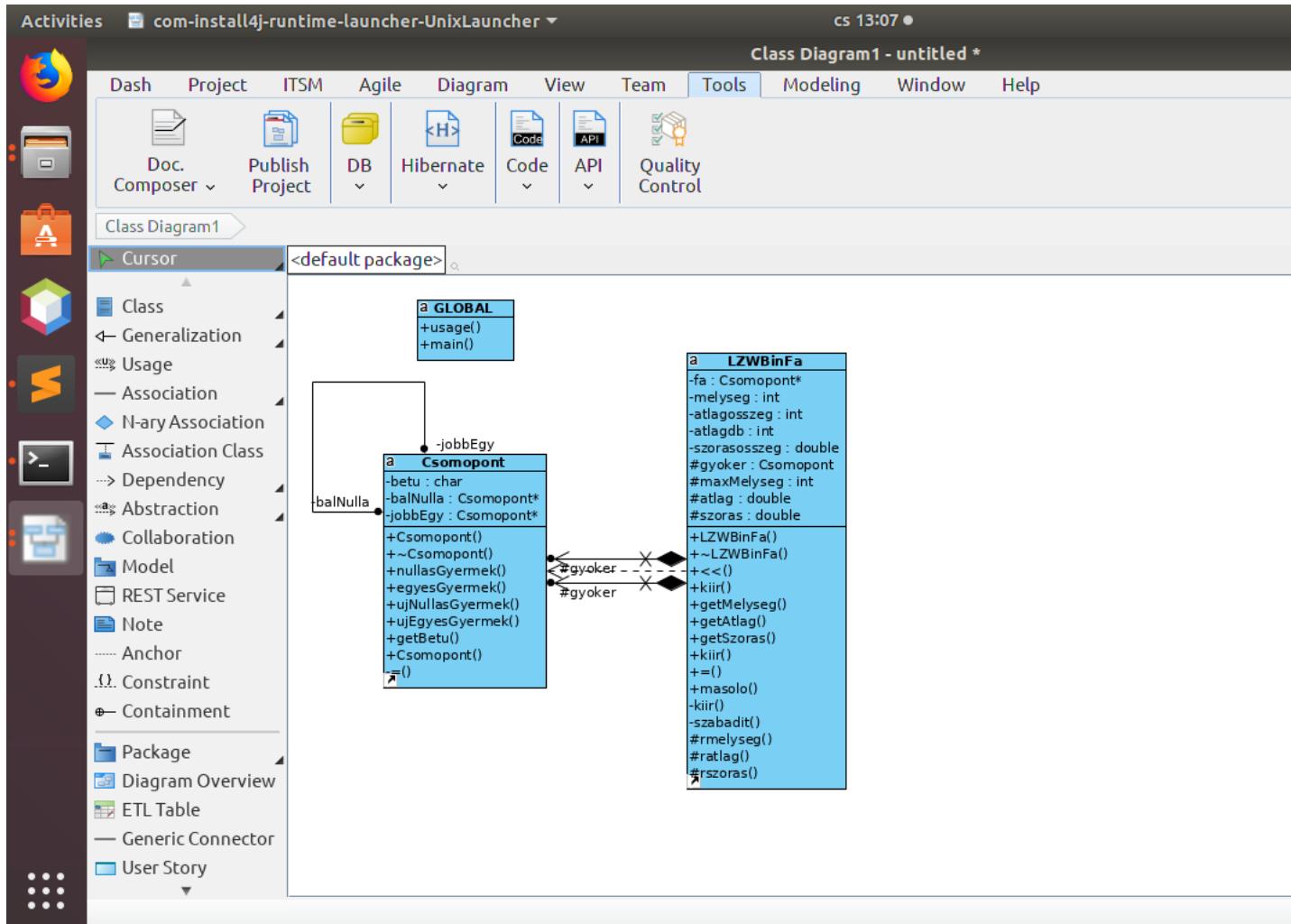


Figure 13.2: Binfa osztálydiagrammja

Fontos még rámutatni az UML diagrammokban használatos kapcsolatokra. Amikor két osztály kommunikál egymással, akkor kapcsolat áll fent közöttük, ezt az ábra jelöli is egy nyíllal. Azonban, különféle kapcsolatok vannak, az elemek állhatnak egymással aggregációban és kompozícióban is. Mindkettő az asszociáció esetei, egy lényeges különbségge. Az aggregáció azt jelenti, hogy a kapcsolatban a gyermek létezhet a szülőjétől függetlenül, a kompozícióban pedig nem. Például, a szülő osztály legyen egy tankör, a gyermek pedig egy tanuló. Ha megszüntetik a tankört, attól még a tanuló létezik. Legyen egy ház és annak a szobái. Ha a házat lerombolják, a szobák is lerombolódnak, nem léteznek a háztól külön. Az aggregációt egy üres mutatójú nyíl jelöli, a kompozíciót egy teli mutatójú. A képen tehát láthatjuk hogy a Csomopont és az LZWBinfa osztályok kompozícióban állnak egymással.

13.2 Forward engineering UML osztálydiagram

Feladat: UML-ben tervezünk osztályokat és generálunk belőle forrást!

Az előző feladat fordítottja, most UML-ből készítünk forráskódot. A Visual Paradigm erre is lehetőséget nyújt. (Sőt, még szinkronizált módon is össze lehet kapcsolni az UML diagramot a forráskóddal, és ha

változtatást üzemelünk be az egyikben, a másik is aszerint fog alakulni, de erről most nem beszélünk.) Miután elkészült egy modell, az még nem elég, kódra is szükségünk van. De mivel kódjából lehet modellt készíteni és modellből is kódot, ezért megválaszthatjuk melyik illik jobban az ízlésünkhez, és debár a modellből készített kód struktúrailag helyes, az implementáció hiányzik, azt természetesen meg kell írni.

<https://circle.visual-paradigm.com/docs/code-engineering/instant-generator/>

Az Instant Generate segítséggel kiválaszthatjuk a modellt és kiválaszthatjuk milyen nyelvre szeretnénk generálni. A modell amit használunk egy, a Visual Paradigm-ba beépített, előre elkészített template.

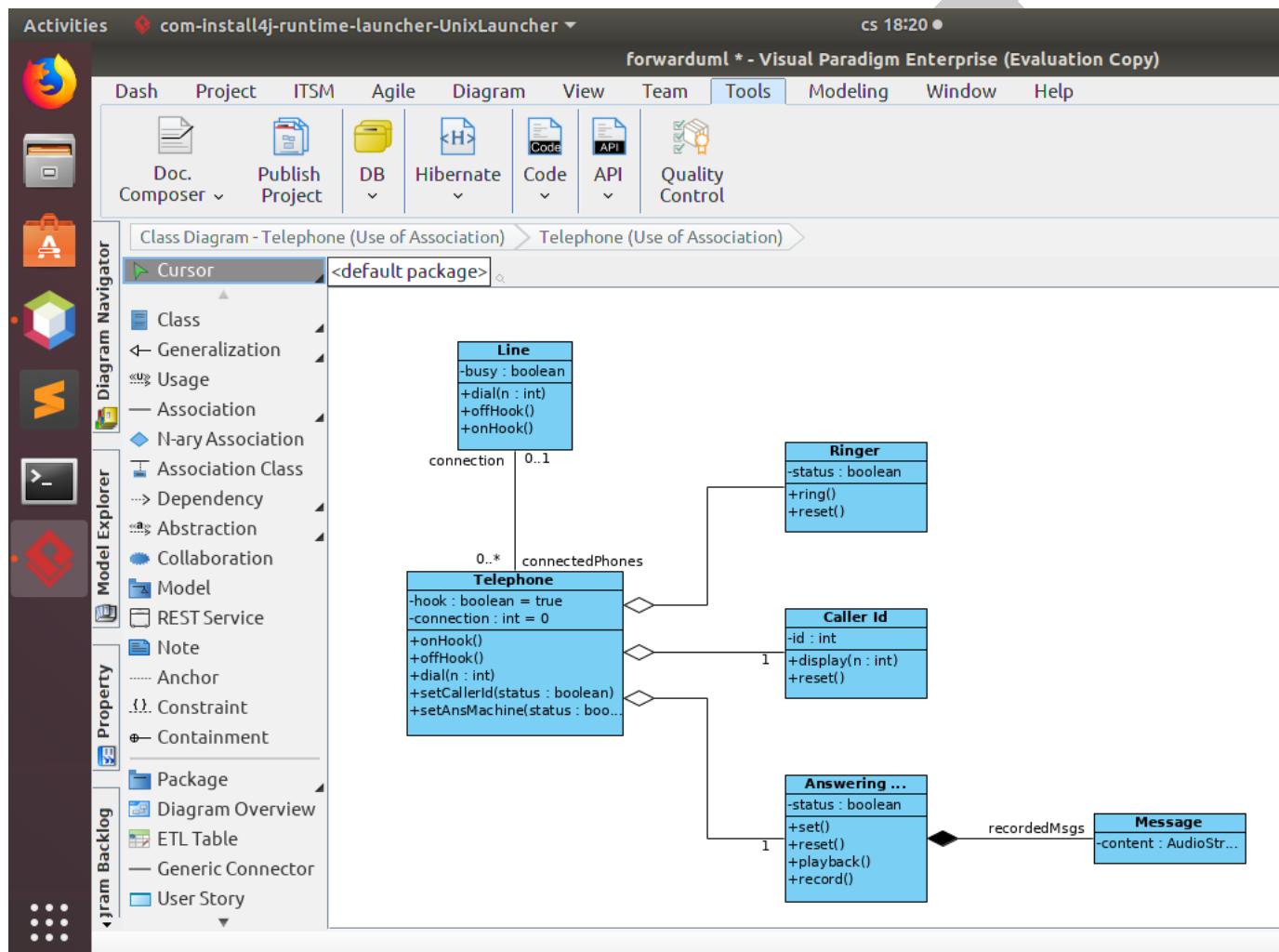


Figure 13.3: Egy beépített telefon kommunikációs template

Jól látható az osztályok kapcsolata, felépítése, a változók és metódusok láthatósága és típusa.

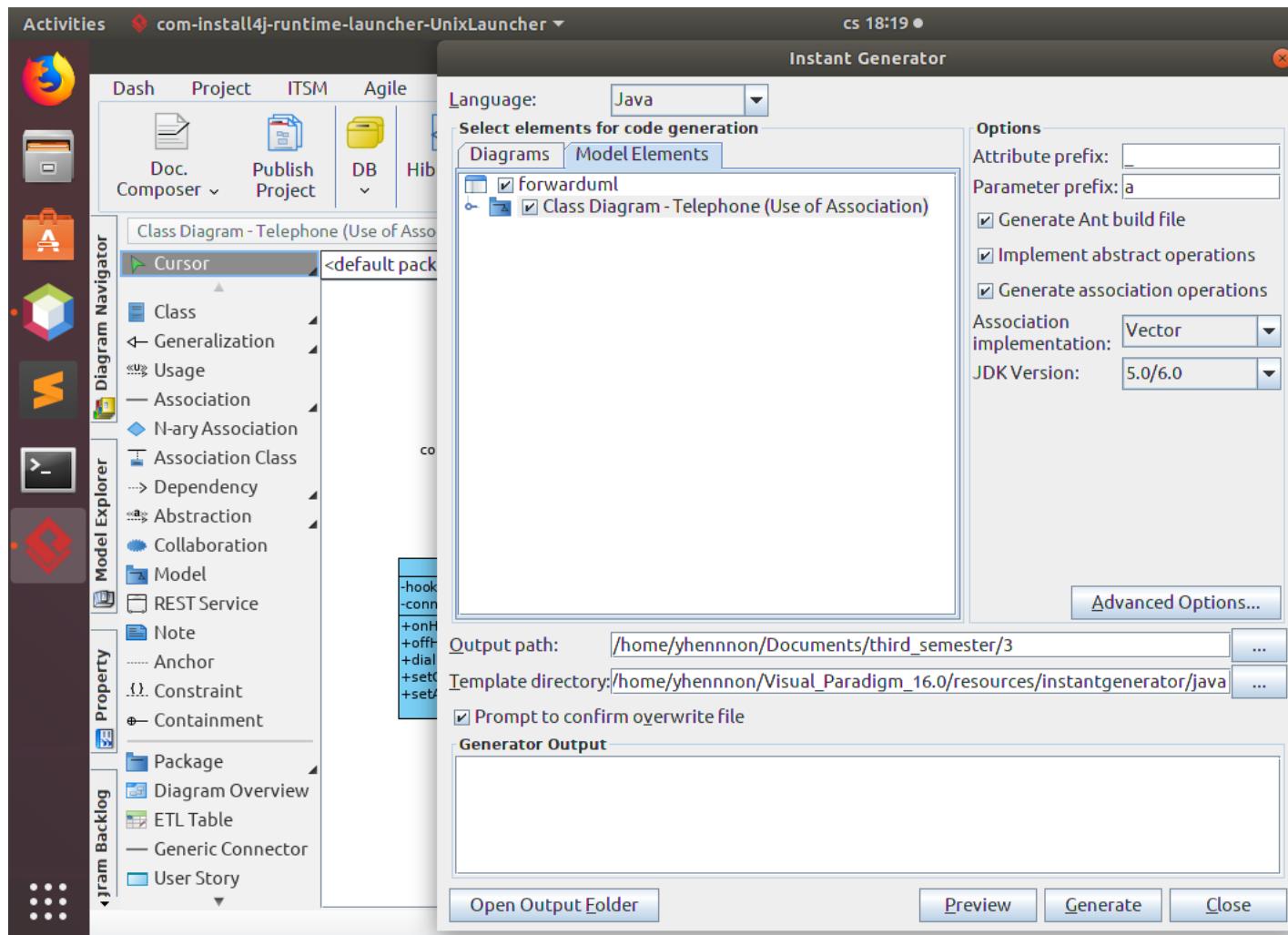


Figure 13.4: Instant Generate használata

A program tökéletesen legenerálja az osztályokat:

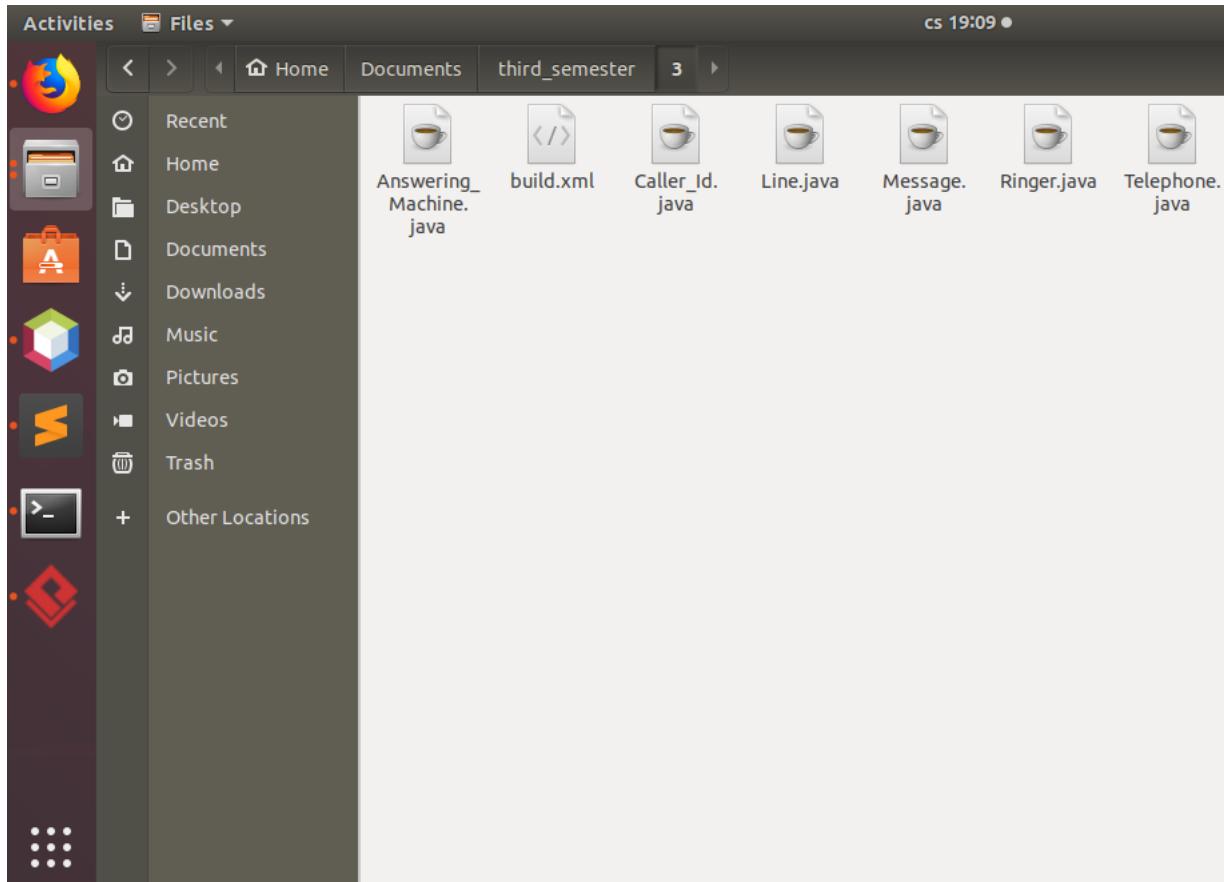


Figure 13.5: A megadott output mappába generált kódok

Pl a Telephone.java generált kódja:

```
public class Telephone {  
    private boolean _hook = true;  
    private int _connection = 0;  
    public Answering_Machine _unnamed_Answering_Machine_;  
    public Caller_Id _unnamed_Caller_Id_;  
    public Ringer _unnamed_Ringer_;  
  
    public void onHook() {  
        throw new UnsupportedOperationException();  
    }  
  
    public void offHook() {  
        throw new UnsupportedOperationException();  
    }  
  
    public void dial(int aN) {  
        throw new UnsupportedOperationException();  
    }  
}
```

```
public void setCallerId(boolean aStatus) {  
    throw new UnsupportedOperationException();  
}  
  
public void setAnsMachine(boolean aStatus) {  
    throw new UnsupportedOperationException();  
}  
}
```

13.3 Egy esettan

Feladat: A BME-s C++ tankönyv 14. fejezetét (427-444 elmelet, 445-469 az esettan) dolgozzuk fel!

13.4 BPMN

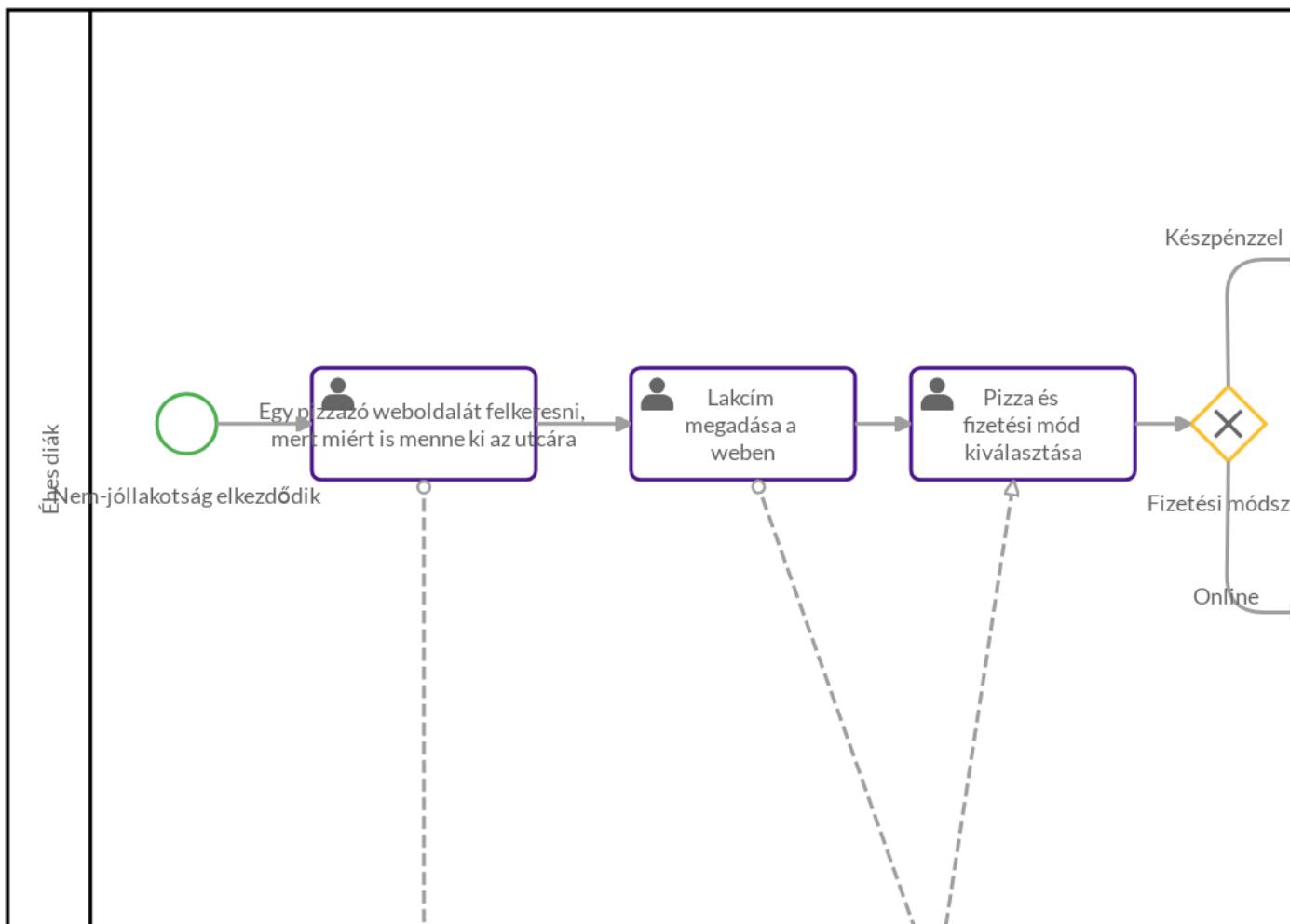
Feladat: Rajzolunk le egy tevékenységet BPMN-ben!

[Pizzarendelés.pbm](#)n

A BPMN, azaz a 'Business Process Model and Notation' egy módszer az üzleti folyamat megközelítésének feltérképezésére; azaz egy összetett üzleti gyakorlat vagy folyamat vizuális ábrázolása. Ennek célja, hogy a főbb érdekelt felek számára egyértelműen egy perspektívát nyújtson a megalapozott döntések segítéséhez, ugyanúgy mint ahogy egy térkép segít kitalálni a lehető legjobb útvonalat.

A cardanit tervezőjét használtam a feladat elkészítéséhez, ami online, regisztrálás után ingyenesen használható-
<https://www.cardanit.com>

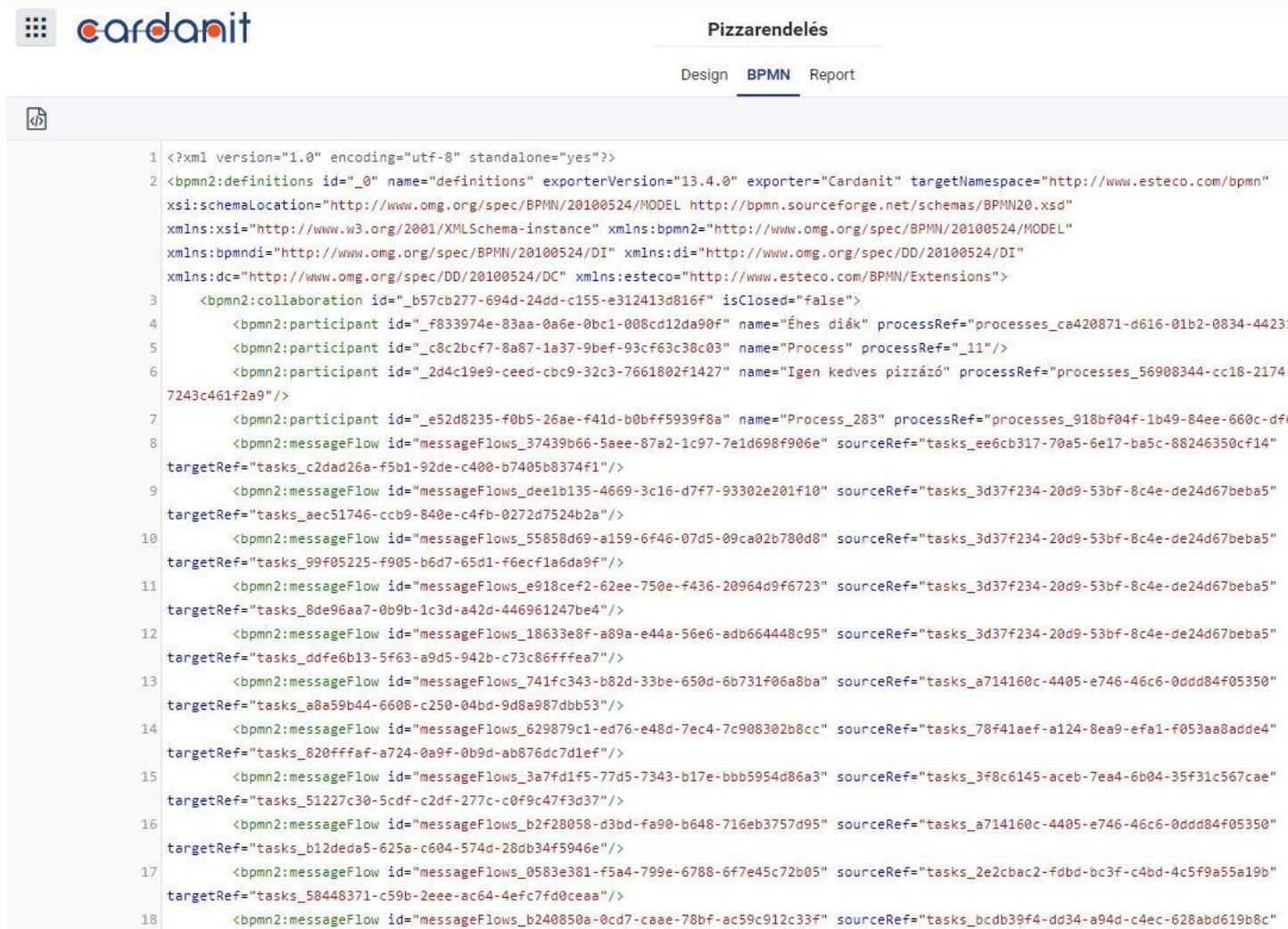
A weboldal rengeteg tutorialnal szolgál, ezekből egy párat megnéztem, könnyen megérhető és el is készítettem egy pizzarendelés menetét. A kép elég széles, a repoban a képek között megtalálható, pizzarendelés.png néven.



Process_283

A futár kiküldése a weboldalon megadott címre

Amit mi grafikusan elkészítünk, abból természetesen egy kódot generál az oldal, ezt le lehet tölteni mint ahogy a fényképet is. A tevékenységeknek vannak tulajdonságai, struktúrái, ezeknek a kapcsolatát írja meg a generátor.



The screenshot shows a web-based BPMN editor interface. At the top, there's a logo for 'cardanit' and a navigation bar with tabs: 'Design', 'BPMN' (which is underlined), and 'Report'. Below the navigation is a toolbar with icons for file operations like new, open, save, and print. The main area contains a large block of XML code representing a BPMN process definition. The code is numbered from 1 to 18 on the left side. It defines various participants, message flows, and tasks, illustrating a complex workflow for pizza ordering. The XML uses standard BPMN 2.0 namespace declarations and includes references to specific processes and tasks defined in the model.

```

1 <?xml version="1.0" encoding="utf-8" standalone="yes"?>
2 <bpmn2:definitions id="_0" name="definitions" exporterVersion="13.4.0" exporter="Cardanit" targetNamespace="http://www.esteco.com/bpmn"
xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL http://bpmn.sourceforge.net/schemas/BPMN20.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC" xmlns:esteco="http://www.esteco.com/BPMN/Extensions">
3   <bpmn2:collaboration id="_b57cb277-694d-24dd-c155-e312413d816f" isClosed="false">
4     <bpmn2:participant id="_f833974e-83aa-0a6e-0bc1-008cd12da90f" name="Éhes diákok" processRef="processes_ca420871-d616-01b2-0834-4423"
5       <bpmn2:participant id="_c8c2bcf7-8a87-1a37-9bef-93cf63c38c03" name="Process" processRef="_11"/>
6       <bpmn2:participant id="_2d4c19e9-ceed-cbc9-32c3-7661802f1427" name="Igen kedves pizzázó" processRef="processes_56908344-cc18-2174
7243c461f2a9"/>
7       <bpmn2:participant id="_e52d8235-f0b5-26ae-f41d-b0bff5939f8a" name="Process_283" processRef="processes_918bf04f-1b49-84ee-660c-df
8         <bpmn2:messageFlow id="messageflows_37439b66-5aee-87a2-1c97-7e1d698f906e" sourceRef="tasks_ee6cb317-70a5-6e17-ba5c-88246350cf14"
targetRef="tasks_c2dad26a-f5b1-92de-c400-b7405b374f10"/>
9         <bpmn2:messageFlow id="messageflows_deeb135-4669-3c16-d7f7-93302e201f10" sourceRef="tasks_3d37f234-20d9-53bf-8c4e-de24d67beba5"
targetRef="tasks_aec51746-ccb9-840e-c4fb-0272d7524b2a"/>
10        <bpmn2:messageFlow id="messageFlows_55858d69-a159-6f46-07d5-09ca02b780d8" sourceRef="tasks_3d37f234-20d9-53bf-8c4e-de24d67beba5"
targetRef="tasks_99f05225-f905-b6d7-65d1-f6ecf1a6da9f"/>
11        <bpmn2:messageFlow id="messageFlows_e918cef2-62ee-750e-f436-20964d9f6723" sourceRef="tasks_3d37f234-20d9-53bf-8c4e-de24d67beba5"
targetRef="tasks_8de96aa7-0b9b-1c3d-a42d-446961247be4"/>
12        <bpmn2:messageFlow id="messageFlows_18633e8f-a89a-e44a-56e6-adb664448c95" sourceRef="tasks_3d37f234-20d9-53bf-8c4e-de24d67beba5"
targetRef="tasks_ddfe6b13-5f63-a9d5-942b-c73c86fffea7"/>
13        <bpmn2:messageFlow id="messageFlows_741fc343-b82d-33be-650d-6b731f06a8ba" sourceRef="tasks_a714160c-4405-e746-46c6-0ddd84f05350"
targetRef="tasks_a8a59b44-6608-c250-04bd-9d8a987dbb53"/>
14        <bpmn2:messageFlow id="messageFlows_629879c1-ed76-e48d-7ec4-7c908302b8cc" sourceRef="tasks_78f41aef-a124-8ea9-efaf1-f053aa8adde4"
targetRef="tasks_820ffffaf-a724-0a9f-0b9d-ab876d7dlef"/>
15        <bpmn2:messageFlow id="messageFlows_3a7fd1f5-77d5-7343-b17e-bbb5954d86a3" sourceRef="tasks_3f8c6145-aceb-7ea4-6b04-35f31c567cae"
targetRef="tasks_51227c30-5cdf-c2df-277c-c0f9c47f3d37"/>
16        <bpmn2:messageFlow id="messageFlows_b2f28058-d3bd-fa90-b648-716eb3757d95" sourceRef="tasks_a714160c-4405-e746-46c6-0ddd84f05350"
targetRef="tasks_b12deda5-625a-c604-574d-28db34f5946e"/>
17        <bpmn2:messageFlow id="messageFlows_0583e381-f5a4-799e-6788-6f7e45c72b05" sourceRef="tasks_2e2cbac2-fdbd-bc3f-c4bd-4c5f9a55a19b"
targetRef="tasks_58448371-c59b-2eee-ac64-4efc7fd0ceaa"/>
18        <bpmn2:messageFlow id="messageFlows_b240850a-0cd7-caee-78bf-ac59c912c33f" sourceRef="tasks_bcdb39f4-dd34-a94d-c4ec-628abd619b8c"/>

```

Figure 13.7: Generált kód

13.5 TeX UML || Tutorom: Rácz András István

Feladat: Valamilyen TeX-es csomag felhasználásával készíts szép diagramokat az OOCWC projektről (pl. use case és class diagramokat).

Forrás : [Robocar.mp](#)

A szöveges UML eszköz támogatja a szöveges jelölések / nyelvek használatát az UML modellek leírására, és automatikusan megjeleníti a megfelelő grafikus UML diagramot a szöveges leírásból. A szabad forráskód miatt bármilyen operációs rendszeren használható. Ez egyben meghatározza a hordozhatóságot is, mivel a dokumentum tényleges létrehozása nem függ a használt szerkesztő programtól vagy az operációs rendszer sajátosságaitól. A kész dokumentum minden szerkesztő és minden szoftveres környezet esetén azonos.

Nem fordulhat elő az a helyzet, hogy egy dokumentum megjelenése erősen függ a szövegszerkesztő verziójától vagy a különböző szerkesztőprogramok kompatibilitási szintjétől. A rendszer része egy viszonylag egyszerű szövegjelölő nyelv, amely alapján a szöveget először ellátjuk a megjelenítési információkat hor-dozó utasításokkal. A jelöléssel ellátott szövegből azután egy fordítóprogram létrehozza a megjeleníthető dokumentumot.

Ezen a nyelven az összes osztály:

```
Class.A("MyShmClient")
  ("# nr_graph: NodeRefGraph*", "# m_teamname: string", "- nr2v: map<↔
    unsigned_object_id_type, NRGVertex>")
  ("+MyShmClient()", "+~MyShmClient()", "+start()", "+start10()", "+num_vertices()", "+print_edges()", "+print_vertices()", "+bgl_graph()", "+hasDijkstraPath()", "+hasBellmanFordPath()", "-foo()", "-init()", "-gangsters()", "-initcops()", "-pos()", "-car()", "-route());

Class.B("SmartCar")
  ("+ id: int",
  "+ from: unsigned",
  "+ to: unsigned",
  "+ step: int")
  ();

Class.C("Gangster")
  ();
  ();

Class.D("Cop")
  ();
  ();

Class.E("ShmClient")
  ("# shm_map: boost.interprocess.offset_ptr<justine.robocar.shm_map_Type ↔
    >",

  
```

```
"- segment: boost.interprocess.managed_shared_memory*)  
(" + ShmClient(),  
" + ~ShmClient(),  
" + start(),  
" + get_random_node(),  
" + num_edges,  
" + alist(),  
" + alist_inv(),  
" + salist(),  
" + set_salist(),  
" + palist(),  
" + hasNode(),  
" + dst(),  
" + dst(),  
" + toGPS(),  
" + toGPS(),  
"- foo(),  
"- init(),  
"- gangsters(),  
"- pos(),  
"- car(),  
"- route1(),  
"- route2());  
  
Class.F("OSMReader")  
(" + onewayc: int,  
" + onewayf: int,  
" #nOSM_nodes: int,  
" #nOSM_ways: int,  
" #nOSM_relations: int,  
" #sum_unique_highhway_nodes: int,  
" #sum_highhway_nodes: int,  
" #sum_highhway_length: int,  
" #edge_multiplicity: int,  
" #nbuses: int,  
" #max_edge_length: double,  
" #mean_edge_lenght: double,  
" #cedges: int,  
" #locations: OSMLocations,  
"- m_estimator: size_t,  
"- alist: AdjacencyList&,  
"- palist: AdjacencyList&,  
"- waynode_locations: WaynodeLocations&,  
"- busWayNodesMap: WayNodesMap&,  
"- way2nodes: Way2Nodes&,  
"- way2name: WayNames&)  
(" + OSMreader(),  
" + ~OSMReader(),  
" + get_estimated_memory(),  
" + edge(),
```

```
"+node()",  
"+way()",  
"+relation",  
"-distance()");
```

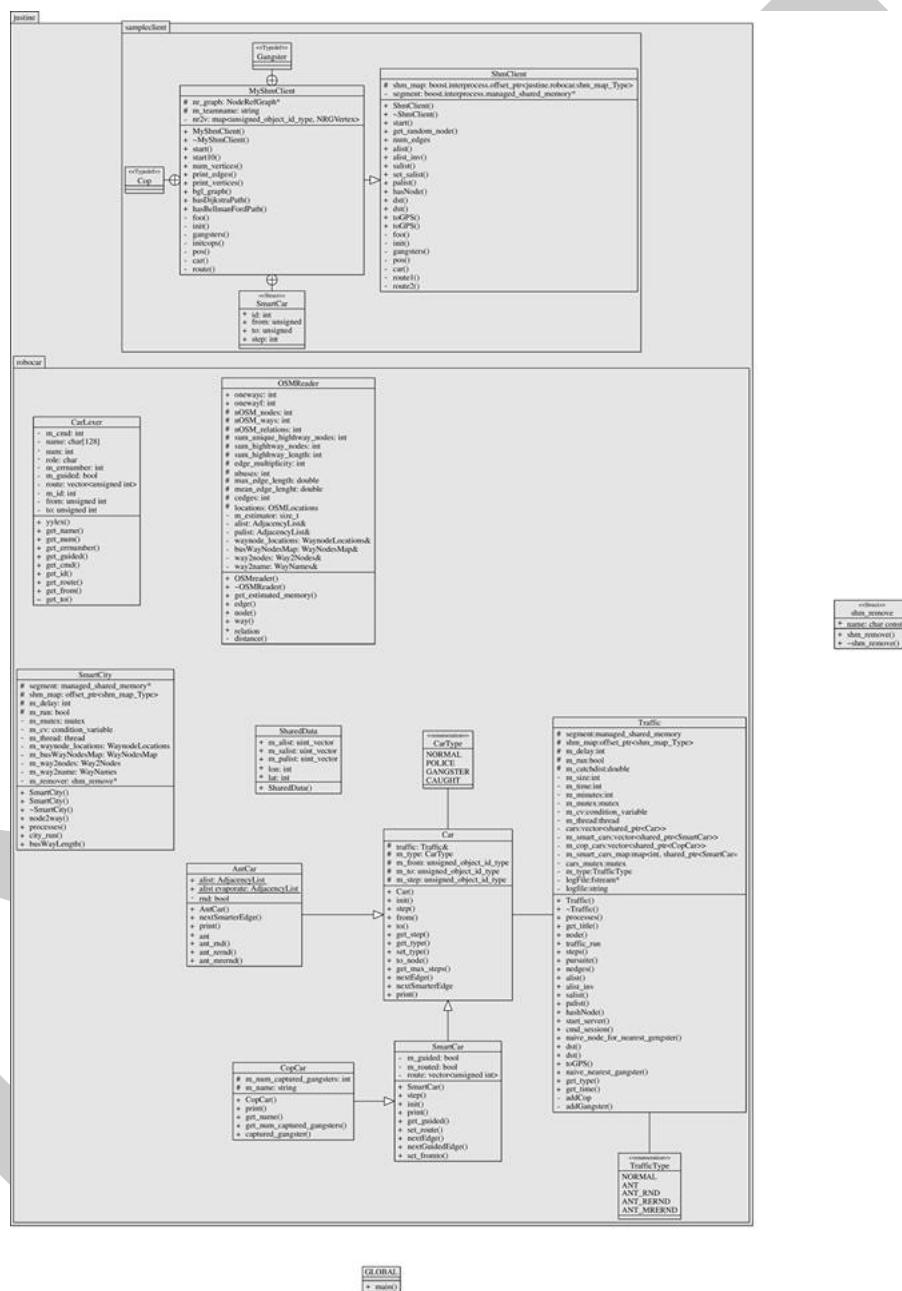


Figure 13.8: OOCWC

Chapter 14

Helló, Chomsky!

14.1 Encoding

Feladat: Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket!

Forrás: [MandelbrotHalmazNagyító.java](#)

A megadott forrásunkban az eddig megszokottól eltérően most magyarul írtuk a forráskódot. Ez felveti az encode-olás problémáját, ugyanis nem mindegyik beszélt nyelv ugyanazokat a karaktereket tartalmazza. Ha megpróbáljuk simán fordítani a nagyítót, hibaüzenetet kapunk amikor olyan karakterrel találkozik a compiler amit nem ismer fel az alapértelmezett UTF-8 as kódolással.

```
MandelbrotHalmazNagyító.java:40: error: unmappable character (0xF3) for ←
encoding UTF-8
    // vizsg\xfffd;ljuk egy adott pont iter\xfffd;ci\xfffd;it:
                                         ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xE9) for ←
encoding UTF-8
    // Az eg\xfffd;rmutat\xfffd; poz\xfffd;ci\xfffd;ja
          ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xF3) for ←
encoding UTF-8
    // Az eg\xfffd;rmutat\xfffd; poz\xfffd;ci\xfffd;ja
          ^
MandelbrotHalmazNagyító.java:42: error: unmappable character (0xED) for ←
encoding UTF-8
    // Az eg\xfffd;rmutat\xfffd; poz\xfffd;ci\xfffd;ja
          ^
100 errors
```

Ahhoz hogy ezt megoldjuk, tudnunk kell hogy milyen kódolásba esik bele az amit használtunk a forrás megírásához. Mi a magyar nyelvet használtuk, ami közép-európai, latin nyelvbe esik bele(több "rokonjával", pl: Lengyel,Szlovák,Cseh,Szerb,Szlovén etc..) Ezeket a nyelveket a Windows-1250 karakterkészlete

fedi le, azért a javanak egy -encoding kapcsolóval megadva az ehhez tartozó Cp1250-et már fordítani tudjuk a kódot,

```
yhennnon@Yhennnon-vm:~/Documents/third_semester/4/javat- ←
tanitok-javat/forrasok/javat_tanitok_forrasok/ ←
nehany_egyeb_pelda$ javac -encoding Cp1250 ←
MandelbrotHalmazNagyító.java
```

és futtathatjuk .

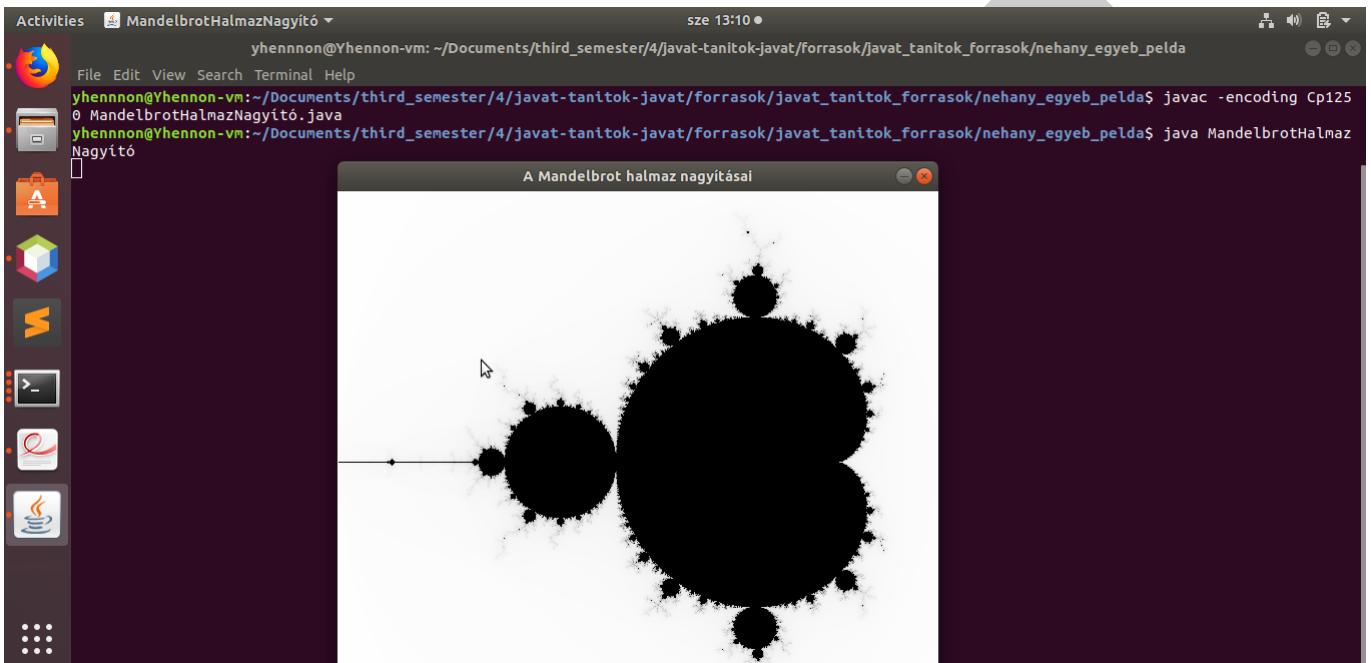


Figure 14.1: Fordítás Cp1250-el

14.2 I334d1c4 5

Feladat: Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet>

Forrás: [leet.cpp](#)

A leet nyelv a latin betűket helyettesíti számokkal, különböző betűkkel, ASCII, karakterekkel. A leet, vagy eleet jelentése az elit-et jelképezi, amikor kialakult ez a beszédstílus, azok az emberek használták, akik el akarták titkolni a beszélgetésüket a casual usereketől, csak a "kiváltságosak" érthették meg ezt a beszédet. Manapság ez a szándékosan elírt beszéd már széles körben ismert, és a régebbi időktől eltérően nem rosszindulatú célokra használják.

A feladat egy olyan encodert létrehozni, aminek megadunk egy szöveget, és a karaktereket a leet abcbeli megfelelőire alakítja át. Ehhez a használt abcn-k minden betűjéhez társítanunk kell egy másik karaktert, ezeket sorrendben elhelyezzük egy tömbben.

```
[  
string alphabet= " ←  
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  
string leet[62]={"4","8",")","D","3","F","9","H","I","J","K",  
"1","M","N","0","P","9","R","5","7","U","V","W","X","Y","2",  
"4","b","(,"d","3","f","9","h","i","j","k","l","m","n","0",  
"p","q","r","5","7","u","v","w","x","y","2","O","I","Z","E","h","S","b","T" ←  
, "X", "g"};
```

Ezután bekérünk egy szöveget,majd kiírjuk az eredményt:

The screenshot shows a terminal window titled 'yhennon@Yhennon-vm: ~/Downloads'. The user runs 'g++ leet.cpp -o leet' and then './leet'. The program prompts for input with 'Irja be a fordítando szoveget.' and receives the message '>Oi bruv, name me a better TF than me!'. It then outputs the converted Leet text: 'Oi bruv, name me a better TF than me! -> Oi Xr|_| nhmE mE h XETTER TF Thhn mE!'.

Figure 14.2: Leet

14.3

Feladat:

14.4 Full screen

Feladat: Készítsünk egy teljes képernyős Java programot!

Forrás : [BouncingBall.java](#)

A feladat elkészítéséhez egy régebben megírt pattogó labda játékot használok fel, amit kiegészíték. Felhasználom a java abstract window toolkit-jét, aminek segítségével GUI-kat és ablak-alapú alkalmazásokat lehet készíteni. A BufferStrategy osztály azt a mechanizmust képviseli, amellyel az összetett memória egy adott vászonon vagy ablakon megszervezhető. A hardver és a szoftver korlátozásai határozzák meg, hogy egy adott pufferstratégia megvalósítható-e és hogyan. Ezek a korlátozások a vászon vagy ablak létrehozásakor használt GraphicsConfiguration képességein keresztül észlelhetők.

```
import java.awt.image.BufferStrategy;
```

A device.setFullScreenWindow();-al teljesképernyőt választunk ki, egy négyzet lesz az ablak, aminek a méretezése a képernyőnknek megfelelő lesz, megnézzük van-e beállított legjobb megjeleníti mód, ha nincs beállítva, de állítható a megjelenítés, akkor ezt úgy állítjuk be különöző képekre, hogy a használt eszközünk konfigurációját lekérdezzük, a szélességet és a magasságot beállíttatjuk.

```
..  
..  
public static void chooseBestDisplayMode(GraphicsDevice device) {  
    DisplayMode best = getBestDisplayMode(device);  
    if (best != null) {  
        device.setDisplayMode(best);  
    }  
}  
..  
.
```

Fordítjuk, futtajuk a programot, és az eddigi pattogó labda most már teljes képernyőn garázdálkodik:



Figure 14.3: Teljes képernyőn

14.5 Perceptron osztály

Feladat: Dolgozzuk be egy külön projektbe a projekt Perceptron osztályát! <https://youtu.be/XpBnR31BRJY>

Megoldás forrása: [main.cpp](#) [mandel.cpp](#) [mlp.hpp](#)

A perceptron osztályt az mlp.hpp fájlban találhatjuk meg, ezt fogjuk felhasználni a main.cpp ben. Előző félévben volt feladat megismерkedni Neurális Or, And, és Exor kapukkal. Röviden, a Perceptron osztály ezeket a neurális hálókat valósítja meg a mesterséges intelligenciák számára használható módon. A kapuknak tulajdonsága hogy taníthatóak , meg kell adni a bemenetet, és ez alapján különböző mintákat alkot, különböző csomópontokon megy keresztül, amelyek használattól függően fognak kiértékelődi, és kimeneti értéket adni. Nem logikai kifejezéseket kell most a Perceptronnak megadni, hanem egy mandelbrot hal-mazról készített képet,méretet.

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);

    double* image = new double[size];

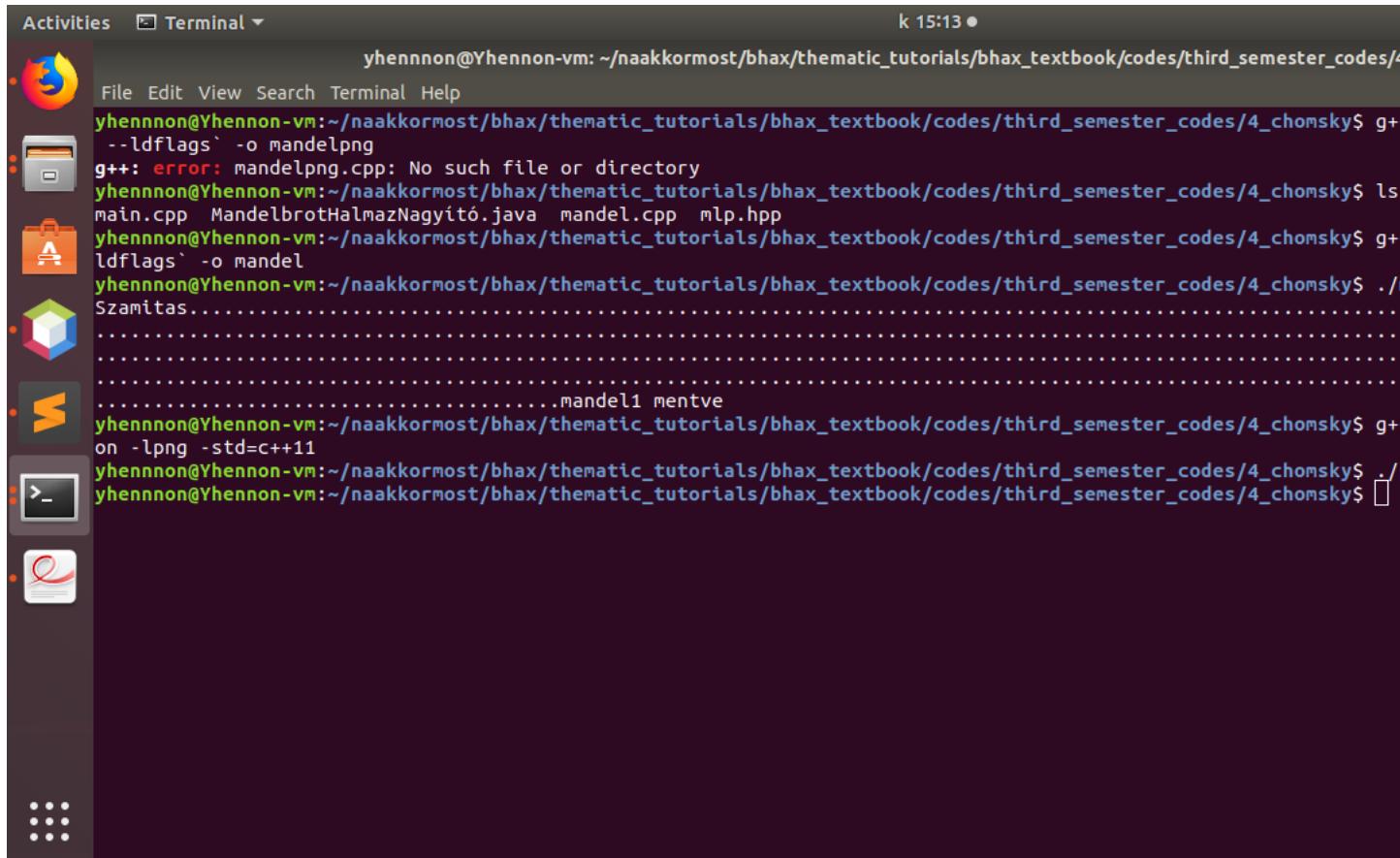
    for(int i {0}; i<png_image.get_width(); ++i)
        for(int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;

    double value = (*p) (image);

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

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

Fordítjuk, lefuttatjuk, megadjuk értéknek a készített képet, és egy értéket kell visszakapnunk.:



The screenshot shows a Linux desktop environment with a dark theme. On the left is a vertical dock titled "Activities" containing icons for various applications: a browser, file manager, terminal, code editor, and others. The main window is a terminal titled "Terminal". The terminal window has a dark background and light-colored text. It displays the following command-line session:

```
k 15:13 ●  
yhennnon@Yhennon-vm: ~/naakkormost/bhax/thematic_tutorials/bhax_textbook/codes/third_semester_codes/4_chomsky$ g++ --ldflags` -o mandelpng  
g++: error: mandelpng.cpp: No such file or directory  
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/codes/third_semester_codes/4_chomsky$ ls  
main.cpp MandelbrothalmazNagyító.java mandel.cpp mlp.hpp  
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/codes/third_semester_codes/4_chomsky$ g++ ldflags` -o mandel  
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/codes/third_semester_codes/4_chomsky$ ./Szamitas  
.....  
.....mandel1 mentve  
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/codes/third_semester_codes/4_chomsky$ g++ on -lpng -std=c++11  
yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/codes/third_semester_codes/4_chomsky$ ./yhennnon@Yhennon-vm:~/naakkormost/bhax/thematic_tutorials/bhax_textbook/codes/third_semester_codes/4_chomsky$
```

Figure 14.4: Perceptron

Chapter 15

Helló, Stroustrup!

15.1 JDK osztályok & Összefoglaló

Feladat: Írunk olyan Boost C++ programot (indulj ki például a fénykardból) amely kilistázza a JDK összes osztályát (miután kicsomagoltuk az src.zip állományt, arra ráengedve)!

Forrás: [boost.cpp](#) [fenykard.cpp](#)

Mi is a Boost:

A Boost Filesystem könyvtár hordozható eszközöket kínál az elérési utak, fájlok és könyvtárak lekérdezésére és kezelésére. A könyvtár létrehozására való motiváció az volt, hogy képes legyen hordozható szkriptszerű műveleteket végrehajtani a C programokon belül. A cél nem a Python, Perl vagy shell nyelvekkel való versengés, hanem a hordozható fájlrendszer műveletek biztosítása, amikor a C már a választott nyelv. A kialakítás ösztönzi, de nem igényli a biztonságos és hordozható fájlrendszer használatát. A Filesystem Library számos fejlécet szolgáltat, mindegyik a boost/filesystem könyvtárban található (amit egy sudo apt-get install libboost-all-dev parancsal telepítünk):

- A path.hpp a path osztályt biztosítja, amely egy hordozható mechanizmus az elérési utak ábrázolásához C programokban. Érvényesség-ellenőrző funkciók is rendelkezésre állnak.
- Az operations.hpp fájlokkal és könyvtárakkal működő funkciókat biztosít, és magában foglalja a directory_iterator-t.
- Az fstream.hpp fejléc ugyanazokat az összetevőket nyújtja, mint a C Standard Library fstream fejléce, azzal a különbséggel, hogy a fájlokat inkább az elérési út objektumok azonosítják, nem a char * -ok.
- Az exception.hpp nyújtja a filesystem_error osztályt.
- A comfort.hpp olyan kényelmi funkciókat kínál, amelyek hasznos módon kombinálják az alacsonyabb szintű funkciókat.

A szervezési elv az, hogy a tisztán lexikális műveleteket az elérési útvonalakon az elérési út osztálytag-függvényeként nyújtja a path.hpp, míg az operációs rendszer által a tényleges külső fájlrendszer könyvtárakban és fájlokban végrehajtott funkciók az operations.hpp-ban állnak rendelkezésre.

A feladathoz megvan adva a fenykard.cpp, amiből nekünk csak az a rész kell, ami a fájlrendszerrel foglalkozik, ennek segítségével , boost könyvtárak használatával tudjuk majd a fájlrendszerünk egy mappájában a .java kiterjesztésű fájlokat keresni. A boost könyvtárat le kellett tölteni, azután importálhatjuk header fájlként. A JDK src mappájában lévő java fájlokat keressük. A miheztartás végett fontos, hogy ne elégedjünk meg egy kiterjesztés szűrésével,mert lehet .java végződésű mappa is, ezért a mappákra nem keresünk. Rekurzív módon fog történni a kiértékelés, ami annyit tesz, hogy egy mappában megkerestetnek a fájlok és a mappák, ha van mappa, akkor abba belépünk és keresünk újra fájlokat és mappákat, amíg végül olyan mappába nem jutunk ahol csak fájl van, és így, a legutolsó ágtól kezdve számoljuk a java fájlokat.

```
void travel(boost::filesystem::path path) {
    boost::filesystem::directory_iterator it{path}, eod;
    BOOST_FOREACH(boost::filesystem::path const& p, std::make_pair(it, ←
        eod)) {
        if (boost::filesystem::is_regular_file(p) && boost::filesystem::extension(p.string()) == ".java") {
            std::cout << p << std::endl;
            _count++;
        }
        else if (boost::filesystem::is_directory(p)) travel(p);
    }
}
```

A programot fordítjuk a boost könyvtárak használatához szükséges kapcsolókkal,(g++ boost.cpp -o searcher -lboost_system -lboost_filesystem -lboost_program_options -std=c++14), majd futtatjuk, és megkapjuk hány java fájt tartalmaz az src könyvtár.

magamnak: képet beszúrni !

15.2 Másoló-mozgató szemantika

Feladat: Kódcsipeteken (copy és move ctor és assign) keresztül vesd össze a C++11 másoló és a mozgató szemantikáját, a mozgató konstruktort alapozd a mozgató értékkadásra!

Ismét egy régebben előkerülő dolgot vesszünk elő,a binfát, ezen jól szemléltethető a feladat.

Forrás: [12binfa.cpp](#)

Másolás esetén:

Rengetegszer van az, hogy egy objektumból több példányra van szükség, ekkor másoljuk az objektumunkat, nem kell felépíteni az egészet minden alkalommal, egyszerűen csak példányosítjuk, így minden változó és érték ugyanaz le, ezeket igény szerint lehet példányszinten alakítani. Az érdekes az, amikor az objektumunk nem igazán egyszerű, és tartalmat pointereket, ekkor a másolt objektum is ugyanazokat fogja tartalmazni, ami viszont már egy nem feltétlen kényelmes megoldás, mert ha megváltozik a mutató által címzett dolog, akkor az minden összes másolat esetében is megváltozik, természetesen. Alapértelmezetten a másolásnál a fordító bitenként másol, ezt hívják sekély másolásnak, és ha ez nem elég, dinamikus egységenként akarunk másolni, azt mélymásolásnak hívják, és ezt nekünk kell megírni, hogy a fordító ezt használhassa.

A mozgató konstruktor és értékkadás:

```
LZWBinFa ( LZWBinFa && regi ) {  
    std::cout << "LZWBinFa mozgató konstruktor" << std::endl;  
  
    gyoker = nullptr;  
    *this = std::move ( regi );  
  
}  
LZWBinFa & operator= ( LZWBinFa && regi)  
{  
    std::cout << "LZWBinFa mozgató értékadás" << std::endl;  
    std::swap ( gyoker, regi.gyoker );  
  
    return *this;  
}
```

A mozgató konstruktorban a gyökeret kinullázzuk, majd a `*this` el az aktuális fának a mutatóját a `regi`-re állítjuk. Így nem kell az összes elemet átmásolni, az új fa a régire és annak értékeire fog mutatni. A `root` egy lvalue, a `regi` pedig egy nem konstans rvalue.

Az értékadás során igénybe vesszük az előző mozgató konstruktort. A `move` tulajdonképpen semmit sem mozgat, a régiből jobbérték(rvalue) referenciát készít, ezzel kiváltja, kikényszeríti a mozgató értékadás hívását, ez a valódi mozgatás így meg végbe.

15.3 Változó argumentumszámú ctor

Feladat: Készítsünk olyan példát, amely egy képet tesz az alábbi projekt Perceptron osztályának bemenetére és a Perceptron ne egy értéket, hanem egy ugyanakkora méretű „képet” adjon vissza. (Lásd még a 4 het/Perceptron osztály feladatot is.)

Forrás: [perceptron.cpp](#)

A már többször is előforduló perceptronnal foglalkozunk újra, aminek a működését már a 4. heti feladatban és korábbi félévi feladatban láthattunk, de most inputként egy képet kapjon, és képet is adjon vissza, ehhez ismét a vele párban járó mandelbrot halmazt fogom használni.

Megadunk 3 értéket, vagyis 4 et, amiből az első az, hogy hány paramétert adunk meg, majd megadjuk az input kép méretét, a azt, hogy hány(256) értékre állítsa a perceptron a képet, és végül azt, hogy az outputként kapott kép mérete mekkora legyen. Így tehát teljesülni fog hogy változó argumentumszámú legyen a perceptron. A képet tömbbként használjuk, kiválasztjuk a piros részeit a képnek, elmentjük őket, és ezt felhasználva a kimeneti képet színezzük át.

```
Perceptron *p = new Perceptron(3, size, 256, size);  
  
double* image = new double[size];  
  
for(int i=0; i<png_image.get_width(); ++i)  
    for(int j=0; j<png_image.get_height(); ++j)
```

```
image[i*png_image.get_width()+j] = png_image[i][j].red;

double* newPicture = (*p) (image);

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

Mindehez az mlp.hpp fájlba is bele kell nyúlni, (ez tartalmazza ugye a perceptron felépítését), és megadni hogy több argumentumot is használhassunk a fentebbi módon.

```
class Perceptron
{
public:
    Perceptron ( int nof, ... )
    {
        n_layers = nof;

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

        va_list vap;

        va_start ( vap, nof );

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

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

        va_end ( vap );
    }

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

magamnak: képet ide ne felejtsd el beilleszteni !

Chapter 16

Helló, Gödel!

16.1 Gengszterek

Feladat: Gengszterek rendezése lambdaival a Robotautó Világbajnokságban

A lambda kifejezések segítségével név nélküli függvényeket készíthetünk. Ezeknek a visszatérési értékénél nem kell megneveznünk típust, csak ritka esetekben, a visszatérési értéket, a compiler kiértékeli, kivéve ritka esetekben. A lambda kifejezésekkel ahelyett hogy interfész implementációt készítenénk egy metódusnak, egy kód részletet adunk át neki. A szintaxis :

```
[ capture clause ] (parameters) -> return-type  
{  
    definition of method  
}
```

Figure 16.1: Lambda expression- from geeksforgeeks.org

A szöglletes zárójelek közötti rész az esemény amit figyelünk. A sima zárójelek közötti rész a paraméterek, a nyíl utána pedig a visszatérési érték.

A robotautó világbajnokságban a nekünk fontos részlet itt ez, ahol lambda expressionot használunk:

```
std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( unsigned x, ←  
    unsigned y )  
{  
    return dst ( cop, x ) < dst ( cop, y );  
} );
```

És hogy mi alapján fogunk rendezni, ami megmondja hogy melyik gengszterek vannak a legközelebb egy rendőrhöz.

```
void sort (RandomAccessIterator first, RandomAccessIterator last, Compare ←
comp);
```

16.2 STL map érték szerinti rendezése

Feladat:

<https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180>

Kezdjük azzal, hogy mi a map és mi az STL osztály. Az STL azt jelenti, hogy Standard Template Library, azaz egy szabványos sablon könyvtár. Általános programozási adatszerkezeteket és funkciókat tesz elérhetővé, listákat, tömböket, halmazokat, térképeket, stb. Témábaillően általános formában vannak megírva a sablonok, paraméterekkel, amiket majd a programozó helyettesít be. Az STL-nek négy osztálya van, Algorithms, Containers, Functions, Iterators. Az tárolók egyike a map, amit használnunk kell. A map-pal értékeket asszociáltathatunk egymással, párosíthatunk össze. Vannak kulcsok, és egy hozzájuk tartozó érték. Fontos, hogy nem lehet két értéknek ugyanaz a kulcsa, mert az zavarodottsághoz vezetne.

A feladatként megadott fénkyardban egy string kulcsokhoz int értékeket rendelünk.

```
( std::map <std::string, int> &rank )
```

Most érték szerint és nem kulcs szerint rendezünk, és egy sort- al ami az STL egy másik algoritmusá. Az első feladathoz hasonlóan itt is lambda-kifejezést használunk a rendezéshez.

```
std::vector<std::pair<std::string, int>> sort_map ( std::map <std::string, ←
int> &rank )
{
    std::vector<std::pair<std::string, int>> ordered;

    for ( auto & i : rank ) {
        if ( i.second ) {
            std::pair<std::string, int> p {i.first, i.second};
            ordered.push_back ( p );
        }
    }

    std::sort (
        std::begin ( ordered ), std::end ( ordered ),
        [ = ] ( auto && p1, auto && p2 ) {
            return p1.second > p2.second;
        } ;

    return ordered;
}
```

```
}
```

16.3 GIMP Scheme hack

A feladattal előző félévben már foglalkoztunk, egy krómozott szöveget iratunk ki, és mandalát készítünk. Lássuk is őket :

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

Megoldás videó: https://youtu.be/OKdAkl_c7Sc

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome_bhax_chrome3.scm

A feladatban az előző feladathoz hasonlóan GIMP-et használunk. A forrás:

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)

;(color-curve)

(define (elem x lista)

  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )

)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
)
```

```
(set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
(set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))

(list text-width text-height)
)
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-chrome text font fontsize width height color ←
    gradient)
(let*
  (
    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
        LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-width (car (text-wh text font fontsize)))
    (text-height (elem 2 (text-wh text font fontsize)))
    (layer2)
  )
  ;step 1
  (gimp-image-insert-layer image layer 0 0)
  (gimp-context-set-foreground '(0 0 0))
  (gimp-drawable-fill layer FILL-FOREGROUND)
  (gimp-context-set-foreground '(255 255 255))

  (set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
    ))
  (gimp-image-insert-layer image textfs 0 0)
  (gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ←
    height 2) (/ text-height 2)))

  (set! layer (car (gimp-image-merge-down image textfs CLIP-TO-BOTTOM- ←
    LAYER)))
  ;step 2
  (plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)

  ;step 3
  (gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)

  ;step 4
  (plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

  ;step 5
  (gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
```

```
(gimp-selection-invert image)

;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ←
    LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)

;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY GRADIENT- ←
    LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ←
        3)))
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

;(script-fu-bhax-chrome "Bátf41 Haxor" "Sans" 120 1000 1000 '(255 0 0) " ←
    Crown molding")

(script-fu-register "script-fu-bhax-chrome"
    "Chrome3"
    "Creates a chrome effect on a given text."
    "Norbert Bátfai"
    "Copyright 2019, Norbert Bátfai"
    "January 19, 2019"
    ""
    SF-STRING      "Text"          "Bátf41 Haxor"
    SF-FONT        "Font"          "Sans"
    SF-ADJUSTMENT  "Font size"    '(100 1 1000 1 10 0 1)
    SF-VALUE       "Width"         "1000"
    SF-VALUE       "Height"        "1000"
    SF-COLOR       "Color"         '(255 0 0)
    SF-GRADIENT    "Gradient"     "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
    "<Image>/File/Create/BHAX"
)
```

Egy beírt szövegre egy krómhatást szeretnénk alkalmazni, ami a következőképpen néz ki:



siker!

Ez a színezés:

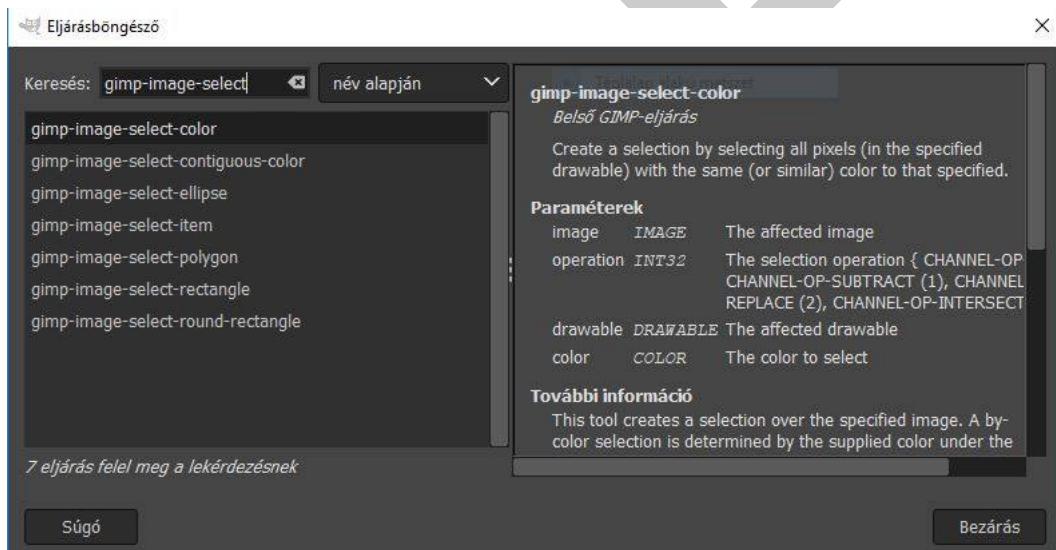
```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
```

```
(aset tomb 7 190)
tomb)
)
```

Egy program egy kiterjesztés a GIMP-hez, bemásolva a scriptek közé, a File-Create alatt egy új, "BHAX"-ként elnevezett lehetőség válik elérhetővé, így állítottuk be.

```
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)
```

A programban saját függvényeket is alkotunk a `define`-al, és a GIMP és a nyelvezet beépített függvényeit is használjuk. Ezeket a beépítetteket az Súgóból az eljárásböngészőt megnyitva megtekinthetjük, és leírást is kapunk róluk, pl.:



A kiterjesztést lehet a beállított File-Create-BHAX ról, és script-fu terminálba írva is használni. Script-fu:

```
(script-fu-bhax-chrome "Siker!" "Sans" 120 1000 1000 '(255 0 0) "Crown ←
molding")
```

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

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

Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala_bhax_mandala9.scm

A mandala egy szanszkrit eredetű szó, aminek a jelentése kör. A hinduizmust gyakorlók körében évezredek óta használt segédeszköz, és emellett szépsége miatt díszítőművészeti formában is elterjedt.

Elkészítése egy pár érdekes lépésből áll. Többféle betűtípusossal és rövidebb-hosszabb szöveggel dolgozhatunk, mindenféleképpen nagyon szép mandalákat alkothatunk. Most is a GIMP-et használjuk, a szöveget

középre leírjuk, vízszintesen tükrözzük és a tükrözöttet a szöveg tetejére illesztjük, a kettőt összeillesztjük. Az összeillesztett alakzatot lemásoljuk, középre illesztjük, és elforgatjuk 90 fokkal. Ezeket megint összeillesztjük, másoljuk, forgatjuk 45 fokkal, és ugyanez mégegyeszer, a végén 30 fokkal elforgatva. Egy kört kapunk, ezt még bekeretezzük, és kész a mandala.

Ezt a már előző feladatban megemlített függvényekkel megalkotjuk, integráljuk a script-et GIMP-be.

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)

(define (text-width text font fontsize)
(let*
  (
    (text-width 1)
  )
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  text-width
)
)

(define (text-wh text font fontsize)
(let*
  (
    (text-width 1)
    (text-height 1)
  )
  ;;;
  (set! text-width (car (gimp-text-get-extents-fontname text fontsize ←
    PIXELS font)))
  ;;; ved ki a lista 2. elemét
  (set! text-height (elem 2 (gimp-text-get-extents-fontname text ←
    fontsize PIXELS font)))
  ;;;
  (list text-width text-height)
)
)

; (text-width "alma" "Sans" 100)

(define (script-fu-bhax-mandala text text2 font fontsize width height color ←
  gradient)
(let*
  (
```

```
(image (car (gimp-image-new width height 0)))
  (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ←
    LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-layer)
    (text-width (text-width text font fontsize))
    ;;
    (text2-width (car (text-wh text2 font fontsize)))
    (text2-height (elem 2 (text-wh text2 font fontsize)))
    ;;
    (textfs-width)
    (textfs-height)
    (gradient-layer)
  )
(gimp-image-insert-layer image layer 0 0)

(gimp-context-set-foreground '(0 255 0))
(gimp-drawable-fill layer FILL-FOREGROUND)
(gimp-image-undo-disable image)

(gimp-context-set-foreground color)

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ←
  ))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (/ ←
  height 2))
(gimp-layer-resize-to-image-size textfs)

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate-simple text-layer ROTATE-180 TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
  -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 2) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
  -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
(gimp-item-transform-rotate text-layer (/ *pi* 4) TRUE 0 0)
(set! textfs (car (gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
  -LAYER)))

(set! text-layer (car (gimp-layer-new-from-drawable textfs image)))
(gimp-image-insert-layer image text-layer 0 -1)
```

```
(gimp-item-transform-rotate text-layer (/ *pi* 6) TRUE 0 0)
(set! textfs (car(gimp-image-merge-down image text-layer CLIP-TO-BOTTOM ←
-LAYER)))

(plug-in-autocrop-layer RUN-NONINTERACTIVE image textfs)
(set! textfs-width (+ (car(gimp-drawable-width textfs)) 100))
(set! textfs-height (+ (car(gimp-drawable-height textfs)) 100))

(gimp-layer-resize-to-image-size textfs)

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 22)
(gimp-edit-stroke textfs)

(set! textfs-width (- textfs-width 70))
(set! textfs-height (- textfs-height 70))

(gimp-image-select-ellipse image CHANNEL-OP-REPLACE (- (- (/ width 2) ←
(/ textfs-width 2)) 18)
(- (- (/ height 2) (/ textfs-height 2)) 18) (+ textfs-width 36) (+ ←
textfs-height 36))
(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(gimp-context-set-brush-size 8)
(gimp-edit-stroke textfs)

(set! gradient-layer (car (gimp-layer-new image width height RGB-IMAGE ←
"gradient" 100 LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image gradient-layer 0 -1)
(gimp-image-select-item image CHANNEL-OP-REPLACE textfs)
(gimp-context-set-gradient gradient)
(gimp-edit-blend gradient-layer BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ←
GRADIENT-RADIAL 100 0
REPEAT-TRIANGULAR FALSE TRUE 5 .1 TRUE (/ width 2) (/ height 2) (+ (+ (/ ←
width 2) (/ textfs-width 2)) 8) (/ height 2))

(plug-in-sel2path RUN-NONINTERACTIVE image textfs)

(set! textfs (car (gimp-text-layer-new image text2 font fontsize PIXELS ←
)))
(gimp-image-insert-layer image textfs 0 -1)
(gimp-message (number->string text2-height))
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text2-width 2)) (- (/ ←
height 2) (/ text2-height 2)))
```

```
; (gimp-selection-none image)
; (gimp-image-flatten image)

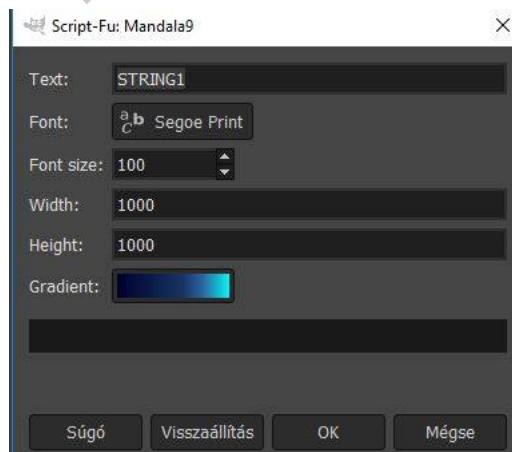
(gimp-display-new image)
(gimp-image-clean-all image)
)

;

; (script-fu-bhax-mandala "Bátfai Norbert" "BHAX" "Ruge Boogie" 120 1920 ←
 1080 '(255 0 0) "Shadows 3")

(script-fu-register "script-fu-bhax-mandala"
  "Mandala9"
  "Creates a mandala from a text box."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 9, 2019"
  ""
  SF-STRING      "Text"          "Bátf41 Haxor"
  SF-STRING      "Text2"         "BHAX"
  SF-FONT        "Font"          "Sans"
  SF-ADJUSTMENT   "Font size"    '(100 1 1000 1 10 0 1)
  SF-VALUE       "Width"         "1000"
  SF-VALUE       "Height"        "1000"
  SF-COLOR       "Color"         '(255 0 0)
  SF-GRADIENT    "Gradient"     "Deep Sea"
)
(script-fu-menu-register "script-fu-bhax-mandala"
  "<Image>/File/Create/BHAX"
)
```

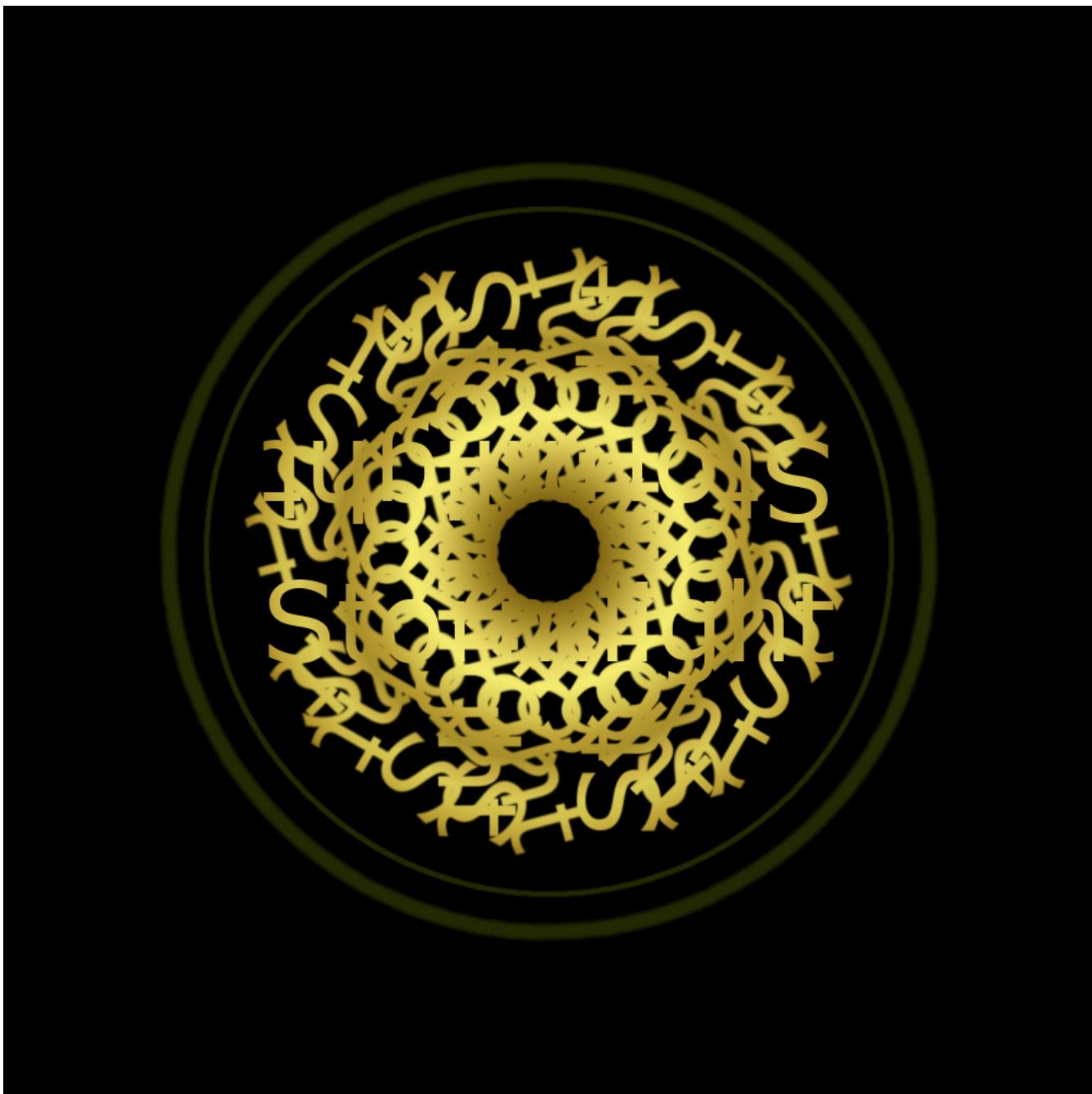
Az alul elhelyezkedő rész felelős azért, hogy ne kelljen script-fu konzolt használni, a felhasználó kiválaszthatja a létrehozásnál a Mandala9 névvel illetett opciót, és egy interaktív felületet fog látni maga előtt amivel a szöveget, a betűtípusat, méreteket, színezést, átmenetet ki tudja választani:



Storm feliratból mandala:



Stormlight feliratból mandala:



16.4 Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang Interface Comparable T szerepét!

Forrás: [Wiki2Matrix.java](#)

[AlternativTabella.java](#)

A feladathoz egy tabellát kaptunk a 2010-11 es magyar labdarúgó-bajnokság adatairól, egy kereszttáblázat formájában azt, hogy az egymással versengő csapatok hogyan teljesítettek mind hazai, mind vendég terépen.

Activities Firefox Web Browser •

2010–2011-es magyar labdarúgó-bajnokság (első osztály) – Wikipédia - Mozilla Firefox

És mégis elindult a tavas... 2010–2011-es magyar labdarúgó-bajnokság (első osztály)

https://hu.wikipedia.org/wiki/2010–2011-es_magyar_labdarúgó-bajnokság_(első_osztály)

Kereszttáblázat [szerkesztés]

Hazai \ Vendég ¹	SIÓ	HON	VAS	DEB	FTC	ETO	KAP	KTE	PÁP	MTK	PAK	SZÓ	HAL	ÚJP	VID	ZTE
BFC Siófok		1-3	0-0	4-1	1-1	0-1	0-0	1-2	0-1	0-0	1-3	1-1	0-0	1-1	1-1	0-4
Budapest Honvéd	0-1		0-0	1-0	0-1	1-1	1-0	1-2	2-4	1-2	1-0	0-0	3-1	1-4	2-2	1-0
Vasas	3-0	3-2		1-5	1-3	2-1	1-3	1-1	1-1	1-1	2-3	3-0	2-1	1-0	0-0	2-0
Debrecenti VSC	2-3	2-2	3-1		2-1	1-1	3-1	6-2	2-0	3-1	2-1	4-0	1-2	1-1	3-1	2-1
Ferencváros	1-2	1-3	0-1	1-1		3-0	1-0	2-1	3-0	3-0	2-1	1-0	2-1	1-0	0-5	4-4
Györi ETO	1-0	3-0	0-1	3-0	1-0		2-0	2-1	0-1	1-1	1-1	4-2	2-2	2-1	1-1	0-1
Kaposvári Rákóczi	3-0	0-0	0-1	0-0	2-1	3-0		2-1	3-2	2-1	1-2	0-1	1-0	0-1	1-4	2-1
Kecskeméti TE	2-3	2-1	3-1	3-0	1-2	3-3	4-1		0-1	3-0	0-1	4-2	1-0	4-3	2-4	1-0
Lombard Pápa	0-1	0-1	2-1	1-1	0-5	2-1	1-1	4-1		0-2	1-2	0-0	5-1	3-1	1-2	4-3
MTK Budapest	1-2	3-1	4-0	0-0	1-3	0-0	0-2	4-2	2-1		3-4	1-0	0-3	1-0	0-3	3-4
Paksi FC	0-0	0-1	3-0	2-2	3-2	2-1	2-3	2-0	4-0	1-1		3-1	2-1	2-1	1-0	2-2
Szolnoki MÁV FC	0-4	0-2	0-1	1-2	2-3	0-3	1-2	2-1	2-2	2-1	3-1		0-0	1-0	1-1	1-3
Szombathelyi Haladás	4-0	1-1	3-0	3-0	1-1	3-3	2-0	1-0	1-0	2-0	1-2	3-1		0-2	2-0	0-0
Újpest	1-1	3-1	2-2	2-2	6-0	0-0	3-2	2-1	2-1	2-1	2-3	1-0	3-1		1-0	4-2
Videoton	2-0	0-2	3-0	2-1	1-1	2-1	3-1	2-2	4-0	2-1	2-1	3-1	3-1	1-0		3-0
Zalaegerszegi TE	2-1	2-1	2-1	1-1	2-1	1-1	3-5	2-1	3-1	1-0	1-0	2-1	1-1	2-1	1-2	

Utolsó felvett mérkőzés dátuma: 2011. május 22.

Ezt a táblázatot bedolgozzuk tömb formájában egy java fájlba, és aszerint írjuk le az eredményeket, ahogy a képen látható rublikák milyen színnel vannak kitöltve. A képen sajnos nem igazán látszik a piros és a sárga közötti eltérés, de ha megvizsgáljuk, akkor láthatjuk hogy különböző szinekről van szó.

Kereszttáblázat [szerkesztés]

Hazai \ Vendég ¹	SIÓ	HON	VAS	DEB	FTC	ETO	KAP	KTE	PÁP	MTK	PAK	SZÓ	td	37 × 25	ZTE
BFC Siófok		1-3	0-0	4-1	1-1	0-1	0-0	1-2	0-1	0-0	1-3	1-1	1-1	1-1	0-4
Budapest Honvéd	0-1		0-0	1-0	0-1	1-1	1-0	1-2	2-4	1-2	1-0	0-0	3-1	1-4	2-2
Vasas	3-0	3-2		1-5	1-3	2-1	1-3	1-1	1-1	1-1	2-3	3-0	2-1	1-0	0-0
Debrecenti VSC	2-3	2-2	3-1		2-1	1-1	3-1	6-2	2-0	3-1	2-1	4-0	1-2	1-1	3-1
Ferencváros	1-2	1-3	0-1	1-1		3-0	1-0	2-1	3-0	3-0	2-1	1-0	2-1	1-0	0-5
Györi ETO	1-0	3-0	0-1	3-0	1-0		2-0	2-1	0-1	1-1	1-1	4-2	2-2	2-1	1-1
Kaposvári Rákóczi	3-0	0-0	0-1	0-0	2-1	3-0		2-1	3-2	2-1	1-2	0-1	1-0	1-4	2-1
Kecskeméti TE	2-3	2-1	3-1	3-0	1-2	3-3	4-1		0-1	3-0	0-1	4-2	1-0	4-3	2-4
Lombard Pápa	0-1	0-1	2-1	1-1	0-5	2-1	1-1	4-1		0-2	1-2	0-0	5-1	3-1	1-2
MTK Budapest	1-2	3-1	4-0	0-0	1-3	0-0	0-2	4-2	2-1		3-4	1-0	0-3	1-0	0-3
Paksi FC	0-0	0-1	3-0	2-2	3-2	2-1	2-3	2-0	4-0	1-1		3-1	2-1	2-1	1-0
Szolnoki MÁV FC	0-4	0-2	0-1	1-2	2-3	0-3	1-2	2-1	2-2	2-1	3-1		0-0	1-0	1-1
Szombathelyi Haladás	4-0	1-1	3-0	3-0	1-1	3-3	2-0	1-0	1-0	2-0	1-2	3-1		0-2	2-0
Újpest	1-1	3-1	2-2	2-2	6-0	0-0	3-2	2-1	2-1	2-1	2-3	1-0	3-1		1-0
Videoton	2-0	0-2	3-0	2-1	1-1	2-1	3-1	2-2	4-0	2-1	2-1	3-1	3-1	1-0	
Zalaegerszegi TE	2-1	2-1	2-1	1-1	2-1	1-1	3-5	2-1	3-1	1-0	1-0	2-1	1-1	2-1	

Utolsó felvett mérkőzés dátuma: 2011. május 22.

Az értékek szinezéstől függően:

- Üres = 0
- Zöld = 1

- Sárga = 2
 - Piros = 3

```
int[][] kereszt = {
    {0, 0, 0, 1, 0, 3, 2, 3, 3, 2, 0, 0, 0, 0, 2, 2, 3},
    {3, 0, 2, 1, 3, 2, 0, 3, 3, 3, 0, 0, 0, 0, 0, 0, 1},
    {1, 1, 0, 0, 3, 1, 3, 0, 0, 0, 3, 1, 1, 0, 2, 0},
    {0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 2, 1, 1},
    {3, 3, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0},
    {1, 0, 3, 1, 0, 0, 0, 1, 3, 2, 0, 0, 0, 1, 2, 3},
    {0, 2, 0, 0, 0, 1, 0, 1, 1, 0, 3, 0, 1, 3, 3, 1},
    {0, 0, 1, 1, 3, 0, 0, 0, 0, 1, 3, 1, 1, 1, 3, 0},
    {0, 0, 1, 2, 3, 0, 0, 1, 0, 0, 2, 1, 1, 3, 1},
    {0, 1, 1, 2, 0, 0, 3, 1, 1, 0, 0, 0, 0, 1, 3, 3},
    {2, 3, 0, 2, 1, 1, 0, 0, 1, 2, 0, 1, 0, 0, 0, 2},
    {3, 3, 0, 0, 0, 3, 3, 0, 2, 1, 1, 0, 2, 0, 0, 0},
    {1, 2, 0, 1, 0, 2, 1, 0, 0, 1, 3, 1, 0, 0, 0, 2},
    {2, 1, 2, 0, 1, 2, 1, 0, 0, 0, 3, 1, 1, 0, 0, 0},
    {1, 3, 1, 0, 2, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0},
    {0, 0, 1, 0, 1, 0, 3, 1, 1, 0, 0, 1, 2, 1, 3, 0}
};
```

Lefuttatjuk a Wiki2Matrixot, és megkapunk egy link mátrixot:

Ezt az eredményt felhasználva lett elkészítve az AlternativTabella.java program, amiben keresendő is a feladatunk, hogy magyarázzuk el a java.lang Interface Comparable T szerepét.

A Comparable interfészről arra használhatjuk, hogy egy általunk definiált osztályt rendezzünk vele. Ahogy látható, a java.lang csomagban van , és kis rákeresés után megtudható, hogy egyetlen metódusa van, a compareTo (Object obj) . Ez arra szolgál, hogy egy aktuális objektumot viszonyítsunk egy másik objektumhoz. Háromféle visszatérési értéke van,

- Pozitív integer, akkor, ha a jelenlegi objektum nagyobb a megjelölt objektumnál
- Negatív integer, akkor, ha a jelenlegi objektum kisebb a megjelölt objektumnál.
- Nulla, akkor, ha a két objektum egyenlő.

Ez a kódban így van implementálva:

```
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;  
        }  
    }  
}
```

Felmerülhet a kérdés, hogy miért implements, és nem a megszokott extends kulcsszó van használva. Az extends kulcsszót használva a gyermek örökli a szülő interfész metódusait és attribútumait. Az implements kulcsszó használatával hozzáférünk egy interfész metódusaihoz , a felépítése az implementáló osztálynak pedig az interfészben van meghatározva.

Chapter 17

Helló, !

17.1

Feladat:

17.2

Feladat:

17.3

Feladat:

17.4

Feladat:

17.5

Feladat:

DRAFT

Chapter 18

Helló, Schwarzenegger!

18.1

Feladat:

18.2

Feladat:

18.3

Feladat:

18.4

Feladat:

18.5

Feladat:

DRAFT

Chapter 19

Helló, Calvin!

19.1

Feladat:

19.2

Feladat:

19.3

Feladat:

19.4

Feladat:

19.5

Feladat:

DRAFT

Chapter 20

Helló, Berners-Lee!

20.1 1.hét - Python

Megadott könyv:

- *Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípus-fejlesztés Python és Java nyelven*

A Python egy magas szintű, dinamikus, objektumorientált és platformfüggetlen programozási nyelv. Ezzel a nyelvvel egyszerű és komplex dolgokat is egyaránt, effektíven elkészíthetünk, kiemelkedően prototípusfejlesztésre, tesztelésre használható, a tanulási idő az átlagosnál kevesebb az egyszerű szintaksisból eredően. Ami mutatja, hogy fejlesztésre alkalmas a nyelv, az az, hogy nincs szükség a például C-ben vagy Java-ban megszokott fordításra/tesztelésre/újrafordításra(...), elég megadni a forrást és a Python értelmezője automatikusan lefuttatja a programot.

Ehhez hasonlóan, a programírás során elhagyható a változók típusainak a megadása, az értékek alapján a rendszer magától a lehetséges, megfelelő típust rendeli hozzájuk. Az adattípusok hasonlóak a C,C++ és Javahoz, és vannak tuple, dictionary és list típusok is. Adattípustól függően számos műveletet el lehet végezni rajtuk, sok metódus áll a rendelkezésünkre. Pythonban minden adatot objektumok képviselnek. Javahoz hasonlóan, de C++-tól eltérően, garbage collectort használ a memória kezelésre.

Természetesen használhatunk for, if, while ciklusokat, definiálhatunk függvényeket, írhatunk osztályokat. A nyelv a fejlesztés támogatása érdekében tartalmaz szabványos modulokat, amik mobilfejlesztés során nagyon hasznosak, igaz, mi a könyv ezen részére nem fókuszálunk, csak magára a Pythonra.

A C++ ban megszokott objektumorientált eljárást is támogatja a nyelv, készíthetünk osztályokat, a benne lévő objektumok a példányai annak az osztálynak. Ha valamit változtatunk az osztályban, például egy változót, akkor az megváltoztatja az osztályaiban is a változót. Ezt lehet akár csak példányszinten kivitelezni. Példa a könyvből:

```
class T #osztályszinten
    x=1
    ..
    ..
    T.x = 2 #példányszinten
```

Metódusokat ugyanúgy definiálunk mint globális függvényeket, csak egy extra self paraméterrel. Pythonban az osztályokban lehet egy speciális, konstruktur tulajdonságú metódusa, ez a `__init__`.

Tartalmaz a nyelv szabványos modulokat és a mobilfejlesztéshez szükséges modulokat is, pl:

- appuifw
- messaging
- sysinfo

20.2 1.hét - A Java nyelvről C++ programozóknak-1.

Megadott könyvek:

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

II 420. A fejezetben a két nyelv közötti hasonlóságokról és eltérésekről olvashatunk, amiket példákkal is szemlélte a könyv. --- objektumorientáltság ---mutatük,referenciák ---sok apró különbség van még a két nyelv között, amit példákkal le is ír a könyv.

A Java nyelv a C és C++ nyelvek szintaxisát vette sajátja alapjául, így sok hasonló kifejezés van, de nem minden ugyanaz a jelentés. Nagyobb alkalmazási területet lefed a C++ -nál, amiknek a kezelésére ott is lehetőség van, csak extra könyvtárak hozzáadásával.

A Java a C++ -tól eltérően kezeli a program fordítását, a forrást először bájkóddá alakítja, amit egy java virtuális környezetben használunk. Így hordozható kódokat írhatunk, amik több rendszeren használhatóak. Ennek érdekében szigorú feltételeket szab meg a Java, a C++ pedig arra törekszik, hogy egy adott környezetben a lehető leghatékonyabb kódot lehessen megírni.

Különbség az is, hogy miként kezeljük a memóriát. A C++ nyelvben az objektumokat, mint a memória egy összefüggő területén elhelyezkedő bájtsorozatokat fogja fel, ami ismert memóriakiosztással rendelkezik, és amit ennek megfelelően manipulál a lefordított program, használhatunk mutatókat.. A Javában a memóriakezelést egy garbage collector végzi, ami a következő;következőképpen működik. A nyelvnek egy beépített eszköze, lehetővé teszi hogy új objektumokat hozzunk létre és ne kelljen a memória allokálással és feloldással mauálisan játszanunk, automatikusan újraosztja a használható memóriát. A collector összegyűjt és kiszórja a "halott" objektumokat, azaz azokat amiknek nincs referencia, nincsenek felhasználva. Vagyis, sokkal inkább a használt tárgyakat számon tartja, és minden másat szemétnek jelöl meg.

A Javában nincsenek külön objektumok és mutatók, pointerek. Ezenkívül nincsenek csak alapértelmezett operátorok, más a konstans(const) jelentése, és még számos más kisebb nagyobb eltérés van, amiket példákkal is megmutat a könyv.

20.3 2.hét- Öröklődés, osztályhierarchia. Polimorfizmus, metódustúlterhelés. Hatáskörkezelés. A bezárási eszközrendszer, láthatósági szintek. Absztrakt osztályok és interfések.

Java nyelvben a legkisebb egységek az osztályok. Egy osztály azonos típusú egységek szerkezetét írja le. Példaként lehet gondolni az "autó"-ra mint osztályra, aminek van színe, típusa, modellje, gyártója, etc, ezek a változói. Az osztálynak több példánya is létrehozható, ezeknek a változói eltérhetnek, ezért példányváltozó a nevük. Ha az adott osztály public, akkor az hozzáérhető más csomagokból(import package-el). Ha nem public, akkor az osztályt csak a saját csomagján belül használhatjuk. Alapértelmezetten az osztályok egy névtelen csomagba kerülnek fordításkor. ... 6. csomagok? .. Egy osztály fejlécből és törzsből áll. Az osztály fejlécében kell megadni, hogy az osztályt melyik más osztály kiterjesztéseként definiáljuk, és azt, hogy melyik interfésekkel valósítja meg. A törzsében szerepelnek az osztály tagjai, azaz a változói és metódusai. A változók általában főnevek, a metódusok igék, és vannak konstans változók(final módosítóval ellátva), amelyek csupa nagybetűből állnak.

Metódustúlterhelés

Egy metódus feje megadja a metódus visszatérési típusát, azonosítóját, és zárójelek között, egymástól vesszővel elválasztva a formális paramétereit. Példa a könyvben:

```
boolean kozepesJovedelmu(int minimum, int maximum) {  
    ..  
    ..  
}
```

Egy osztály több metódusának is lehet ugyanaz a neve, azért, mert egy metódust a szignatúrája azonosítja. Tehát a szignatúrának eltérőnek kell lennie, azaz a formális paramétereinek(mint az int minimum, int maximum..) száma és/vagy típusa más kell legyen. Ha egy metódusnevét többször is használunk más szignatúrával, akkor azt metódustúlterhelésnek nevezzük. Ekkor a program fordítási időben fogja eldönteni, hogy melyiket kell használni, attól függően hogy milyen formális paramétereket adtunk meg. A könyvben erre is van egy példa:

```
public class Alkalmazott {  
    void fizetestEmel(int novkemeny) {  
        fizetes +=novekmeny;  
    }  
    void fizetestEmel() {  
        fizetes += 5000;  
    }  
    void fizetestEmel(Alkalmazott masik) {  
        if (kevesebbetKeresMint(masik)) fizetes = masik.fizetes;  
    }  
}
```

Láthatósági szintek, hatáskörkezelés.

Egy osztály nem hivatkozhat bármely más osztály tagjaira, különben nagy kavalkád lenne az egész,nem lenne átlátható a rendszer. Hogy ezt elkerüljük, el kell rejtenünk a változókat és metódusokat, így biztosíthatjuk hogy csak azok lássák más osztályok tagjait, akiknek meg szeretnénk engedni, akiknek szüksége van rá.

Tehát, a láthatóságát lehet és kell beállítani a tagoknak. Vannak :

- *Félnyilvános tagok*

Ha nem adunk meg módosítót egy tagnak, akkor azt félnyilvános tagnak nevezzük, és ezeket a tagokat csak az azonos csomagban definiált osztályok érhetik el.

- *Nyilvános tagok*

Ha public módosítóval látunk el egy tagot, akkor azt nyilvánosnak, publikusnak nevezzük, az ilyen tagokat minden osztály, így más csomagban lévő osztályok is el tudják érni. Ilyen módon kontrollálni tudjuk, hogy egy másik osztály csak bizonyos részeit láthassák egy adott osztálynak.

- *Privát tagok*

A private módosítóval rendelkező változókat és metódusokat csak az adott osztály látja, azaz az osztály metódusai és inicializátorai.Fontos, hogy az ilyen privát tagokat az az osztály metódusai ahol azok definiálva vannak,akkor is elérik, ha az egy példányosított osztály tagja.

- *Protected tagok*

A protected tagok az előzőektől eltérően figyelembe veszik az öröklődési viszonyokat, : "Egy más csomagban definiált X osztály akkor férhet hozzá egy protected taghoz, ha a kérdéses tagot definiáló osztálynak leszármazottja, és a tagra minősíts nélkül hivatkozik, vagy olyan minősítéssel, amelynek típusa az X osztály vagy annak leszármazottja."

Vannak osztályváltozók és osztálymetódusok, amelyek csak az adott osztályhoz tartoznak, példányaihoz nem , ezeket nevezik statikus tagoknak, mert a static jelölést használják:

```
..  
..  
public class Alkalmazott {  
    static int nyugdíjKorhatár = 60;  
..  
..
```

Számomra nagyon hasznos mondat volt a könyvben a következő: "A példányváltozók a példány állapotát rögzítik, az osztályváltozók az osztályét."

A bezáras eszközrendszerei

A korábban megemlített szemétygyűjtő mechanizmusnak köszönhetően a legtöbb esetben nem kell egy objektum megszüntetésével törölnünk, mint például C++-ban a destruktorkor használatával. Azonban, vannak olyan esetek, ahol muszáj saját magunknak megtenni ezt, erre van kitalálva a `finalize()` amit `protected`, `void` típusú, paraméter nélküli metódusok esetén lehet használni. Osztályok esetében ennek a metódusnak a neve `classFinalize()`.Ennek a törzsében kell a szükséges kódot elhelyezni.

Öröklődés,osztályhierarchia,Poliformizmus

Egy meglévő osztályt ki is lehet egészíteni utólag is. A kiegészítéshez az extends-t használjuk:

```
public class Fonok extends Alkalmazott {  
    ..  
    ..
```

A példában így a Fönök az Alkalmazott osztály kiterjesztése, ebben az új osztályban új változókat és metódusokat is be tudunk vezetni. Ekkor a Fönök osztályt az Alkalmazott osztály gyermekének, az Alkalmazott pedig a Fönök ősének nevezzük. A gyermekosztály természetesen rendelkezik a saját osztályában megadott változókkal és metódusokkal, ezen felül pedig az ősével is, ezeket örököli. Hozzáérni viszont nem feltétlen fér hozzá mindegyikez, csak a public tagokhoz, és a félnyilvános tagokhoz(amiknek nincsen kategóriája megadva) akkor, ha azt a tagot leíró osztállyal egy csomagban van a gyermek.A private tagokhoz pedig nem fér hozzá. Vannak még protected tagok, ezek a félnyilvános kategóriák kiterjesztései:"Egy más csomagban definiált X osztály akkor férhet hozzá egy protected taghoz, ha a kérdéses tagot definiáló osztálynak leszármazottja, és a tagra minősíts nélkül hivatkozik, vagy olyan minősítéssel, amelynek típusa az X osztály vagy annak leszármazottja." A konstruktorkat nem örököli a gyermek,ezért ha a szülő egy konstruktort akarjuk meghívni , akkor a super kulcsszót kell használnunk a this helyett. Attól függetlenül hogy a gyermek látja vagy sem az őse tagjait, örököli őket.

Mivel egy gyermek örököli az őse összes változóját és metódusát, ezért a gyermeket minden olyan környezetben lehet használni, ahol az őst is lehetne.(Ugye mivel minden meg lehet tenni a gyermekkel amit az őssel is, fordítva viszont már nem feltétlen lenne igaz a gyermek saját tagjai miatt.) Tehát, ha valahol egy "Ós" típusú változót kellene megadni, ott megadhatunk egy "Ósgyermek" típusú változót is. Ezt a tulajdonságot nevezzük poliformizmusnak, és ez a típuskonverziónak lehetőségének köszönhető.

A rokonsági kapcsolatokat osztályhierarchiának nevezzük. Van egy kiemelt osztály, az Object, amely a java.lang csoport része, és ez implicit módon minden osztálynak az őse, amelyeket nem kiterjesztésel(extends) hoztunk létre. Ez az osztály írja le minden más osztály és tömb metódusait, amit használni lehet.

Absztrakt osztályok és interfések

Az őssosztályokra tekinthetünk úgy, mint azokra amik megalkotnak egy interfész, egy olyan felületet, amely egységesen kezelhetővé teszi a leszármazott osztályokat. Léteznek absztrakt osztályok, amiket az abstract módodítóval jelölünk, ezek különlegessége, hogy tartalmazhatnak absztract, törzs nélküli metódusokat.(ezek az abstract metódusok nem lehetnek private/final/static résszel ellátni, mert az előzőeket figyelembe véve, ahhoz nem férhetne hozzá a gyermek, és nem tudná majd felülírni) Ezeket az abstract osztályokat nem lehet példányosítani, mert úgy a példányokra nem lenne értelmezve minden metódus. Először ki kell őket terjeszteni, és a kiterjesztésben felül lehet írni a metódusokat. Nem kötelező minden metódust felülírni, de ha marad abstract metódus, akkor a gyermekosztály is abstract marad, és azt se lehet példányosítani egyből.

Az interfész egy referencia típus, ami absztrakt metódusok deklarációjának és konstans értékeknek az összessége. Az interfésekben a metódusokat deklaráljuk, de nem lesznek megvalósítva, majd csak a kiterjesztés után. Az interfész egy felület, egy útmutatás, hogy mit implementálunk, amitől el lehet térni, ez egy absztraktiós eszköz. Az interfész igazából akkor használunk fel, amikor egy osztályt származtatunk belőle. Ha az összes , az interfészben szereplő metódust felhasznál egy osztály, akkor azt mondjuk hogy im-

plementálja az interfész. A könyv példának a java.lang csomag Runnable , és a java.io csomag Serializable interfészeit hozza fel.

DRAFT

Part IV

Irodalomjegyzék

DRAFT

20.4 Általános

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

20.5 C

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

20.6 C++

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

20.7 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 tanulós-zuba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett 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.