

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i>		
	Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Nagy, Martin	2019. december 2.	

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

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	9
2.6. Helló, Google!	10
2.7. 100 éves a Brun téTEL	12
2.8. A Monty Hall probléma	13
3. Helló, Chomsky!	15
3.1. Decimálisból unárisba átváltó Turing gép	15
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	16
3.3. Hivatalos nyelv	16
3.4. Saját lexikális elemző	17
3.5. l33t.l	18
3.6. A források olvasása	20
3.7. Logikus	21
3.8. Deklaráció	22

4. Helló, Caesar!	24
4.1. double ** háromszögmátrix	24
4.2. C EXOR titkosító	25
4.3. Java EXOR titkosító	26
4.4. C EXOR törő	27
4.5. Neurális OR, AND és EXOR kapu	29
4.6. Hiba-visszaterjesztéses perceptron	31
5. Helló, Mandelbrot!	33
5.1. A Mandelbrot halmaz	33
5.2. A Mandelbrot halmaz a std::complex osztállyal	35
5.3. Biomorfok	37
5.4. A Mandelbrot halmaz CUDA megvalósítása	40
5.5. Mandelbrot nagyító és utazó C++ nyelven	41
5.6. Mandelbrot nagyító és utazó Java nyelven	46
6. Helló, Welch!	47
6.1. Első osztályom	47
6.2. LZW	49
6.3. Fabejárás	51
6.4. Tag a gyökér	52
6.5. Mutató a gyökér	66
6.6. Mozgató szemantika	80
7. Helló, Conway!	81
7.1. Hangyszimulációk	81
7.2. Java életjáték	81
7.3. Qt C++ életjáték	89
7.4. BrainB Benchmark	89
8. Helló, Schwarzenegger!	90
8.1. Szoftmax Py MNIST	90
8.2. Mély MNIST	90
8.3. Minecraft-MALMÖ	90

9. Helló, Chaitin!	91
9.1. Iteratív és rekurzív faktoriális Lisp-ben	91
9.2. Gimp Scheme Script-fu: króm effekt	91
9.3. Gimp Scheme Script-fu: név mandala	91
10. Helló, Gutenberg!	92
10.1. Programozási alapfogalmak	92
10.2. Programozás bevezetés	92
10.3. Programozás	92
III. Második felvonás	93
11. Helló, Arroway!	95
11.1. OO szemlélet	95
11.2. „Gagyi”	98
11.3. Yoda	100
12. Helló, Liskov!	102
12.1. Liskov helyettesítés sértése	102
12.2. Szülő-gyerek	104
12.3. Ciklomatikus komplexitás	106
13. Helló, Mandelbrot!	108
13.1. Forward engineering UML osztálydiagram	108
13.2. Egy esettan	110
13.3. BPMN	111
14. Helló, Chomsky!	113
14.1. Encoding	113
14.2. OOCWC lexer	114
14.3. l334d1c45	117
15. Helló, Stroustrup!	121
15.1. String osztály előkészítése	121
15.2. Hibásan implementált RSA törése	123
15.3. Összefoglaló	127

16. Helló, Gödel!	128
16.1. Gengszterek	128
16.2. STL map érték szerinti rendezése	128
16.3. Alternatív Tabella rendezése	130
17. Helló, !	132
17.1. FUTURE tevékenység editor	132
17.2. OOCWC Boost ASIO hálózatkezelése	133
17.3. BrainB	134
18. Helló, Schwarzenegger!	135
18.1. Port scan	135
18.2. AOP	136
18.3. Junit teszt	137
19. Helló, Calvin!	139
19.1. MNIST	139
19.2. CIFAR-10	140
19.3. Android telefonra a TF objektum detektálója	141
20. Helló, Berners-Lee!	144
20.1. C++ és Java összehasonlítás	144
20.2. Python	145
IV. Irodalomjegyzék	146
20.3. Általános	147
20.4. C	147
20.5. C++	147
20.6. Lisp	147

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

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

Ebben a példában egy végtelen ciklust láthatunk aminek a "for"-os sora miatt 100%-os a CPU leterhelés 1 mag esetén mivel szünet nélkül fut. A **top** segítségével tudjuk ellenőrizni, hogy jól csináltuk-e meg a feladatot.

```
int main()
{
for(;;);
sleep(123);
}
```

Hasonlóan az előző példához egy végtelen ciklus van csak hozzá lett téve egy "sleep" funkció ami szünetelteti adott(pl.: 123 másodperc jelen esetben) időre a programot ezáltal 0%-ra csökken a CPU terhelés 1 mag esetén.

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

Itt a feladat hasonló az 1. példához, annyiban tér el, hogy az összes magon 100%-nak kell lennie a CPU terhelésnek. A kódba beírt "#pragma omp parallel" engedélyezi, hogy több szál fusson egyszerre. Ezen kívül még van két dolog egyik a "#include omp.h" a tetején illetve egy egy kapcsolót kell a végére tenni a fordításnál "-fopenmp". Ezeket teljesítve az összes magon 100%-os CPU terhelést fogunk kapni.

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

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

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{
    boolean Lefagy(Program P)
    {
        if (P-ben van végtelen ciklus)
```

```
    return true;
else
    return false;
}

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

main(Input Q)
{
    Lefagy2(Q)
}

}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

A T100-as program bemenetnek fog kapni egy másik programot aminek el kell döntenie, hogy van-e benne végtelen for ciklus vagy nincs. Ezután a T1000-es program ugyan ezt megcsinálja és megvizsgálja, hogy az adott kódban van-e végtelen for ciklus vagy nincs. Ha igaz értéket ad megáll ha hamisat akkor végtelen for ciklusba kerül. Ezáltal ilyen programot eddig feltételezhetően lehetetlenség megírni, de egy ilyen program megírása sok gondunkra adna megoldást.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

```
#include <iostream>
```

```
int main ()
{
    int a = 7;
    int b = 1;
    std::cout<<"Jelenleg : a:"<< a << " " "b:" << b << std::endl;
    b = b + a;
    a = b - a;
    b = b - a;
    std::cout<<"Utánna : a:"<< a << " " "b:" << b << std::endl;
}
```

Tegyük fel, hogy az eredeti $a=7$, az eredeti $b=1$. $b = 1 + 7 \rightarrow 8$ | $a = 8 - 7 \rightarrow 1$, az eredeti b | $b = 8 - 1 \rightarrow 7$, az eredeti a) "cout" sorral pedig kiíratjuk a szöveget illetve az "a" és a "b" értékeket. Ez szimpla logika.

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 forrása:

```
int main (void)
{
    WINDOWS *ablak;
    ablak = initscr ();

    int x = 0;
    int y = 0;
    int xnov = 1;
    int ynov = 1;
    int mx;
    int my;

    for (;;)
    {
        getmaxyx ( ablak,my,mx );
        mvprintw ( y,x,"O" );

        refresh ();
        usleep ( 100000 );
        clear ();

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

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

Az első sorban definiáljuk, hogy az *ablak Window pointer. Ezután egy végtelen ciklust hozunk létre amiben az első sor(getmaxxyz) tartalmazza azt a területet ahol a labda pattogni fog. Ezt követően a második sor(mvprintw)-vel íratjuk ki koordinántáként a labdánkat az ablakba. Az "usleep" sor egy kis finomítás, hogy jobban látszódjon az eredmány mivel felfüggeszti a program futását adott időre. Ezek után az if-ek segítségével "írányítjuk" a labdát. Például ha balra felfelé megy a labda és eléri az ablak szélét ez esetben jobbra felfelé fog menni, de ha jobbra lefelé megy és eléri a végpontot akkor balra lefelé fog.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás forrása:

```
int main
{
    int szohossz = 0;
    int szo = 1;
    do
    {
        szohossz++;
    }
    while(szo<<=1);
    printf("A szó %d bites\n"; szohossz)
    return 0;
}
```

Bev. progon használt bit shifletésről van ebben a feladatban szó azaz addig shiftelünk míg eltünik az 1-es és csak nullákat fog tartalmazni. Közben számolva, hogy ez milyen hosszú. A fő folyamat a "while"-ban látható mikor is balra haladva shiftelünk egyet.

2.6. Helló, Google!

Írj olyan C programot, amely egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét! Ennél a feladatnál Dékány Róbert Zsolt tutoriált.

Megoldás videó:

```
#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] += 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 semmirre sem mutat:\n");
pagerank(L1);

printf("\nAmikor az egyik oldal csak magára mutat:\n");
pagerank(L2);
```

```
    printf("\n");

    return 0;
}
```

Ennek a feladatnak a kitalálója Larry Page volt elsősorban ezáltal is lett a feladat lényegének az elnevezése azaz a PageRankolás. Ez egy google álltal kifejlesztett algoritmus ami fél publikus fél publikus nem. Nem olyan bonyolult, de picit összetett és relatív ez az egész. Nevéből kiindulva azt vizsgálja, hogy milyen fontos egy adott oldalt. Józan paraszti ésszel ezt úgy néznénk, hogy ahány oldal mutat a kijelölt oldalunkra annál fontosabb de ez tévhiz. A nagyobb oldalból mutató oldalak nagyobb értékkel bírnak ezáltal lehet, hogy 1 neves oldalról mutató link álltali oldal többet ér mint 10db szenny oldalról mutató oldal.

2.7. 100 éves a Brun tétele

Írj R szimulációt a Brun tétele demonstrálására!

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

Ennél a feladatnál Dékány Róbert Zsolt tutoriált.

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

```
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")
```

Ehez a feladathoz szükséges lesz egy kiegészítő mégpedig a matlab.Majd létrehozunk egy függvényt(x).Elsőként egy olyan listát fog adni "x"-ig ami tartalmazza az összes prímet.Ezt követően kivonást végez addig ameddig nem talán olyan párt aminek a különbsége 2(azaz ikerprím).Legvégül pedig visszaadja a reciprokosszegüket. A függvény meghatározása után már nincs más dolgunk minthogy ábrázoljuk ezt a függvényt. x tengelyt felosztjuk 13-tól 1 000 000-ig 100 000 különbséggel. y érték egyenlő lesz a függvény értékével majd kirajzoltatjuk a plot segítségével ezt.

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

Tanulságok, tapasztalatok, magyarázat...

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

for (i in 1:kiserletek_szama)
{
  if(kiserlet[i]==jatekos[i])
  {
    mibol=setdiff(c(1,2,3), kiserlet[i]) }
  else{
    mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i])) }
  musorvezeto[i] = mibol[sample(1:length(mibol),1)] }

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

for (i in 1:kiserletek_szama) {
  holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
  valtoztat[i] = holvalt[sample(1:length(holvalt),1)] }

valtoztatesnyer = which(kiserlet==valtoztat)

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

Legelsőnek megadjuk a kísérletek számát ami 10millió lesz. A sample() függvény egy véletlenséget szimulál így következő lépésekben ezt használjuk. "1:3" megadja mettől meddig generáljon random számokat. "kiserletek_szama" azt határozza meg, hogyszor tegye meg és a végén replace=T pedig, hogy ismétlődhessen a szám. A musorvezeto vector pedig az előző két tömbtől fog függeni ezért csak a hosszúságát adjuk meg. A for ciklus 1-től a kísérletek számáig fog futni. Két eset fog fennállni egyik hogy eltállta az ajtót a máik pedig, hogy nem. Ha a "kiserlet" és a "jatekos" tömb megegyezik, akkor a "musorvezeto"

nem választhatja azt az ajtót. A setdiff függvény kiveszi a kísérlet tömb első elemével megegyező értéket. Hasonlóan az else-nél is csak ott a kísérlet és a játékos tömb első elemét kikell vonni. Ezt követően pedig már feltudjuk tölteni a musorvezeto vektort amihez a sample() függvényt használjuk.

DRAFT

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

```
#include <iostream>
void tounar(int a){
    for(int i=0; i<a; i++)
        std::cout << "1";

    std::cout << std::endl;
}
int main(){

    int val;
    std::cout << "Type a number in decimal." << std::endl;
    while(std::cin >> val){
        tounar(val);
    }

    return 0;
}
```

Az unáris nem más mint egy számrendszer. Kicsit eltérő mint a rendes mivel itt a legnagyobb szám az egyes. A többi számot úgy kapjuk meg, hogy az egyeseket összeadjuk tehát például a 3-mas számz az 111-ként jön létre. Legelsőnek megvizsgálja azt adott számot a számítógép. Ezt követően elmegy az utolsó szám utánig. Ezután visszalép egyel és elkezd kivonni minden 1et belőlük. Ez végig fog menni a számsorozatunkon. Ezeket az egyeseket amiket kivont elmenti és mi majd innen fogjuk tudni megnézni az eredményt.

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:

S, X, Y változók

a, b, c konstansok

S - abc, S - aXbc, Xb - bX, Xc - Ybcc, bY - Yb, aY - aaX, aY - aa

S (S - aXbc)

aXbc (Xb - bX)

abXc (Xc - Ybcc)

abYbcc (bY - Yb)

aYbbcc (aY - aaX)

aaXbbcc (Xb - bX)

aabXbcc (Xc - Ybcc)

aabbYbcc (bY - Yb)

aabYbbccc (bY - Yb)

aaYbbbccc (aY - aa)

aaabbbccc

A, B, C változók

a, b, c konstansok

A - aAB, A - aC, CB - bCc, cB - Bc, C - bc

A (A - aAB)

aAB (A - aAB)

aaABB (A - aC)

aaaaCBBB (CB - bCc)

aaaabCcBB (cB - Bc)

aaaabCBcB (cB - Bc)

aaaabCBBc (CB - bCc)

aaaabbCcBc (cB - Bc)

aaaabbCBcc (CB - bCc)

aaaabbbCccc (C - bc)

aaaabbbbcccc

Mint ahogy láthatjuk megadtunk két darab szabályt is ebből is látszik, hogy ez a nyelv nem környezetfüggetlen. Mind a két példánál megvannak adva, hogy mik a szimbólumok és a változók és maga a szabályok is.

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), mászával (például C99) igen.

Megoldás videó:

Megoldás forrása:

```
int main()
{
    for(int i = 0; i < 5; i++)
    {
        .....
    }

    return 0;
}
```

Az alábbi általunk megírt kód C89 szabványon nem fut le, mivel egy hibaüzenetet kapunk ami szerint a fejlécben nem megengedett a változó deklarációja. C99-es szabványon viszont lefut ez. Természetesen ha a változó deklarlását a kívülre tesszük akkor már lefog futni C89 szabvály szerint. Ennél a feladatnál még szükséges megemlíteni, hogy két kapcsolóra van szükségünk a lefordításnál, egyik a "-std:c89" másik pedig a "-std:c99".

3.4. Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetnél megjelenő valós számokat! Nem elfogadható olyan megoldás, amely maga olvassa betűnként a bemenetet, a feladat lényege, hogy lexert használunk, azaz óriások vallán állunk és ne kispályázzunk!

Megoldás videó:

Megoldás forrása:

```
% {
#include <stdio.h>
int realnum = 0;
%
digit [0-9]
%%
{digit}*(\.{digit}+) ? {++realnum_count;
printf("[realnum: %s - %f]", yytext, atof(yytext));
%%
```

Legelsőnek is írunk egy C kód csipetet amit látni szeretnénk a generált programba. Egy adott részletet a "%" jellel tudunk lezárnai lexerben. Ezután fognak következni a definicák amik jelen esetben számok definiálását fogja jelenteni. Ezután mint az előzőnél lezárjuk a "%" részletet. Ezt követően megadjuk a szabályt ami alapján felismerhetünk bizonyos részleteket. Jelen esetben csak egy szabályunk van ami a valós számokra fog érvényesülni.

```
int main() {
    yylex();
```

```
    printf("The number of real numbers: %d", realnum_count);
    return 0;
}
```

Itt már csak a főprogramunk maradt amiben a "yylex()" függvény segítségével megívjuk a lexerünket és kiíratjuk az eredményt. Ha ezzel megvagyunk nem maradt más dolgunk mint, hogy beletesszük a flex programba ezt "lex valos.l -o valos.c" parancsal. Ez a C forráskód jóval hosszabb mint az .l kód ezálltal és látható, hogy megspóroltunk vele némi időt.

3.5. I33t.l

Lexelj össze egy l33t cipher!

Megoldás videó:

Megoldás forrása:

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

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

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

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

```
{'u', {"u", "|_|", "(_)", "[_]"},  
'v', {"v", "\\\\", "\\\\\", "\\\\\"}},  
'w', {"w", "VV", "\\//\\//", "(/\\\")}},  
'x', {"x", "%", ")()", "()"},  
'Y', {"Y", "", "", ""}},  
'z', {"z", "2", "7_", ">_"}},  
  
'0', {"D", "0", "D", "0"}},  
'1', {"I", "I", "L", "L"}},  
'2', {"Z", "Z", "Z", "e"}},  
'3', {"E", "E", "E", "E"}},  
'4', {"h", "h", "A", "A"}},  
'5', {"S", "S", "S", "S"}},  
'6', {"b", "b", "G", "G"}},  
'7', {"T", "T", "j", "j"}},  
'8', {"X", "X", "X", "X"}},  
'9', {"g", "g", "j", "j"}}  
  
// https://simple.wikipedia.org/wiki/Leet  
};  
  
%}  
%%  
. {  
  
    int found = 0;  
    for(int i=0; i<L337SIZE; ++i)  
    {  
  
        if(l337d1c7[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int) (100.0*rand() / (RAND_MAX+1.0));  
  
            if(r<91)  
                printf("%s", l337d1c7[i].leet[0]);  
            else if(r<95)  
                printf("%s", l337d1c7[i].leet[1]);  
            else if(r<98)  
                printf("%s", l337d1c7[i].leet[2]);  
            else  
                printf("%s", l337d1c7[i].leet[3]);  
  
            found = 1;  
            break;  
        }  
  
    }  
  
    if(!found)
```

```
    printf("%c", *yytext);

}

%%

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

Ebben a feladatban a szleng nyelvvel fogunk foglalkozni. Különböző betűket tudunk használni akár másik számmal vagy másik szimbólikus karakterekkel is. Ilyen például az S betű helyett a \$ vagy E betű helyett a 3. Manapság ezek igen elterjedtek a fiatalok körébe, viszont sokan pedig nem értik ezeket a szleng nyelveket. Maga ez egész feladat érthető és könnyen végigvihető. A kódcsípet első részében megvannak adva az adott betűnek a szlengben használt "jelei". Ezután következik, hogy a nagybetűket kisbetűvé alakítja át a kód és átírjam az első részben vett leet karakterekre. A legvégén pedig egyszerűen csak hivatkozunk rá és meghívuk.

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);
```

Ha a SIGINT nem volt figyelmen kívül hagyva akkor a jelkezelő kezelje. Ha figyelmen kívül volt hagyva tövábbra is maradjon úgy.

ii.

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

A forciklus fejlécébe hiányzik az i deklaráció.

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

A forciklus fejlécébe hiányzik az i deklaráció.

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

A kód hibamentes ha már létrehoztuk a látható változókat és mutatókat.

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

A kód hibamentes ha már létrehoztuk a látható változókat és mutatókat.

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

Arra kell figyelni hogy ha az f függvény visszatérési értéke nem int akkor a kiírt értékek nem biztos hogy pontosak lesznek.

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

A printf ki fogja írni az f függvény visszatérési értékét a-ra, és a értékét.

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

A kiiratás megtörténik viszont az f függvény most az a változó memória címével fog dolgozni nem az a értékével.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

1. sor : minden x esetén létezik olyan y amelyik nagyobb az x-nél és prímszám.

2. sor : minden x esetén létezik olyan y amelyik nagyobb az x-nél és prímszám illetve y+2 is.

3. sor : létezik y minden x esetén mikor x az prím és kisebb az y-nál.

4. sor : létezik y minden x esetén mikor y kisebb mint az x és x az nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a;
```

- egészre mutató mutató

```
int *b;
```

- egész referencia

```
int &c;
```

- egészek tömbje

```
int T[3];
```

- egészek tömbjének referencia (nem az első elemé)

```
int (&T)[3] = T;
```

- egészre mutató mutatók tömbje

```
int *T[3];
```

- egészre mutató mutatót visszaadó függvény

```
int *func();
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int *(*func)();
```

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

```
int *(*func)();
```

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

```
int *(*func)();
```

Mit vezetnek be a programba a következő nevek?

- ```
int a;
```

Egész declaráció.

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

Pointer, ami az "a" változóra mutat.

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

Az 'a' változó referenciaja.

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

5 elemű tömb deklaráció.

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

A c tömb referenciaja.

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

Int-re mutató 5 elemű pointer tömb.

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

Int-re mutató függvény pointer.

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

Int-re mutató függvénypointer pointere.

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

Egészet visszaadó és két egészet kapó függvényre mutatót visszaadó, egészet kapó függvény

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

Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutatót visszaadó, egészet kapó függvényre

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

Megoldás videó: <https://youtu.be/1MRTuKwRsB0>, <https://youtu.be/RKbX5-EWpzA>.

Megoldás forrása: bhaxx/thematic_tutorials/bhaxx_textbook_IgyNeveldaProgramozod/Caesar/tm.c

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

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

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

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

    for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
```

```
tm[i][j] = i * (i + 1) / 2 + j;

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

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

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

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

free (tm);

return 0;
}
```

A feladat főcélja a dinamikus memóriakezelés. Legelején tisztázni kell mi is az az alsó háromszögmátrix. Ez egy olyan mátrix melyben a főátló felett csak 0 számjegy szerepelhet. Legelső lépés a kódcsípet értelmezésében, hogy deklarálnunk kell az "nr" változót amit jelen esetben 5-re tettünk. Ez egyben egyenlő lesz a mátrix sorainak a számával is. Maga a "malloc" függvény memóriát foglalt le ami visszaad egy pointert. Ez a pointer jelen esetünkben bármi lehet(mi válasszuk meg). A harmadik "for" fogja kiíratni az első kettő "forban" létrejött háromszögmátrixot. Maga a kódcsípetben az "i" a sorok számát míg, a "j" az oszlopok számát mutatja. Azért kell "i+1" mivel így kizártuk a nullával való osztást. Az utolsó for fogja felszabadítani a helyet a lefoglalt tömbnek.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

```
#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);
    }
}
```

Első két sorban felveszünk 2 darab kontstant változót. Egyik a "MAX_KULCS 100" lesz másik pedig a "BUFFER_MERET 256". Nevükhez hűen a MAX_KULCS a a kulcs maximális méretét fogja megadni míg a "BUFFER_MERET" a beolvasandó stringek számát maximum. Az intben lévő "strncpy" az egy függvény amivel a kulcsok méretét vizsgáljuk meg. Amennyiben túl nagy a kulcs mérete, ekkor 100 kitöröl belőle 100 karakternyi helyet. A "while" ciklus olvassa be a bajtakat. A kiolvasott bajtakat össze EXOR-ozza a kulcs adott bajtja segítségével. A kulcs indexet növeljük az EXOR után. A titkosított bajtakat egy bufferbe olvassuk.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás video:

Megoldás forrása:

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

Az előző feladat megoldása JAVA környezetben. Viszont van egy fontos eltérés a között ami nem máős minthogy a klcsot a standard inputról kéri be.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

```
#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 nullazása a titkos bufferben
for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';
// osszes kulcs eloallitasa
for (int ii = '0'; ii <= '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 szöveg: [%s]\n",
        ii, ji, ki, li, mi, ni, oi, pi, titkos);
        // ujra EXOR-ozunk, ily nem kell egy masodik buffer
    exor (kulcs, KULCS_MERET, titkos, p - titkos);
}
return 0;
}
```

Karakter sorozatból megadott kulcs alapján próbálja visszafejteni az EXOR törő a szöveget. Egy adott algoritmussal (Bruteforce) töri fel minden kombinációt felhasználva. Használnunk kell majd még egy algoritmust ami a tiszta szöveg előállításában segédkezik. Ez az algoritmust alapján egyes szavak alapján próbálja meg összerakni az értelmes szöveget.

4.5. Neurális OR, AND és EXOR kapu

Ebben a feladatban Dékány Róbert Zsoltot tutoráltam.

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

```
# Copyright (C) 2019 Dr. Norbert Bátfai, nbatfai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
# https://youtu.be/Koyw6IH5ScQ
```

```
library(neuralnet)

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

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

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

plot(nn.or)

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

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

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

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

plot(nn.orand)

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



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

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

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

plot(nn.exor)

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



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

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

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

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

plot(nn.exor)

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

Egy neurális háló elméletben könnyen működik, megadjuk, hogy egy adott bemenetre milyen kimenetet adjon vissza. Viszont ez a gyakorlatban egy kicsit nehezebb. Először nézzük a "neurális OR kaput". Ez a logikai vagy. Két db értéket kell megadnunk és a végeredményt majd kijön, hogy hány darab lépésből és mennyi idő alatt sikerült ezt neki elérnie. Másodszor a "neurális AND kaput" nézzük meg. Ez a logikai és. Itt is az előzőhöz mérten megadunk 2db értéket és az eredményt majd megpajuk hány lépésből jött ez ki neki és mennyi idő alatt. A harmadik a "kakuktojás" a három közül azaz az "EXOR", mivel itt is megadjuk a két darab értéket és az eredményt de itt kell hozzá adni egy titkos réteget is. Ez logikában amúg a kizárt vaggyal felel meg.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

```
#include <iostream>
#include "mlp.hpp"
#include "png++/png.hpp"
int main (int argc, char **argv)
{
    png::image<png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width()*png_image.get_height();

    Perceptron* p = new Perceptron(3, size, 256, 1);
    double* image = new double[size];

    for(int i {0}; i<png_image.get_width(); ++i)
        for(int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;
    double value = (*p) (image);
    std::cout << value << std::endl;
    delete p;
    delete [] image;
}
```

A feladat lényege, hogy három rétegű hálót csinálunk majd a 3. rétegben kapni fogunk egy eredményt (számot) amit a három rétegből különböző számításokból kaphatunk meg. Elsősorban behívjuk a képet. Második sorban helyet foglalunk le neki. Későbbiekben megkell adnunk pár paramétert ilyen a hány réteget használnuk, méretet, 1 számnak kell lenni a végeredménynek. Ezután megint helyet foglalunk le és feltöljük for ciklussal végül pedig meghívjuk a fgv. operátort. Ellenőrzés képpen pedig kiíratjuk az eredményt.



5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás videó:

Ebben a feladatban Dékány Róbert Zsoltot tutoriáltam.

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

#define MERET 600
#define ITER_HAT 32000

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

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

    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = 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;
    // Végigzongorázzuk a szélesség x magasság rátcsot:
```

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

            ++iteracio;
        }

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

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

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

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

```
{  
    std::cout << "Használat: ./mandelpng fajlnev";  
    return -1;  
}  
  
int kepadat [MERET] [MERET];  
  
mandel (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;  
}
```

Az egész csokorban hasznos lesz tudni a Mandelbrot "fogalmat", hogy mivel foglalkozik. Alapvetően komplex számokkal dolgozik egyenletkben. A megoldásban szereplő számokat "összekötve" egy képet fogunk kapni. Ebből kilogikázatjuk, hogy kapni fogunk egy kimenő fájlt amibe a megoldás lesz illetve egy képet is külön fájlban. Ezért szükséges a csokor elején telepíteni "png++"-t. Sudo apt segítségével tudjuk ezt megtenni. Miután ezt megtettük és utána néztünk több forrásból, hogy hogyan működik ez pontosan el is kezdhetjük a feladatokat megcsinálni. Létfogjuk hozni a képteret (min,max értékekkel). Maga a program további kódcsipetei annyiból állnak, hogy a kijt számok(halmaz) segítségével kirajzoltatunk egy képet, ezt a képet úgy kapjuk meg, hogy pixelről-pixelre rajzoltatja ki a gép.

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

Megoldás videó:

Megoldás forrása:

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

const int N = 500;
const int M = 500;
const double MAXX = 0.7;
const double MINX = -2.0;
const double MAXY = 1.35;
const double MINY = -1.35;

void GeneratePNG(const int tomb[N][M])
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y] ←
                ]);
        }
    }
    image.write("kimenet.png");
}

int main()
{
    int tomb[N][M];

    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;

    std::complex<double> C, Z, Zuj;

    int iteracio;

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            real(C) = MINX + j * dx;
            imag(C) = MAXY - i * dy;

            Z = 0;
            iteracio = 0;

            while (abs(Z) < 2 && iteracio++ < 255)
            {
                Zuj = Z * Z + C;
            }
            if (abs(Z) ≥ 2)
            {
                image[j][i] = png::rgb_pixel(255, 0, 0);
            }
            else
            {
                image[j][i] = png::rgb_pixel(0, 0, 255);
            }
        }
    }
}
```

```
    Z = Zuj;
}

tomb[i][j] = 256 - iteracio;
}

GeneratePNG(tomb);

return 0;
}
```

Mint ahogyan láthatjuk sok hasonlóság van ez előző feladathoz képest viszont van amiben eltér mint például, hogy itt már az " std::complex" osztálytal fogunk dolgozni. Az "std::complex" osztály már tud kezelní komplex számokat ezáltal ez megkönnyíti a munkánkat. Ez annyiba fogja megváltoztatni, hogy nem kell saját magunknak írni egy struktúrát. Ezenfelül szinte ugyan az az előző feladatban kitagláltakhoz.

5.3. Biomorfok

Megoldás videó: <https://youtu.be/IJMbgrzY76E>

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

```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 biomorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatás:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1 --left- ↵
// footer="BATF41 HAXOR STR34M" --right-footer="https://bhaxor. ↵
// blog.hu/" --pro=color
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/ ↵
// or modify
// it under the terms of the GNU General Public License as ↵
// published by
// the Free Software Foundation, either version 3 of the ↵
// License, or
// (at your option) any later version.
//
```

```
// This program is distributed in the hope that it will be ↵
// useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty ↵
// of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See ↵
// the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public ↵
// License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbqRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9\_Iss5\_2305--2315\_Biomorphs\_via\_modified\_iterations.pdf
//
#include <iostream>
#include "png++/png.hpp"
#include <complex>
int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;
    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag = atoi ( argv[3] );
        iteraciosHatar = atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );
    }
    else
    {
```

```
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelessseg ←
                  magassag n a b c d reC imC R" << std::endl;
    return -1;
}
png::image<png::rgb_pixel> kep ( szelessseg, magassag );
double dx = ( xmax - xmin ) / szelessseg;
double dy = ( ymax - ymin ) / magassag;
std::complex<double> cc ( reC, imC );
std::cout << "Szamitas\n";
// j megy a sorokon
for ( int y = 0; y < magassag; ++y )
{
    // k megy az oszlopokon
    for ( int x = 0; x < szelessseg; ++x )
    {
        double reZ = xmin + x * dx;
        double imZ = ymax - y * dy;
        std::complex<double> z_n ( reZ, imZ );
        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {
            z_n = std::pow(z_n, 3) + cc;
            //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
            if (std::real ( z_n ) > R || std::imag ( z_n ) > ←
                R)
            {
                iteracio = i;
                break;
            }
        }
        kep.set_pixel ( x, y,
                        png::rgb_pixel ( (iteracio*20)%255, ←
                                         (iteracio*40)%255, (iteracio ←
                                         *60)%255 ) );
    }
    int szazalek = ( double ) y / ( double ) magassag * ←
        100.0;
    std::cout << "\r" << szazalek << "%" << std::flush;
}
kep.write ( argv[1] );
std::cout << "\r" << argv[1] << " mentve." << std::endl;
}
```

Clifford Pickover találta fel a bioformokat a Julia halmazokat rajzoló programon dolgozott mikor csodálatos módon egy bug miatt rátalált a bioformokra. Az ilyenfajta szabályrendszer szerint valósulnak meg az egysejtűek. Ez a két halmaz között annyi a különbség, hogy míg a mandel brotban a "c" az egy konstants addig a másiknál egy állandó. Itt is a szélesség és a magasság a felhasználó által adott érték lesz mint az előző két feladatnál.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

```
// Forrás: https://github.com/SullyChen/Mandelbrot-Set-Plotter
#include "SFML/Graphics.hpp"
//resolution of the window
const int width = 1280;
const int height = 720;
//used for complex numbers
struct complex_number
{
    long double real;
    long double imaginary;
};
//mandelbrot komplex alapján legenerál egy mandelbrot halmazt
void generate_mandelbrot_set(sf::VertexArray& vertexarray, int ←
    pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            //scale the pixel location to the complex plane for ←
            //calculations
            long double x = ((long double)j - pixel_shift_x) / ←
                zoom;
            long double y = ((long double)i - pixel_shift_y) / ←
                zoom;
            complex_number c;
            c.real = x;
            c.imaginary = y;
            complex_number z = c;
            int iterations = 0; //keep track of the number of ←
            //iterations
            for (int k = 0; k < precision; k++)
            {
                complex_number z2;
                z2.real = z.real * z.real - z.imaginary * z. ←
                    imaginary;
                z2.imaginary = 2 * z.real * z.imaginary;
                z2.real += c.real;
                z2.imaginary += c.imaginary;
                z = z2;
                iterations++;
                if (z.real * z.real + z.imaginary * z.imaginary ←
                    > 4)
```

```
        break;
    }
    //color pixel based on the number of iterations
    if (iterations < precision / 4.0f)
    {
        vertexarray[i*width + j].position = sf::: ←
            Vector2f(j, i);
        sf:::Color color(iterations * 255.0f / ( ←
            precision / 4.0f), 0, 0);
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision / 2.0f)
    {
        vertexarray[i*width + j].position = sf::: ←
            Vector2f(j, i);
        sf:::Color color(0, iterations * 255.0f / ( ←
            precision / 2.0f), 0);
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision)
    {
        vertexarray[i*width + j].position = sf::: ←
            Vector2f(j, i);
        sf:::Color color(0, 0, iterations * 255.0f / ←
            precision);
        vertexarray[i*width + j].color = color;
    }
}
}
```

Maga a CUDA eléggyé elterjed manapság a gamerek és a videósok körében. Mivel nem elég, hogy egy játék elfut már arra is nagy hangsúlyt fektetnek, hogy milyen minőségen fut az el. A CUDA ebben segédkezik leginkább mivel egyszerre több magot futtat a videókártyán ezálltal nagyob teljesítményre képest a számítógép. A kód itt is hasonlít az előzőekhez, a legnagyobb eltérés viszont ott van amikor is itt a CUDA magjai számolják ki a megoldást még az előzőeknél a CPU számolta. Mint ahogy írtam ezálltal jóval gyorsabb a folyamat.

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:

```
// Forrás: https://github.com/SullyChen/Mandelbrot-Set-Plotter
#include "SFML/Graphics.hpp"
//resolution of the window
const int width = 1280;
const int height = 720;
//used for complex numbers
struct complex_number
{
    long double real;
    long double imaginary;
};

//mandelbrot komplex alapján legenerál egy mandelbrot halmazt
void generate_mandelbrot_set(sf::VertexArray& vertexarray, int ←
    pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            //scale the pixel location to the complex plane for ←
            //calculations
            long double x = ((long double)j - pixel_shift_x) / ←
                zoom;
            long double y = ((long double)i - pixel_shift_y) / ←
                zoom;
            complex_number c;
            c.real = x;
            c.imaginary = y;
            complex_number z = c;
            int iterations = 0; //keep track of the number of ←
            //iterations
            for (int k = 0; k < precision; k++)
            {
                complex_number z2;
                z2.real = z.real * z.real - z.imaginary * z. ←
                    imaginary;
                z2.imaginary = 2 * z.real * z.imaginary;
                z2.real += c.real;
                z2.imaginary += c.imaginary;
                z = z2;
                iterations++;
                if (z.real * z.real + z.imaginary * z.imaginary ←
                    > 4)
                    break;
            }
            //color pixel based on the number of iterations
            if (iterations < precision / 4.0f)
            {
                vertexarray[i*width + j].position = sf::: ←
```

```
        Vector2f(j, i);
        sf::Color color(iterations * 255.0f / ( ←
            precision / 4.0f), 0, 0);
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision / 2.0f)
    {
        vertexarray[i*width + j].position = sf::←
            Vector2f(j, i);
        sf::Color color(0, iterations * 255.0f / ( ←
            precision / 2.0f), 0);
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision)
    {
        vertexarray[i*width + j].position = sf::←
            Vector2f(j, i);
        sf::Color color(0, 0, iterations * 255.0f / ←
            precision);
        vertexarray[i*width + j].color = color;
    }
}
}
```

Mint szinte az összes többi feladatnál itt is van hasonlóság a mandelbrotos feladatokhoz. Egy függvény (void generate_mandelbrot_set(sf::VertexArray vertexarray..) fogja legenerálni MBH_t-t.

```
// Forrás: https://github.com/SullyChen/Mandelbrot-Set-Plotter
#include "SFML/Graphics.hpp"
//resolution of the window
const int width = 1280;
const int height = 720;
//used for complex numbers
struct complex_number
{
    long double real;
    long double imaginary;
};
//mandelbrot komplex alapján legenerál egy mandelbrot halmazt
void generate_mandelbrot_set(sf::VertexArray& vertexarray, int ←
    pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
```

```
//scale the pixel location to the complex plane for ←
//calculations
long double x = ((long double)j - pixel_shift_x) / ←
    zoom;
long double y = ((long double)i - pixel_shift_y) / ←
    zoom;
complex_number c;
c.real = x;
c.imaginary = y;
complex_number z = c;
int iterations = 0; //keep track of the number of ←
//iterations
for (int k = 0; k < precision; k++)
{
    complex_number z2;
    z2.real = z.real * z.real - z.imaginary * z. ←
        imaginary;
    z2.imaginary = 2 * z.real * z.imaginary;
    z2.real += c.real;
    z2.imaginary += c.imaginary;
    z = z2;
    iterations++;
    if (z.real * z.real + z.imaginary * z.imaginary ←
        > 4)
        break;
}
//color pixel based on the number of iterations
if (iterations < precision / 4.0f)
{
    vertexarray[i*width + j].position = sf::: ←
        Vector2f(j, i);
    sf:::Color color(iterations * 255.0f / ( ←
        precision / 4.0f), 0, 0);
    vertexarray[i*width + j].color = color;
}
else if (iterations < precision / 2.0f)
{
    vertexarray[i*width + j].position = sf::: ←
        Vector2f(j, i);
    sf:::Color color(0, iterations * 255.0f / ( ←
        precision / 2.0f), 0);
    vertexarray[i*width + j].color = color;
}
else if (iterations < precision)
{
    vertexarray[i*width + j].position = sf::: ←
        Vector2f(j, i);
    sf:::Color color(0, 0, iterations * 255.0f / ←
        precision);
    vertexarray[i*width + j].color = color;
```

```
        }
    }
}
```

Legelső lépés "sf::" ablknál kell beállítani tulajdonságokat (pl.: ablakméret). Ezután lekell generálnunk a MBH-t default paraméterek segítségével. Két állapotot fog figyelni : 1 : Szokásosan várja mikor zárjuk be az ablakot. 2 : Belépve az MBH-ba, 2x nagyítással fogja lefuttatni a fájlt.

```
int main()
{
    sf::String title_string = "Mandelbrot Set Plotter"; //ablak ←
    //címe
    sf::RenderWindow window(sf::VideoMode(width, height), ←
    //ablak objektum(létrehozza az ablakot a ←
    //megadott méretekkel és címmel)
    window.setFramerateLimit(30); //frissített ablak/s vagy ←
    //ilyesmi
    sf::VertexArray pointmap(sf::Points, width * height);

    //értékek inicializálása
    float zoom = 300.0f;
    int precision = 100;
    int x_shift = width / 2;
    int y_shift = height / 2;

    //legenerálja a mbh-t
    generate_mandelbrot_set(pointmap, x_shift, y_shift, ←
    precision, zoom);

    /**
     *
     *
     *
     */
    while (window.isOpen())
    {
        //ciklikusan figyeli az előforduló különböző event-eket ←
        //, ha egy olyan esemény következik be, hogy ←
        //rákattolunk az X gombra, akkor bezárja az ablakot
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }
    }
}
```

```
//ha a bal egérgommbal kattintunk, akkor az egér ←  
    helyére nagyít az alábbi algoritmus segítségével. ←  
    minden nagyítás után újra generálja a mbh-t.  
//zoom into area that is left clicked  
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))  
{  
    sf::Vector2i position = sf::Mouse::getPosition( ←  
        window);  
    x_shift -= position.x - x_shift;  
    y_shift -= position.y - y_shift;  
    zoom *= 2;  
    precision += 200;  
  
    #pragma omp parallel for  
    for (int i = 0; i < width*height; i++)  
    {  
        pointmap[i].color = sf::Color::Black;  
    }  
    generate_mandelbrot_set(pointmap, x_shift, y_shift, ←  
        precision, zoom);  
}  
window.clear();  
window.draw(pointmap);  
window.display();  
}  
  
return 0;  
}
```

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518>

Ez az előző feladat megírása JAVÁ-ban. Itt is meg kell adni a különböző paramétereket. A megoldásnak kapott számok segítségével pixeleket rajzoltatunk ki majd színezünk ki ezáltal kapjuk meg az elvárt képet. Nagyítás során nem történik különösebb változás, ugyan úgy kiszámolja a pixeleket majd kiszínezi őket és így kapjuk meg a képet.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

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!

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
class Random {

public:
    Random();
    ~Random() {}

    double get();

private:
    bool exist;
    double value;

};

Random::Random() {
    exist = false;
    std::srand (std::time(NULL));
}

double Random::get() {
    if (!exist) {
```

```
        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);

        value = r * v2;
        exist = !exist;

        return r * v1;
    }

    else{
        exist = !exist;
        return value;
    }
};

int main()
{

    Random rnd;

    for (int i = 0; i < 10; ++i) std::cout << rnd.get() << std::endl;
}
```

Az elején szükségünk lesz pár különböző függvénykönyvtárra amik tartalmazzák a kódban használt függvényeket. Miután ezt megettük létrehozzuk a "Random" osztály amiben megadjuk, hogy a kódunk melyik részei legyenek privátak illetve publikusok. Publikusba tesszük a konstruktort, destruktort illetve a "get" függvényt. Ezt követően megadjuk a konstuktort ami annyit takar, hogyha új objektumot hozunk létre akkor a a változót hamisra állítjuk. Ezt követően a "get" függvényt írjuk meg amit 2 felé tudunk, az egyik ha nincs legenerált szám tehát hamis a tehát az "exist" hamis, ilyenkor lekell generálnunk 2 számot amiből az egyiket megtartjuk másikat pedig elrakjuk. Másik eset mikor létezik lekért szám ilyenkor az "exist"-et hamisra állítjuk és visszaadjuk a lekért számot. Végül megírjuk a főprogramot amiben az eddig megírt függvényeket felhasználjuk.

JAVA <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsojava/PolarGen.java#l10>

Ez előző kód átvitele JAVÁban.

6.2. LZW

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

Megoldás videó:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
typedef struct binfa
{
    int ertek;
    struct binfa *bal nulla;
    struct binfa *jobb_egy;
} BINFA, *BINFA_PTR;
BINFA_PTR
uj_elem ()
{
    BINFA_PTR p;

    if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
        perror ("memoria");
        exit (EXIT_FAILURE);
    }
    return p;
}
extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);
int
main (int argc, char **argv)
{
    char b;

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

```
    if (fa->jobb_egy == NULL) {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
    }
    else{
        fa = fa->jobb_egy;
    }
}
printf ("\n");
kiir (gyoker);
extern int max_melyseg;
printf ("melyseg=%d", max_melyseg);
szabadit (gyoker);
}

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

Legelőször is muszáj definiálnunk a struktúránkat ami megadja az elem értékét illetve a bal és jobb oldali

mutatónak a mutatóját. Ezt követően létrehozunk egy "uj_elem" függvényt. Amennyiben például nincs elegendő hely vagy bármi féle hibába ütközik ezesetben kidob minket egy hibaüzenettel együtt. Megadjuk a "kiir" és a "szabadiit" függvényeket amelyek elé egy "extern" szót teszünk ami annyit tesz, hogy a programnak ezzel segítünk, hogy csak később definiáljuk teljesen ezeket a függvényeket. Csak ezek a lépések után kezdjük megírni a fő programunkat ami egy karakter típusú változóval és a binfa gyökérelemével kezdünk. Mint ahogy a kódban is láthatjuk az "uj_elem" függvény segítségével létrehozunk a binfához egy új elemet ami jelen esetben egy "/" jel, ezt betesszük a fába ami segítségével látni fogjunk, hogy hol járunk éppen. Ezután írunk egy "while" ciklust. Ezt követően az "if" függvényt használjuk ahol megnézzük, hogy a beolvastott karakter 0 akkor a fa álltal mutatott bal oldali csomópont elemét kell nézni ha nincs ott semmi(NULL) akkor létrehozza a program. Ha 1est kapunk ugyan az a helyzet mint az előzőnél csak jobb oldali ágon. Ezekután ki is íratjuk a fát. A főprogram után megadjuk a "melyseg" és a "max_melyseg" változókat.

6.3. Fabejárás

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

Megoldás video:

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

        kiir (elem->jobb_egy);
        kiir (elem->bal nulla);
        --melyseg;
    }
}

void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;

        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
```

```
    kiir (elem->jobb_egy);
    kiir (elem->bal nulla);

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

    --melyseg;
}
}
```

Egy fának 3 bejárási útvonala lehet. Inorder,preorder illetve postorder. Az előző kód inorder bejárást alkalmazott most pedig megmutatom preorder és postorder bejárásokban is. Először is kezdjük a preorderrel. Igazából nem sok eltérés van csak annyiban, hogy legelőször a gyökérelemmel kell kezdenünk a vizsgálatot, ezt követi az egyik oldal és a másik oldal zárja be a folyamatot. Következő pedig a postorderes bejárás aminél a lényeg, hogy először az egyik oldalt vizsgáljuk meg majd a másikat és a legvégén a gyökérelemet.

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!

```
// z3a7.cpp
//
// Együtt támadjuk meg: http://progpater.blog.hu/2011/04/14/ ←
// egyutt_tamadjuk_meg
// LZW fa építő 3. C++ átirata a C változatból (+mélység, atlag és szórás)
// Programozó Páternoszter
//
// Copyright (C) 2011, 2012, Bátfai Norbert, nbatfai@inf.unideb.hu, ←
// nbatfai@gmail.com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
```

```
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1,      http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2,      csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3,      http://progpater.blog.hu/2011/03/05/ ←
//              labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// 0.0.4,      z.cpp: a C verzióból svn: bevezetes/C/ziv/z.c átírjuk C++- ←
//              ra
//          http://progpater.blog.hu/2011/03/31/ ←
//              imadni_fogjatok_a_c_t_egy_emberkent_tiszta_szivbol
// 0.0.5,      z2.cpp: az fgv(*mut)-ok helyett fgv(&ref)
// 0.0.6,      z3.cpp: Csomopont beágyazva
//          http://progpater.blog.hu/2011/04/01/ ←
//              imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2
// 0.0.6.1     z3a2.c: LZWBInFa már nem barátja a Csomopont-nak, mert ←
//              annak tagjait nem használja direktben
// 0.0.6.2     Kis kommentezést teszünk bele 1. lépésként (hogy a kicsit ←
//              lemaradt hallgatóknak is
//          könnyebb legyen, jól megtűzdeljük további olvasmányokkal)
//          http://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg
//          (majd a 2. lépésben "beletesszük a d.c-t", majd s 3. ←
//          lépésben a parancssorsor argok feldolgozását)
// 0.0.6.3     z3a2.c: Fejlesztgetjük a forrást: http://progpater.blog.hu ←
//              /2011/04/17/a_tizedik_tizenegyedik_labor
// 0.0.6.4     SVN-beli, http://www.inf.unideb.hu/~nbatfai/p1/forrasok-SVN ←
//              /bevezetes/vedes/
// 0.0.6.5     2012.03.20, z3a4.cpp: N betűk (hiányok), sorvégek, vezető ←
//              komment figyelmen kívül: http://progpater.blog.hu/2012/03/20/ ←
//              a_vedes_elokeszitese
// 0.0.6.6     z3a5.cpp: mamenyaka kolléga észrevételére a több komment ←
//              sor figyelmen kívül hagyása
//          http://progpater.blog.hu/2012/03/20/a_vedes_elokeszitese/ ←
//              fullcommentlist/1#c16150365
// 0.0.6.7     Javaslom ezt a verziót választani védendő programnak
// 0.0.6.8     z3a7.cpp: pár kisebb javítás, illetve a védések támogatásához ←
//              további komment a <<
```

```
//      eltoló operátort tagfüggvényként, illetve globális függvényként ←
//      túlterhelő részekhez.
//      http://progpater.blog.hu/2012/04/10/ ←
//      imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_4/fullcommentlist/1# ←
//      c16341099
//



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

/* Az LZWBInFa osztályban absztraháljuk az LZW algoritmus bináris fa ←
   építését. Az osztály
definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ←
   ez lesz a
beágyazott Csomopont osztály. Miért ágyazzuk be? Mert külön nem szánunk ←
   neki szerepet, ezzel
is jelezük, hogy csak a fa részeként számolunk vele.*/


class LZWBInFa
{
public:
    /* Szemben a bináris keresőfánkkal (BinFa osztály)
       http://progpater.blog.hu/2011/04/12/ ←
       imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3
       itt (LZWBInFa osztály) a fa gyökere nem pointer, hanem a '/' betüt ←
       tartalmazó objektum,
       lásd majd a védett tagok között lent: Csomopont gyoker;
       A fa viszont már pointer, minden az épülő LZW-fánk azon csomópontjára ←
       mutat, amit az
       input feldolgozása során az LZW algoritmus logikája diktál:
       http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
       Ez a konstruktor annyit csinál, hogy a fa mutatót ráállítja a gyökérre ←
       . (Mert ugye
       laboron, blogon, előadásban tisztáztuk, hogy a tartalmazott tagok, ←
       most "CMopont gyoker"
       konstruktora előbb lefut, mint a tagot tartalmazó LZWBInFa osztály ←
       konstruktora, éppen a
       következő, azaz a fa=&gyoker OK.)
    */
    LZWBInFa ():fa (&gyoker)
    {
    }
    ~LZWBInFa ()
    {
        szabadít (gyoker.egyesGyermek ());
        szabadít (gyoker.nullasGyermek ());
    }
}
```

```
/* Tagfüggvényként túlterheljük a << operátort, ezzel a célunk, hogy ←
   felkeltsük a
   hallgató érdeklődését, mert ekkor így nyomhatjuk a fába az inputot: ←
       binFa << b; ahol a b
   egy '0' vagy '1'-es betű.
   Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett ←
   paraméterként"
   kapott ) példány, azaz annak a fának amibe éppen be akarjuk nyomni a b ←
   betűt a tagjai
   (pl.: "fa", "gyoker") használhatóak a függvényben.
```

A függvénybe programoztuk az LZW fa építésének algoritmusát tk.:
http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

a b formális param az a betű, amit éppen be kell nyomni a fába.

```
a binFa << b (ahol a b majd a végén látszik, hogy már az '1' vagy a ←
   '0') azt jelenti
tagfüggvényként, hogy binFa.operator<<(b) (globálisként így festene: ←
   operator<<(binFa, b) )

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

```
{  
    Csomopont *uj = new Csomopont ('1');  
    fa->ujEgyesGyermek (uj);  
    fa = &gyoker;  
}  
else  
{  
    fa = fa->egyesGyermek ();  
}  
}  
/* A bejárással kapcsolatos függvényeink (túlterhelt kiir-ók, atlag, ←  
ratlag stb.) rekurzívak,  
tk. a rekurzív fabejárást valósítják meg (lásd a 3. előadás "Fabejárás ←  
" c. fóliáját és társait)  
  
(Ha a rekurzív függvénnyel általában gondod van => K&R könyv megfelel ←  
ő része: a 3. ea. izometrikus  
részében ezt "letáncoltuk" :) és külön idéztük a K&R álláspontját :)  
*/  
void kiir (void)  
{  
    // Sokkal elegánsabb lenne (és más, a bevezetésben nem kibontandó ←  
    // reentráns kérdések miatt is, mert  
    // ugye ha most két helyről hívják meg az objektum ilyen függvényeit, ←  
    // tehát ha kétszer kezd futni az  
    // objektum kiir() fgv.-e pl., az komoly hiba, mert elromlana a mélység ←  
    // ... tehát a mostani megoldásunk  
    // nem reentráns) ha nem használnánk a C verzióban globális változókat, ←  
    // a C++ változatban példánytagot a  
    // mélység kezelésére: http://progpter.blog.hu/2011/03/05/ ←  
    // there_is_no_spoon  
    melyseg = 0;  
    // ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát, ←  
    // akkor a  
    // sztenderd out-ra nyomjuk  
    kiir (&gyoker, std::cout);  
}  
/* már nem használjuk, tartalmát a dtor hívja  
void szabadit (void)  
{  
    szabadit (gyoker.egyesGyermek ());  
    szabadit (gyoker.nullasGyermek ());  
    // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a ←  
    // szabad tárban (halmon).  
}  
*/  
  
/* A változatosság kedvéért ezeket az osztálydefiníció (class LZWBinFa ←  
{...};) után definiáljuk,
```

```
hogy kénytelen légy az LZWBInFa és a :: hatókör operátorral minősítve ←
definiálni :) l. lentebb */
int getMelyseg (void);
double getAtlag (void);
double getSzoras (void);

/* Vágunk, hogy a felépített LZW fát ki tudjuk nyomni ilyenformán: std:: ←
cout << binFa;
de mivel a << operátor is a sztenderd névtérben van, de a using ←
namespace std-t elvből
nem használjuk bevezető kurzusban, így ez a konstrukció csak az ←
argfüggő névfeloldás miatt
fordul le (B&L könyv 185. o. teteje) ám itt nem az a lényeg, hanem, ←
hogy a cout ostream
osztálybeli, így abban az osztályban kéne módosítani, hogy tudjon ←
kiírni LZWBInFa osztálybelieket...
e helyett a globális << operátort terheljük túl,
a kiFile << binFa azt jelenti, hogy

- tagfüggvényként: kiFile.operator<<(binFa) de ehhez a kiFile ←
valamilyen
std::ostream stream osztály forrásába kellene beleírni ezt a ←
tagfüggvényt,
amely ismeri a mi LZW binfánkat...

- globális függvényként: operator<<(kiFile, binFa) és pont ez látszik ←
a következő sorban:

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

private:
    class Csomopont
{
public:
    /* A paraméter nélküli konstruktor az elepértelmezett '/' "gyökér-bet ←
       űvel" hozza
       létre a csomópontot, illet hívunk a fából, aki tagként tartalmazza a ←
       gyökeret.
    Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu" ←
```

```
        tagba, a két
        gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is ←
        megteszi. */
Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
{
};
~Csonopont ()
{
};
// Aktuális csomópont, mond meg nékem, ki a bal oldali gyermeked
// (a C verzió logikájával műxik ez is: ha nincs, akkor a null megy ←
// vissza)
Csonopont *nullasGyermek () const
{
    return balNulla;
}
// Aktuális csomópon, mond meg nékem, ki a jobb oldali gyermeked?
Csonopont *egyesGyermek () const
{
    return jobbEgy;
}
// Aktuális csomópont, ímholt legyen a "gy" mutatta csomópont a bal ←
// oldali gyereked!
void ujNullasGyermek (Csonopont * gy)
{
    balNulla = gy;
}
// Aktuális csomópont, ímholt legyen a "gy" mutatta csomópont a jobb ←
// oldali gyereked!
void ujEgyesGyermek (Csonopont * gy)
{
    jobbEgy = gy;
}
// Aktuális csomópont: Te milyen betűt hordozol?
// (a const kulcsszóval jelezzük, hogy nem bántjuk a példányt)
char getBetu () const
{
    return betu;
}

private:
    // friend class LZWBInFa; /* mert ebben a valtozatban az LZWBInFa ←
    // metódusai nem közvetlenül
    // a Csonopont tagjaival dolgoznak, hanem beállító/lekérdező ←
    // üzenetekkel érik el azokat */

    // Milyen betűt hordoz a csomópont
    char betu;
    // Melyik másik csomópont a bal oldali gyermeke? (a C változatból " ←
    // örököl" logika):
```

```
// ha hincs ilyen csermek, akkor balNulla == null) igaz
Csomopont *balNulla;
Csomopont *jobbEgy;
// nem másolható a csomópont (ökörszabály: ha van valamelye a szabad ←
// tárban,
// letiltjuk a másoló konstruktort, meg a másoló értékkadást)
Csomopont (const Csomopont &);
Csomopont & operator= (const Csomopont &);
};

/* Mindig a fa "LZW algoritmus logikája szerinti aktuális" csomópontjára ←
mutat */
Csomopont *fa;
// technikai
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
// szokásosan: nocopyable
LZWBInFa (const LZWBInFa &);
LZWBInFa & operator= (const LZWBInFa &);

/* Kiírja a csomópontot az os csatornára. A rekurzió kapcsán lásd a ←
korábbi K&R-es utalást... */
void kiir (Csomopont * elem, std::ostream & os)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    // leállítása
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->>nullasGyermek (), os);
        --melyseg;
    }
}
void szabadit (Csmopont * elem)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    // leállítása
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek ());
        szabadit (elem->>nullasGyermek ());
        // ha a csomópont minden gyermekét felszabadítottuk
        // azután szabadítjuk magát a csomópontot:
        delete elem;
    }
}
```

```
        }

protected:      // ha esetleg egyszer majd kiterjesztjük az osztályt, mert
// akarunk benne valami újdonságot csinálni, vagy meglévő tevékenységet ←
// más hogy... stb.
// akkor ezek látszanak majd a gyerek osztályban is

/* A fában tagként benne van egy csomópont, ez erősen ki van tüntetve, ō ←
   a gyökér: */
Csomopont gyoker;
int maxMelyseg;
double atlag, szoras;

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

};

// Néhány függvényt az osztálydefiníció után definiálunk, hogy lássunk ←
// illet is ... :)
// Nem erőltetjük viszont a külön fájlba szedést, mert a ←
// sablonosztályosított tovább
// fejlesztésben az linkelési gondot okozna, de ez a téma már kivezet a ←
// laborteljesítés
// szükséges feladatából: http://progpater.blog.hu/2011/04/12/ ←
// imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3

// Egyébként a melyseg, atlag és szoras fgv.-ek a kiir fgv.-el teljesen egy ←
// kaptafa.

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

void
```

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

// teszt pl.: http://progpater.blog.hu/2011/03/05/ ←
// labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// [norbi@sgu ~]$ echo "01111001001001000111"|./z3a2
// -----1(3)
// -----1(2)
// -----1(1)
// -----0(2)
// -----0(3)
// -----0(4)
// ---/(0)
// -----1(2)
// -----0(1)
// -----0(2)
// depth = 4
// mean = 2.75
// var = 0.957427
// a laborvédéshez majd ezt a tesztelést használjuk:
// http://

/* Ez volt eddig a main, de most komplexebb kell, mert explicite bejövő, ←
   kimenő fájlokkal kell dolgozni
int
main ()
{
    char b;
    LZWBinFa binFa;

    while (std::cin >> b)
    {
        binFa << b;
    }

    //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ←
    // verziókban de, hogy izgalmasabb legyen
```

```
// a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

std::cout << binFa; // ehhez kell a globális operator<< túlterhelése, ↵
lásd fentebb

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

binFa.szabadít ();

return 0;
}

*/
/* A parancssor arg. kezelést egyszerűen bedolgozzuk a 2. hullám kapcsolódó ↵
feladatából:
http://progpter.blog.hu/2011/03/12/hey\_mikey\_he\_likes\_it\_ready\_for\_more\_3
de mivel nekünk sokkal egyszerűbb is elég, alig hagyunk meg belőle valamit ↵
...
*/
void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    // http://progpter.blog.hu/2011/03/12/ ↵
    // hey_mikey_he_likes_it_ready_for_more_3
    // alapján a parancssor argok ottani elegáns feldolgozásából kb. ennyi ↵
    // marad:
    // "*((++argv)+1)"...

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

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

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

// ha igen, akkor az 5. előadásból kimásoljuk a fájlkezelés C++ ←
// változatát:
std::fstream beFile (inFile, std::ios_base::in);

// fejlesztgetjük a forrást: http://propater.blog.hu/2011/04/17/ ←
// a_tizedik_tizenegyedik_labor
if (!beFile)
{
    std::cout << inFile << " nem létezik..." << std::endl;
    usage ();
    return -3;
}

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

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

// a bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen ←
// írjuk, hogy meg tudjuk
// majd nézni... :) l. az említett 5. ea. C -> C++ gyökkettes átírási ←
// példáit

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

bool kommentben = false;

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

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

```
}

    if (kommentben)
continue;

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

    // egyszerűen a korábbi d.c kódját bemásoljuk
    // laboron többször lerajzoltuk ezt a bit-tolatást:
    // a b-ben lévő bajt bitjeit egyenként megnézzük
    for (int i = 0; i < 8; ++i)
{
    // maszkolunk eddig..., most már simán írjuk az if fejébe a legmagasabb ←
        helyisértékű bit vizsgálatát
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ←
        megmondja melyik:
    if (b & 0x80)
        // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW fa ←
            objektumunkba
        binFa << '1';
    else
        // különben meg a '0' betűt:
        binFa << '0';
    b <= 1;
}

//std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ←
    verziókban de, hogy izgalmasabb legyen
// a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

kiFile << binFa;      // ehhez kell a globális operator<< túlterhelése, ←
    lásd fentebb
// (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, ←
    mégis megy, hurrá)

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

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

return 0;
}
```

Bevezetés a programozásba című tantárgyon már taglaltuk ezt a programot és csipekadtuk benne. Itt már

alapból a gyökér része a binfának.

6.5. Mutató a gyökér

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

Megoldás videó:

A teljes forráskód:

```
// z3a7.cpp
//
// Együtt támadjuk meg: http://progpater.blog.hu/2011/04/14/ ←
// egyutt_tamadjuk_meg
// LZW fa építő 3. C++ átirata a C változatból (+mélység, atlag és szórás)
// Programozó Páternoszter
//
// Copyright (C) 2011, 2012, Bátfai Norbert, nbatfai@inf.unideb.hu, ←
// nbatfai@gmail.com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthető illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszőleges) későbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
```

```
// 0.0.1,          http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2,          csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3,          http://progpater.blog.hu/2011/03/05/ ←
//                  labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// 0.0.4,          z.cpp: a C verzióból svn: bevezetes/C/ziv/z.c átírjuk C++- ←
//                  ra
//                  http://progpater.blog.hu/2011/03/31/ ←
//                  imadni_fogjatok_a_c_t_egy_emberkent_tiszta_szivbol
// 0.0.5,          z2.cpp: az fgv(*mut)-ok helyett fgv(&ref)
// 0.0.6,          z3.cpp: Csomopont beágyazva
//                  http://progpater.blog.hu/2011/04/01/ ←
//                  imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2
// 0.0.6.1         z3a2.c: LZWBInFa már nem barátja a Csomopont-nak, mert ←
//                  annak tagjait nem használja direktben
// 0.0.6.2         Kis kommentezést teszünk bele 1. lépésként (hogy a kicsit ←
//                  lemaradt hallgatóknak is
//                  könnyebb legyen, jól megtűzdeljük további olvasmányokkal)
//                  http://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg
//                  (majd a 2. lépésben "beletesszük a d.c-t", majd s 3. ←
//                  lépésben a parancssorsor argok feldolgozását)
// 0.0.6.3         z3a2.c: Fejlesztgetjük a forrást: http://progpater.blog.hu ←
//                  /2011/04/17/a_tizedik_tizenegyedik_labor
// 0.0.6.4         SVN-beli, http://www.inf.unideb.hu/~nbatfai/p1/forrasok-SVN ←
//                  /bevezetes/vedes/
// 0.0.6.5         2012.03.20, z3a4.cpp: N betűk (hiányok), sorvégek, vezető ←
//                  komment figyelmen kívül: http://progpater.blog.hu/2012/03/20/ ←
//                  a_vedes_elokeszitese
// 0.0.6.6         z3a5.cpp: mamenyaka kolléga észrevételére a több komment ←
//                  sor figyelmen kívül hagyása
//                  http://progpater.blog.hu/2012/03/20/a_vedes_elokeszitese/ ←
//                  fullcommentlist/1#c16150365
// 0.0.6.7         Javaslom ezt a verziót választani védendő programnak
// 0.0.6.8         z3a7.cpp: pár kisebb javítás, illetve a védések támogatásához ←
//                  további komment a <<
//                  eltoló operátort tagfüggvényként, illetve globális függvényként ←
//                  túlterhelő részekhez.
//                  http://progpater.blog.hu/2012/04/10/ ←
//                  imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_4/fullcommentlist/1# ←
//                  c16341099
//                  http://progpater.blog.hu/2012/04/10/a_vedes_elokeszitese/ ←
//                  fullcommentlist/1#c16341099

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

/* Az LZWBInFa osztályban absztraháljuk az LZW algoritmus bináris fa ←
// építését. Az osztály
definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ←
// ez lesz a
```

beágyazott Csomopont osztály. Miért ágyazzuk be? Mert külön nem szánunk neki szerepet, ezzel is jelezzük, hogy csak a fa részeként számolunk vele.*/

```
class LZWBinFa
{
public:
    /* Szemben a bináris keresőfánkkal (BinFa osztály)
     * http://progpater.blog.hu/2011/04/12/
     * imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3
     * itt (LZWBinFa osztály) a fa gyökere nem pointer, hanem a '/' betüt
     * tartalmazó objektum,
     * lásd majd a védett tagok között lent: Csomopont gyoker;
     * A fa viszont már pointer, minden az épülő LZW-fánk azon csomópontjára
     * mutat, amit az
     * input feldolgozása során az LZW algoritmus logikája diktál:
     * http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
     * Ez a konstruktor annyit csinál, hogy a fa mutatót ráállítja a gyökérre
     * . (Mert ugye
     * laboron, blogon, előadásban tisztáztuk, hogy a tartalmazott tagok,
     * most "Csomopont gyoker"
     * konstruktora előbb lefut, mint a tagot tartalmazó LZWBinFa osztály
     * konstruktora, éppen a
     * következő, azaz a fa=&gyoker OK.)
*/
LZWBinFa ()
{
    gyoker = new Csomopont ('/');
    fa = gyoker;
}
~LZWBinFa ()
{
    szabadít (gyoker->egyesGyermekek ());
    szabadít (gyoker->>nullasGyermekek ());
    delete (gyoker);
}

/* Tagfüggvényként túlterheljük a << operátort, ezzel a célunk, hogy
   felkeltsük a
   hallgató érdeklődését, mert ekkor így nyomhatjuk a fába az inputot:
   binFa << b; ahol a b
   egy '0' vagy '1'-es betű.
   Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett"
   paraméterként"
   kapott ) példány, azaz annak a fának amibe éppen be akarjuk nyomni a b
   betűt a tagjai
   (pl.: "fa", "gyoker") használhatóak a függvényben.
```

A függvénybe programoztuk az LZW fa építésének algoritmusát tk.:
http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

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

```
}

/* A bejárással kapcsolatos függvényeink (túlterhelt kiir-ók, atlag, ←
   ratlag stb.) rekurzívak,
tk. a rekurzív fabejárást valósítják meg (lásd a 3. előadás "Fabejárás ←
   " c. fóliáját és társait)

(Ha a rekurzív függvényel általában gondod van => K&R könyv megfelel ←
   ō része: a 3. ea. izometrikus
részében ezt "letáncoltuk" :) és külön idéztük a K&R álláspontját :)

*/
void kiir (void)
{
    // Sokkal elegánsabb lenne (és más, a bevezetésben nem kibontandó ←
    // reentráns kérdések miatt is, mert
    // ugye ha most két helyről hívják meg az objektum ilyen ←
    // függvényeit, tahát ha kétszer kezd futni az
    // objektum kiir() fgv.-e pl., az komoly hiba, mert elromlana a ←
    // mélység... tehát a mostani megoldásunk
    // nem reentráns) ha nem használnánk a C verzióban globális ←
    // változókat, a C++ változatban példánytagot a
    // mélység kezelésére: http://progpater.blog.hu/2011/03/05/ ←
    // there_is_no_spoon
    melyseg = 0;
    // ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát, ←
    // akkor a
    // sztenderd out-ra nyomjuk
    kiir (gyoker, std::cout);
}
/* már nem használjuk, tartalmát a dtor hívja
void szabadit (void)
{
    szabadit (gyoker.egyesGyermek ());
    szabadit (gyoker.nullasGyermek ());
    // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a ←
    // szabad tárban (halmon).
}
*/

/* A változatosság kedvéért ezeket az osztálydefiníció (class LZWBinFa ←
   {...}); után definiáljuk,
hogy kényetlen légy az LZWBinFa és a :: hatókör operátorral minősítve ←
   definiálni :) l. lentebb */
int getMelyseg (void);
double getAtlag (void);
double getSzoras (void);

/* Vágunk, hogy a felépített LZW fát ki tudjuk nyomni ilyenformán: std ←
   ::cout << binFa;
de mivel a << operátor is a sztenderd névtérben van, de a using ←
   namespace std-t elvből
```

nem használjuk bevezető kurzusban, így ez a konstrukció csak az ↪
argfüggő névfeloldás miatt
fordul le (B&L könyv 185. o. teteje) ám itt nem az a lényeg, hanem, ↪
hogy a cout ostream
osztálybeli, így abban az osztályban kéne módosítani, hogy tudjon ↪
kiírni LZWBinFa osztálybelieket...
e helyett a globális << operátort terheljük túl,
a kiFile << binFa azt jelenti, hogy

- tagfüggvényként: kiFile.operator<<(binFa) de ehhez a kiFile ↪
valamilyen
std::ostream stream osztály forrásába kellene beleírni ezt a ↪
tagfüggvényt,
amely ismeri a mi LZW binfánkat...
- globális függvényként: operator<<(kiFile, binFa) és pont ez látszik ↪
a következő sorban:

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

private:
class Csomopont
{
public:
    /* A paraméter nélküli konstruktor az elepértelmezett '/' "gyökér- ↪
       betűvel" hozza
       létre a csomópontot, illet hívunk a fából, aki tagként tartalmazza a ↪
       gyökeret.
    Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu" ↪
       tagba, a két
       gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is ↪
       megteszi. */
    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
    {
    };
    ~Csomopont ()
    {
    };
    // Aktuális csomópont, mondd meg nékem, ki a bal oldali gyermeked
```

```
// (a C verzió logikájával műxik ez is: ha nincs, akkor a null megy ←
// vissza)
Csomopont *nullasGyernek () const
{
    return balNulla;
}
// Aktuális csomópon, t mond meg nékem, ki a jobb oldali gyermeked?
Csomopont *egyesGyernek () const
{
    return jobbEgy;
}
// Aktuális csomópont, ímholt legyen a "gy" mutatta csomópont a bal ←
// oldali gyerekek!
void ujNullasGyernek (Csmopont * gy)
{
    balNulla = gy;
}
// Aktuális csomópont, ímholt legyen a "gy" mutatta csomópont a jobb ←
// oldali gyerekek!
void ujEgyesGyernek (Csmopont * gy)
{
    jobbEgy = gy;
}
// Aktuális csomópont: Te milyen betűt hordozol?
// (a const kulcsszóval jelezzük, hogy nem bántjuk a példányt)
char getBetu () const
{
    return betu;
}

private:
// friend class LZWBinFa; /* mert ebben a valtozatban az LZWBinFa ←
// metódusai nem közvetlenül
// a Csomopont tagjaival dolgoznak, hanem beállító/lekérdező ←
// üzenetekkel érik el azokat */

// Milyen betűt hordoz a csomópont
char betu;
// Melyik másik csomópont a bal oldali gyermek? (a C változatból "←
// örökölt" logika:
// ha hincs ilyen csermek, akkor balNulla == null) igaz
Csmopont *balNulla;
Csmopont *jobbEgy;
// nem másolható a csomópont (ökörszabály: ha van valamelye a ←
// szabad tárban,
// letiltjuk a másoló konstruktort, meg a másoló értékkadást)
Csmopont (const Csmopont &); //másoló konstruktor
Csmopont & operator= (const Csmopont &);
};
```

```
/* Mindig a fa "LZW algoritmus logikája szerinti aktuális" ←
   csomópontjára mutat */
Csomopont *fa;
// technikai
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
// szokásosan: nocopyable
LZWBInFa (const LZWBInFa &);
LZWBInFa & operator= (const LZWBInFa &);

/* Kiírja a csomópontot az os csatornára. A rekurzió kapcsán lásd a ←
   korábbi K&R-es utalást... */
void kiir (Csmopont * elem, std::ostream & os)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    leállítása
    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
        kiir (elem->>nullasGyermek (), os);
        --melyseg;
    }
}
void szabadit (Csmopont * elem)
{
    // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ←
    leállítása
    if (elem != NULL)
    {
        szabadit (elem->egyesGyermek ());
        szabadit (elem->>nullasGyermek ());
        // ha a csomópont minden gyermekét felszabadítottuk
        // azután szabadítjuk magát a csomópontot:
        delete elem;
    }
}

protected:      // ha esetleg egyszer majd kiterjesztjük az osztályt, mert
    // akarunk benne valami újdonságot csinálni, vagy meglévő tevékenységet ←
    // más hogy... stb.
    // akkor ezek látszanak majd a gyerek osztályban is

/* A fában tagként benne van egy csomópont, ez erősen ki van tüntetve, ←
```

```
Ó a gyökér: */
Csomopont *gyoker;
int maxMelyseg;
double atlag, szoras;

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

};

// Néhány függvényt az osztálydefiníció után definiálunk, hogy lássunk ←
// illet is ... :)
// Nem erőltetjük viszont a külön fájlba szedést, mert a ←
// sablonosztályosított tovább
// fejlesztésben az linkelési gondot okozna, de ez a téma már kivezet a ←
// laborteljesítés
// szükséges feladatából: http://progpater.blog.hu/2011/04/12/ ←
// imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3

// Egyébként a melyseg, atlag és szoras fgv.-ek a kiir fgv.-el teljesen egy ←
// kaptafa.

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 (Csmopont * 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 (Csmopont * elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyesGyermek ());
        rszoras (elem->nullasGyermek ());
        --melyseg;
```

```
if (elem->egyesGyermek () == NULL && elem->>nullasGyermek () == NULL -->
)
{
    ++atlagdb;
    szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
}
}

// teszt pl.: http://progpater.blog.hu/2011/03/05/ ←
// labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// [norbi@sgu ~]$ echo "01111001001001000111" | ./z3a2
// -----1(3)
// -----1(2)
// -----1(1)
// -----0(2)
// -----0(3)
// -----0(4)
// ---/(0)
// -----1(2)
// -----0(1)
// -----0(2)
// depth = 4
// mean = 2.75
// var = 0.957427
// a laborvédéshez majd ezt a tesztelést használjuk:
// http://

/* Ez volt eddig a main, de most komplexebb kell, mert explicite bejövő, ←
   kimenő fájlokkal kell dolgozni
int
main ()
{
    char b;
    LZWBinFa binFa;

    while (std::cin >> b)
    {
        binFa << b;
    }

    //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ←
    // verziókban de, hogy izgalmasabb legyen
    // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

    std::cout << binFa; // ehhez kell a globális operator<< túlterhelése, ←
    // lásd fentebb

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

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

binFa.szabadit ();

return 0;
}

*/
/* A parancssor arg. kezelést egyszerűen bedolgozzuk a 2. hullám kapcsolódó ←
 feladatából:
http://progpater.blog.hu/2011/03/12/hey_mikey_he_likes_it_ready_for_more_3
de mivel nekünk sokkal egyszerűbb is elég, alig hagyunk meg belőle valamit ←
 ...
*/
void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    // http://progpater.blog.hu/2011/03/12/ ←
    // hey_mikey_he_likes_it_ready_for_more_3
    // alapján a parancssor argok ottani elegáns feldolgozásából kb. ennyi ←
    // marad:
    // "*((++argv)+1)"...

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

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

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

```
// ha igen, akkor az 5. előadásból kimásoljuk a fájlkezelés C++ ←
// változatát:
std::fstream beFile (inFile, std::ios_base::in);

// fejlesztgetjük a forrást: http://progpater.blog.hu/2011/04/17/ ←
// a_tizedik_tizenegyedik_labor
if (!beFile)
{
    std::cout << inFile << " nem létezik..." << std::endl;
    usage ();
    return -3;
}

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

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

// a bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen ←
// írjuk, hogy meg tudjuk
// majd nézni... :) l. az említett 5. ea. C -> C++ gyökkettes átírási ←
// példáit

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

bool kommentben = false;

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

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

    if (kommentben)
        continue;

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

```
// egyszerűen a korábbi d.c kódját bemásoljuk
// laboron többször lerajzoltuk ezt a bit-tolotatást:
// a b-ben lévő bájt bitjeit egyenként megnézzük
for (int i = 0; i < 8; ++i)
{
    // maszkolunk eddig..., most már simán írjuk az if fejébe a ←
    // legmagasabb helyiértékű bit vizsgálatát
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ←
    // megmondja melyik:
    if (b & 0x80)
        // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW ←
        // fa objektumunkba
        binFa << '1';
    else
        // különben meg a '0' betűt:
        binFa << '0';
    b <<= 1;
}

//std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ←
// verziókban de, hogy izgalmasabb legyen
// a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

kiFile << binFa;      // ehhez kell a globális operator<< túlterhelése, ←
                      // lásd fentebb
// (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, ←
// mégis megy, hurrá)

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

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

return 0;
}
```

Maga az "aggreráció" szó jelentése annyit tesz, hogy különböző elemeket kapcsolnak össze. Egy könnyebb példa erre az összeadás művelet. Megkell változtatnunk azokat a kód csípeteket ahol a gyökér változóira hivatkozunk. A gyökér típusát könnyen megtudjuk változtatni csomópontra mutató pointerré ami ehez a feladathoz fog kellenni (Csomopont *gyoker;). A gyökér elől törölni kell a különböző operátorokat.

6.6. Mozgató szemantika

Írj az előző programhoz mozgató konstruktort és értékkadást, a mozgató konstruktor legyen a mozgató értékkadásra alapozva!

Megoldás forrása:

Ez egy egyszerű feladat azoknak akik átlátják feladatot.

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

    gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
    gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

    regi.gyoker.ujEgyesGyermek ( nullptr );
    regi.gyoker.ujNullasGyermek ( nullptr );

}

LZWBInFa& operator = (LZWBInFa && regi)
{
    if (this == &regi)
        return *this;

    gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
    gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

    regi.gyoker.ujEgyesGyermek ( nullptr );
    regi.gyoker.ujNullasGyermek ( nullptr );

    return *this;
}
```

Legelőször a feladatban szereplő két szempontot kell végezni vinnünk azaz a mozgató konstruktort illetve az értékkadást. Egyenlővé kell tennük a régi fa illetve az új fa tartalmát, ezt úgy tehetjük meg, ha a régi fát átmásoljuk az új fába. Maga a kódcsípetben szereplő konstruktor illetve érték adásnak a mintája szinte megegyező.

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

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

Miután ezek megtörténtek a "move" függvényel átvisszük az új fába a régi fát. Ezek után nincs más dolgunk mint már csak kiíratni az új fa tartalmát amit a szokott módon tesszük meg a szükséges paraméterekkel.

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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

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

1. passzolási lehetősségem (labor)

7.2. Java életjáték

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

```
/*
 * Sejtautomata.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * Sejtautomata osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class Sejtautomata extends java.awt.Frame implements Runnable {
    /** Egy sejt lehet élő */
    public static final boolean ÉLŐ = true;
    /** vagy halott */
    public static final boolean HALOTT = false;
    /** Két rácsot használunk majd, az egyik a sejttér állapotát
     * a t_n, a másik a t_n+1 időpillanatban jellemzi. */
    protected boolean[][][] rácsok = new boolean[2][][];
```

```
/** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
 * [2][]]-ból az első dimenziót használni, mert vagy az egyikre
 * állítjuk, vagy a másikra. */
protected boolean [][] rács;
/** Megmutatja melyik rács az aktuális: [rácsIndex][][] */
protected int rácsIndex = 0;
/** Pixelben egy cella adatai. */
protected int cellaSzélesség = 20;
protected int cellaMagasság = 20;
/** A sejttér nagysága, azaz hányszor hány cella van? */
protected int szélesség = 20;
protected int magasság = 10;
/** A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
 * közötti valós idő. */
protected int várakozás = 1000;
// Pillanatfelvétel készítéséhez
private java.awt.Robot robot;
/** Készítünk pillanatfelvételt? */
private boolean pillanatfelvétel = false;
/** A pillanatfelvételek számozásához. */
private static int pillanatfelvételSzámláló = 0;
/**
 * Létrehoz egy <code>Sejtautomata</code> objektumot.
 *
 * @param szélesség a sejttér szélessége.
 * @param magasság a sejttér szélessége.
 */
public Sejtautomata(int szélesség, int magasság) {
    this.szélesség = szélesség;
    this.magasság = magasság;
    // A két rács elkészítése
    rácsok[0] = new boolean[magasság][szélesség];
    rácsok[1] = new boolean[magasság][szélesség];
    rácsIndex = 0;
    rács = rácsok[rácsIndex];
    // A kiinduló rács minden cellája HALOTT
    for(int i=0; i<rács.length; ++i)
        for(int j=0; j<rács[0].length; ++j)
            rács[i][j] = HALOTT;
    // A kiinduló rácsra "élőlényeket" helyezünk
    //sikló(rács, 2, 2);
    siklóKilövő(rács, 5, 60);
    // Az ablak bezárásakor kilépünk a programból.
    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
            setVisible(false);
            System.exit(0);
        }
    });
    // A billentyűzetről érkező események feldolgozása
```

```
addKeyListener(new java.awt.event.KeyAdapter() {
    // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük
    public void keyPressed(java.awt.event.KeyEvent e) {
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
            // Felezük a cella méreteit:
            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
            // Duplázzuk a cella méreteit:
            cellaSzélesség *= 2;
            cellaMagasság *= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel = !pillanatfelvétel;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
            várakozás /= 2;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
            várakozás *= 2;
        repaint();
    }
});
// Egér kattintó események feldolgozása:
addMouseListener(new java.awt.event.MouseAdapter() {
    // Egér kattintással jelöljük ki a nagyítandó területet
    // bal felső sarkát vagy ugyancsak egér kattintással
    // vizsgáljuk egy adott pont iterációit:
    public void mousePressed(java.awt.event.MouseEvent m) {
        // Az egérmutató pozíciója
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});
// Egér mozgás események feldolgozása:
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});
// Cellaméretek kezdetben
```



```
        szélesség*cellaSzélesség,
        magasság*cellaMagasság)) ;
    }
}

/** 
 * Az kérdezett állapotban lévő nyolcszomszédok száma.
 *
 * @param rács a sejttér rács
 * @param sor a rács vizsgált sora
 * @param oszlop a rács vizsgált oszlopa
 * @param állapot a nyolcszomszédok vizsgált állapota
 * @return int a kérdezett állapotbeli nyolcszomszédok száma.
 */
public int szomszédokSzáma(boolean [][] rács,
    int sor, int oszlop, boolean állapot) {
    int állapotúSzomszéd = 0;
    // A nyolcszomszédok végigzongorázása:
    for(int i=-1; i<2; ++i)
        for(int j=-1; j<2; ++j)
            // A vizsgált sejtet magát kihagyva:
            if(!((i==0) && (j==0))) {
                // A sejttérből szélénék szomszédai
                // a szembe oldalakon ("periódikus határfeltétel")
                int o = oszlop + j;
                if(o < 0)
                    o = szélesség-1;
                else if(o >= szélesség)
                    o = 0;

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

                if(rács[s][o] == állapot)
                    ++állapotúSzomszéd;
            }
    return állapotúSzomszéd;
}
/** 
 * A sejttér időbeli fejlődése a John H. Conway féle
 * életjáték sejtautomata szabályai alapján történik.
 * A szabályok részletes ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 171. oldal.)
 */
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 = szomszédochSzáma(rácsElőtte, i, j, ÉLŐ);

        if(rácsElőtte[i][j] == ÉLŐ) {
            /* Elő 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;
}
/** A sejttér időbeli fejlődése. */
public void run() {

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

        időFejlődés();
        repaint();
    }
}
/**
 * A sejttérbe "élőlényeket" helyezünk, ez a "sikló".
 * Adott irányban halad, másolja magát a sejttérben.
 * Az élőlény ismertetését lásd például a
 * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
 * matematikai játékok. Polygon, Szeged 1998. 172. oldal.)
 *
 * @param rács a sejttér ahová ezt az állatkát helyezzük
 * @param x a befoglaló téglalal bal felső sarkának oszlopa
 * @param y a befoglaló téglalal bal felső sarkának sora
 */

```

```
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;  
  
}  
/**  
 * A sejttérbe "élőlényeket" helyezünk, ez a "sikló ágyú".  
 * Adott irányban siklókat lő ki.  
 * Az élőlény ismertetését lásd például a  
 * [MATEK JÁTÉK] hivatkozásban /Csákány Béla: Diszkrét  
 * matematikai játékok. Polygon, Szeged 1998. 173. oldal./,  
 * de itt az ábra hibás, egy oszloppal told még balra a  
 * bal oldali 4 sejtes négyzetet. A helyes ágyú rajzát  
 * láasd pl. az [ÉLET CIKK] hivatkozásban /Robert T.  
 * Wainwright: Life is Universal./ (Megemlíthetjük, hogy  
 * minden kettő tartalmaz két felesleges sejtet is.)  
 *  
 * @param rács      a sejttér ahová ezt az állatkát helyezzük  
 * @param x          a befoglaló téglalap bal felső sarkának oszlopa  
 * @param y          a befoglaló téglalap bal felső sarkának sora  
 */  
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;

}

/** Pillanatfelvételek készítése. */
public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
    // A pillanatfelvétel kép fájlneve
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("sejtautomata");
    sb.append(++pillanatfelvételSzámláló);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(felvetel, "png",
            new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
// Ne villogjon a felület (mert a "gyári" update()
// lemeszelné a vászon felületét).
public void update(java.awt.Graphics g) {
```

```
    paint(g);
}
/**
 * Példányosít egy Conway-féle életjáték szabályos
 * sejttér obektumot.
 */
public static void main(String[] args) {
    // 100 oszlop, 75 sor mérettel:
    new Sejtautomata(100, 75);
}
}
```

A címből kiderült, hogy John Horton Conway alkotta meg ezt a játékot 1970-ben. Az addigi játékokhoz képest ez eltérő volt mivel ehez a játékhöz nem kellett felhasználó hanem akár elég volt egy számítógép is. Ez nem teljesen igaz mert a felhasználónak volt egy dolga mégpedig beállítani a kezdő pozíciót ahonnan indul maga a játék. Maga a játéktér egy saktáblához hasonlít ami bármekkora méretű lehet. Egy cellának 8 szomszédos cellája van. A szabályok : Ha a cella üres esetben a mellette lévő cella is üres lesz kivéve egy esetben amikor a 8-ból pontosan 3db cella foglalt. Ezesetben a következő cella is foglalt lesz. Következő szabály mikor egy cella foglalt, esetben a következő cella is foglalt lesz kivéve ha 2 vagy 3db cella foglalt, ekkor üresek lesznek a következők. Maga a játék kitalálója ezeket a cellákat elő illetve holt állípótúaknak titulálta a foglalt és üres helyet.

7.3. Qt C++ életjáték

Most Qt C++-ban!

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

Ennél a feladatnál ugyan az a lényeg mint az előzőnél csak míg annál JAVÁ-ban írtuk meg itt C++-ban kell megírni ezt. Maga a játék értelmezése és a szabályzatok detto ugyan azok. Annyi változás van még, hogy itt a QT-t kell segítségül hívni. Maga a QT egy keretrendszeres alkalmazás amit GUI-s alkalmazások és nem GUI-s programoknál használnak.

7.4. BrainB Benchmark

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

A nevéből adódoan az agyat veszi igénybe ez a "játék". Az e-sport területén használandó ez a szoftver mivel az adott játékos reakcióképességét képes felismeri és visszaadni. A játék lényege, hogy "Samu Entropy" karakterünket kell figyelnünk és, hogy a kék körben kell tartanunk a kurzorunkat. Ahogyan sikerül "level upolni" úgy egyre több doboz jelenik meg a képernyőn illetve ha ez nem lenne elég egyre gyorsabban is fognak mozogni. A játék során tudunk előre lépni illetve akár hátra is ha nem sikerült teljesíteni az adott "szintet". Elég egyszerű és könnyen használható kis program ami ad egy virtuális képet a reakcióképességekről ami az e-sportolók körében elég izgalmas téma.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

Megoldás videó: <https://youtu.be/j7f9SkJR3oc>

A megoldás forrása [Bátfai Norbert](#) tulajdona.

A szoftver lényege, hogy kézzel írott számjegyeket ismerjen fel a számítógép. Mesterséges intelligencia. Ebben az esetben MNIST adatbázisból vett számjegyeket fogunk felismertetni vele. Mivel most használjuk először a "Python-t" és a "TensorFlow-t" ezért ezt most telepítenünk kell. Maga "TensorFlow" nagyon hasznos program, sok nagyobb cég is nem hiába használja. Maga a gépi tanulásban játszik fontos szerepet. Jelen esetben ezt a kód csípetet nevezhetjük a "TensorFlow" Hello Worldjének. Ez az első programunk célja, hogy összeszorozzon két számot neurális hálókk felhasználásával. Mivel most kezdjük a Python használatát érdemes tudni, hogy nem "{" és "}" adják meg a blokkokat hanem tabulátoros behúzások.

8.2. Mély MNIST

Python

Megoldás videó:

1. passzolási lehetősségem (előadás)

8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndl8>

Megoldás forrása: https://bhaxor.blog.hu/2018/10/28/minecraft_steve_szemuvege

2. passzolási lehetősségem (előadás)

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

3. passzolási lehetősségem (előadás)

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

4. passzolási lehetősségem (előadás)

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_hackelete_a_scheme_programozasi_nyelv

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

Mint ahogy a feladat nevéből is kiderül a feladatunk nem más lesz mint, egy mandala kirajzolása a GIMP programban. Annyi a kikötés, hogy nem lehet grafikus felületet használni hanem kódossal kell dolgoznunk. Mint ahogy láthatjuk a mandala egy szimetrikus kör alapú kép. Először a szöveg hosszás és méretét határozzuk meg. Csak ezek után jön maga a mandala kirajzolása. Egy rétegen létrejön a szöveg és a szöveg betűtípusa is amit maga a felhasználó állított be. Amiért szimetrikus kör alapú képnek nevezük az azért van mert mikor létrejött a réteg tükrözi a program majd elfordítja és úgy is tükrözi. Így jön létre a szimetrikus kör alapú mandala. A legvégén mint eddig is már csak a kíratás / kirajzolás jelen esetben a kirajzolás marad.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

5. passzolási lehetősségem (előadás)

10.2. Programozás bevezetés

A faceebookon kiírt kutatásban való résztével 1. passzolási lehetősségem

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

10.3. Programozás

A faceebookon kiírt kutatásban való résztével 2. passzolási lehetősségem

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. OO szemlélet

A módosított polártranszformációs normális generátor beprogramozása Java nyelven. Mutassunk rá, hogy a mi természetes saját megoldásunk (az algoritmus egyszerre két normálist állít elő, kell egy példánytag, amely a nem visszaadottat tárolja és egy logikai tag, hogy van-e tárolt vagy futtatni kell az algot.) és az OpenJDK, Oracle JDK-ban a Sun által adott OO szervezés ua.! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG> (16-22 fólia) Ugyanezt írjuk meg C++ nyelven is! (lásd még UDPROG repó: source/labor/polargen)

Legelején az alap gondolatmenetet kell meghatároznunk. Ennél a feladatnál maga a matematika háttér részletezése nem fontos. A főbb szakasza a feladatnak az, hogy lefutás (egyszeri) után nem egy hanem kettő darab véletlenszerű számot kapunk. Ez azért lényeges mivel ilyenkor az előző futtatásnál tárolt egyik számot visszatudjuk adni a következő futtatáshoz. Processzor terhelés szempontjából jóval hasznosabb. Egy oszályt kell megírnunk ennek segítségére. JAVÁ-ban megírt kód két nagyobb részre osztható fel.

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

    public PolárGenerátor()
    {
        nincsTárolt = true;
    }

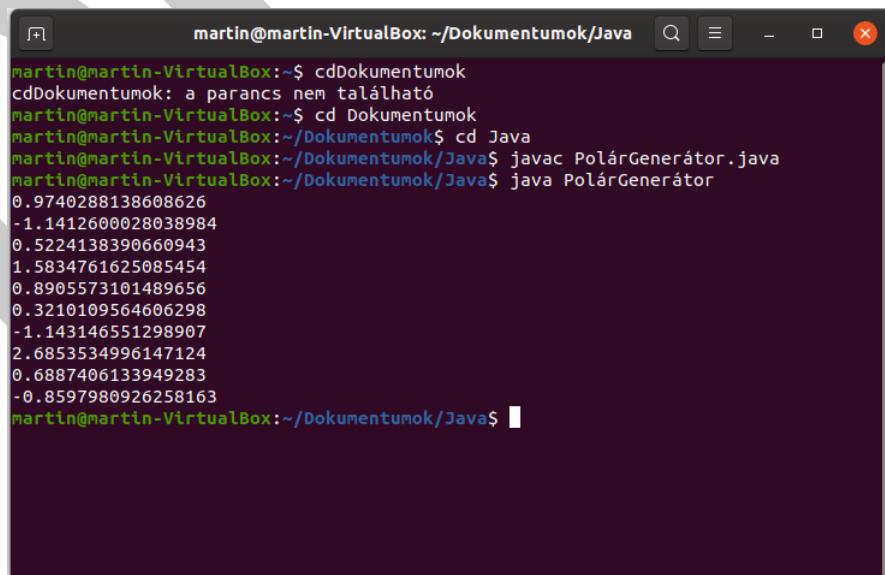
    public double következő()
    {
        if (nincsTárolt)
        {
            double u1, u2, v1, v2, w;
            do
            {
                u1 = Math.random();
                u2 = Math.random();
                v1 = 2 * u1 - 1;
                v2 = 2 * u2 - 1;
                w = Math.sqrt(u1 * u1 + u2 * u2);
                if (w > 0.707)
                    w = 0.707;
                else if (w < 0.707)
                    w = -0.707;
                else
                    w = 0;
                if (v1 * v2 - u1 * u2 < 0)
                    w = -w;
            } while (w == 0);
            nincsTárolt = false;
            tárolt = w;
        }
        return tárolt;
    }
}
```

```
        v2 = 2 * u2 - 1;
        w = v1 * v1 + v2 * v2;
    }
    while (w >1);
    double r = Math.sqrt((-2 * Math.log(w)) / w);
    tárolt = r * v2;
    nincsTárolt = !nincsTárolt;
    return r * v1;
} else
{
    nincsTárolt = !nincsTárolt;
    return tárolt;
}
}
```

boolean egy változó ami visszatér (akár TRUE / FALSE) egy értékkel, hogy van-e már kiszámolt véletlenszámunk. double ez is egy változó ami az értéket menti el. Ezután található meg, a feladat lényege, hogy visszaadjunk egy számot a következő számításhoz. Ez az egész egy osztályban található. Most következik a main függvényben az osztály meghívása.

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

Itt a meghíváson kívül még leteszteljük a programot. 10-szer fogjuk meghívni ami során 5-ször fog számítást végrehajtani mivel minden második zajlik le(1 lefut következőnél az előző számot veszi figyelembe és így tovább 5-ször). Lássuk az eredményt.



```
martin@martin-VirtualBox: ~/Dokumentumok/Java
martin@martin-VirtualBox:~$ cdDokumentumok
cdDokumentumok: a parancs nem található
martin@martin-VirtualBox:~$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ javac PolárGenerátor.java
martin@martin-VirtualBox:~/Dokumentumok/Java$ java PolárGenerátor
0.9740288138608626
-1.1412600028038984
0.5224138390660943
1.5834761625085454
0.8905573101489656
0.3210109564606298
-1.143146551298907
2.6853534996147124
0.6887406133949283
-0.8597980926258163
martin@martin-VirtualBox:~/Dokumentumok/Java$
```

Eddig JAVA-ban volt most következzen a C++ verziója. Lényegében ugyan az mivel a feladat is ugyan az. Szintaktikai eltérések vannak a kettő között javarészt viszont még annyit hozzátnék, hogy C++-ban az osztályon kívül van a main.

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

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

private:
    bool nincsTarolt;
    double tarolt;

};

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

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

        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

        return r * v1;
    }
}
```

```

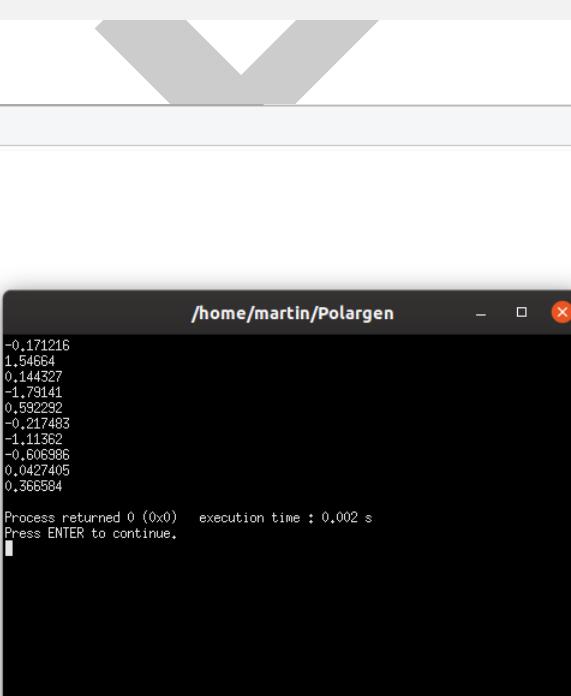
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
int
main (int argc, char **argv)
{
    PolarGen pg;

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

    return 0;
} }

```

Lássuk az eredményt újra.



```

/home/martin/Polargen - 0.171216
1.54664
0.144327
-1.79141
0.592292
-0.217483
-1.11362
-0.606986
0.0427405
0.366584

Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.

```

11.2. „Gagyí”

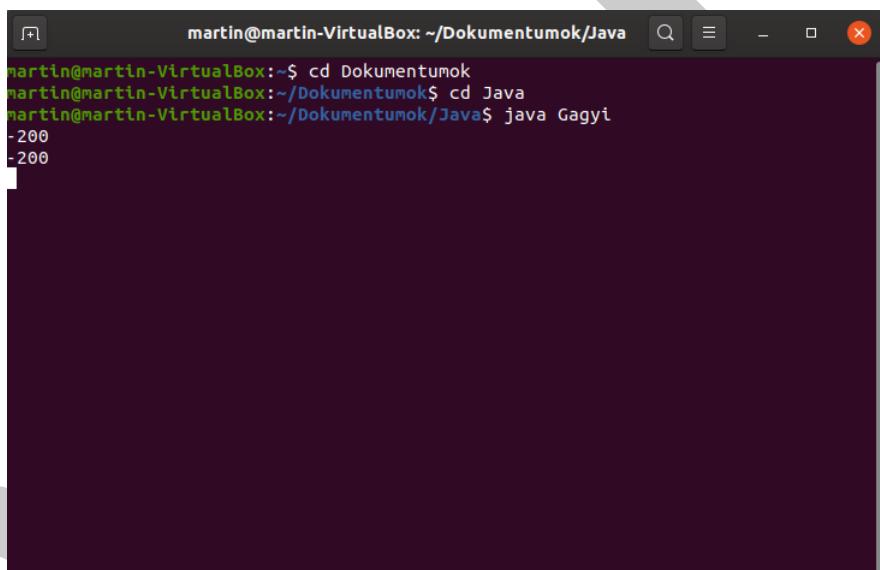
Az ismert formális „while ($x \leq t \wedge x \geq t \wedge t \neq x$)” tesztkérdéstípusra adj a szokásosnál (miszerint x , t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciaja) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x , t értékekkel meg nem! A példát építsd a JDK Integer.java forrására3 , hogy a 128-nál inkluzív objektum példányokat poolozza!

Mint ahogy olvashattuk a leírásban végtelen ciklust két esetben kaphatunk ennél a tesztkérdésnél(while ($x \leq t \wedge x \geq t \wedge t \neq x$);]). Egyik eset az amikor az objektum által hordozott érték, míg a másiknál az

objektum referenciajára. Jó tisztázni az `Integer` fogalmát. Amikor egy ilyen objektumot létrehozunk abban az esetben ha két intervallumba tesszük meg (-128 - 127) poolból fogunk értéke visszakapni. Például ha x-nek illetve y-nak 0 értéket adunk ezesetben mind a kettő változónál ugyan az lesz az objektum. Viszont ha ez az intervallumon kívülre esik a megadott szám akkor egy végtelen ciklust fogunk kapni. Lássuk is a példaprogramot :

```
public class Gagyi
{
    public static void main (String[]args)
    {
        Integer x = -200;
        Integer t = -200;
        System.out.println (x);
        System.out.println (t);
        while (x <= t && x >= t && t != x);
    }
}
```

Eredmény :



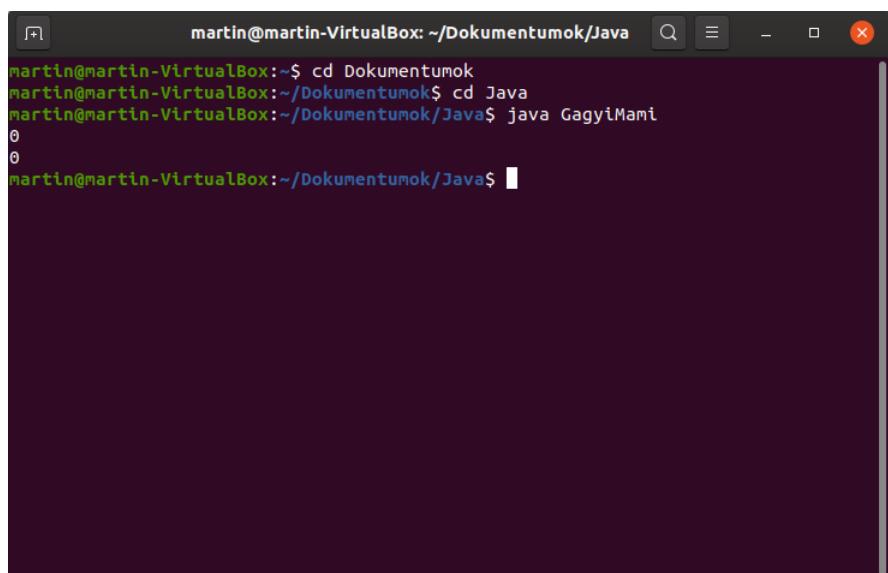
```
martin@martin-VirtualBox:~/Dokumentumok$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ java Gagyi
-200
-200
```

Itt láthatjuk, hogy egy végtelen ciklust kaptunk mivel az intervallum külsejére esett a megadott (-200) szám. Két egymástól eltérő objektum címűnk is lesz.

Most lássuk azt az esetet amikor is az adott szám belesik az intervallumba.

```
public class GagyiMami
{
    public static void main (String[]args)
    {
        Integer x = 0;
        Integer t = 0;
        System.out.println (x);
        System.out.println (t);
        while (x <= t && x >= t && t != x);
    }
}
```

Eredmény :



A screenshot of a terminal window titled "martin@martin-VirtualBox: ~/Dokumentumok/Java". The terminal shows the following command and its output:

```
martin@martin-VirtualBox:~$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ java GagyMami
0
0
martin@martin-VirtualBox:~/Dokumentumok/Java$
```

Itt láthatjuk, hogy ugyan az az egész kód csak a szám más. Ez nem végtelen ciklus illetve ugyan az lesz a memóriacíműk is.

11.3. Yoda

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

Ezt a feladatot összelehet kötni a jól ismert "Star Wars" filmben szereplő "Yoda" karakterével. Nem hiába az elnevezése mivel neki a beszéde volt kicsit fura mert nem a megszokott szintaktikát használta hanem eltért erről. Míg a filmben "Yoda" tárgy-alany-ige így teszi össze a mondatait addig mi a konstansot írjuk át a bal oldalra összehasonlításnál, a változót pedig jobbra. Akkor csináltuk jól a feladatot ha leállt a program amikor nem követtük Yoda szabályát. Ezzel a módszerrel a sok időt tudunk megspórolni hibakeresés során. Nulla pointerhez nem lehet metódust írni.

```
class yoda
{
    public static void main(String[] args)
    {
        String marka = null;

        if (marka.equals("Nike"))
            System.out.println("Helyes megfejtés.");
    }
}
```

Eredmény :

```
Exception in thread "main" java.lang.NullPointerException  
at yoda.main(yoda.java:24)
```

Jól csináltuk mivel leállt a program futtatása. Ez a gondolat menet hasznos lehet hibák keresésénél is.

DRAFT

12. fejezet

Helló, Liskov!

12.1. Liskov helyettesítés sértése

Írunk olyan OO, leforduló Java és C++ kódcsipetet, amely megséríti a Liskov elvet! Mutassunk rá a megoldásra: jobb OO tervezés. https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (93-99 fólia) (számos példa szerepel az elv megsértésére az UDPROG repóban, lásd pl. source/binom/Batfai-Barki/madarak/)

Az objektumorientált programozásnak öt alapelve van amik közül az egyik a Liskov helyettesítés. Először is magát a "Liskov elvet" kell rendben tenni. Ha "S" osztály "T" osztály leszármazottja, akkor "S" szabadon behelyettesíthető minden olyan helyre ,ahol "T" típust várunk. Hozzátéve, hogy ilyenkor az "S" osztály örökli az összes tulajdonságát a "T" osztálynak. Ez program lefutás céljából tökéletes viszont maga a tartalom nem minden esetben igaz. Kezdjük C++-ban a kódcsípetek bemutatását.

```
#include <iostream>
class Madar {
public:
virtual void repul() {};
};

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

}
```

Itt látszódik, hogy 2 darab osztályból fog állni a "P program". Most következik, hogy létrehozzuk az "S" osztályakat amik lesznek a leszármazottak.

```
class Sas : public Madar
{
};

class Pingvin : public Madar
{
void repul() {
```

```

std::cout<<"A pingvin nemtud repülni.\n";
}
};

int main ( int argc, char **argv )
{
Program program;
Madar madar;
program.fgv ( madar );
Sas sas;
program.fgv ( sas );
Pingvin pingvin;
program.fgv ( pingvin ); }
```

The screenshot shows a code editor window titled "Liskov.cpp" containing C++ code. The code defines four classes: Madar, Program, Sas, and Pingvin, each with a repul() method. The Pingvin class overrides the repul() method to output a specific message. The main() function creates instances of these classes and calls their fgv() methods. To the right of the editor is a terminal window showing the execution of the program and its output.

```

Liskov.cpp ✘
1 #include <iostream>
2 class Madar {
3 public:
4     virtual void repul() {};
5 };
6 class Program {
7 public:
8     void fgv ( Madar &madar ) {
9         madar.repul();
10    }
11 }
12 class Sas : public Madar
13 {
14 };
15 class Pingvin : public Madar
16 {
17     void repul() {
18         std::cout<<"A pingvin nemtud repülni.\n";
19     }
20 };
21 int main ( int argc, char **argv )
22 {
23     Program program;
24     Madar madar;
25     program.fgv ( madar );
26     Sas sas;
27     program.fgv ( sas );
28     Pingvin pingvin;
29     program.fgv ( pingvin ); }
```

/home/martin/Dokumentumok/C++/Liskov -

A pingvin nemtud repülni.
Process returned 0 (0x0) execution time : 0,002 s
press ENTER to continue.

Az utolsó sorból látszik a feladatunk, hogy sérti a Liskov elvet mivel a függvény röptetné a pingvint, de ez lehetetlen a való életben. Most ezt együk át JAVA-ba.

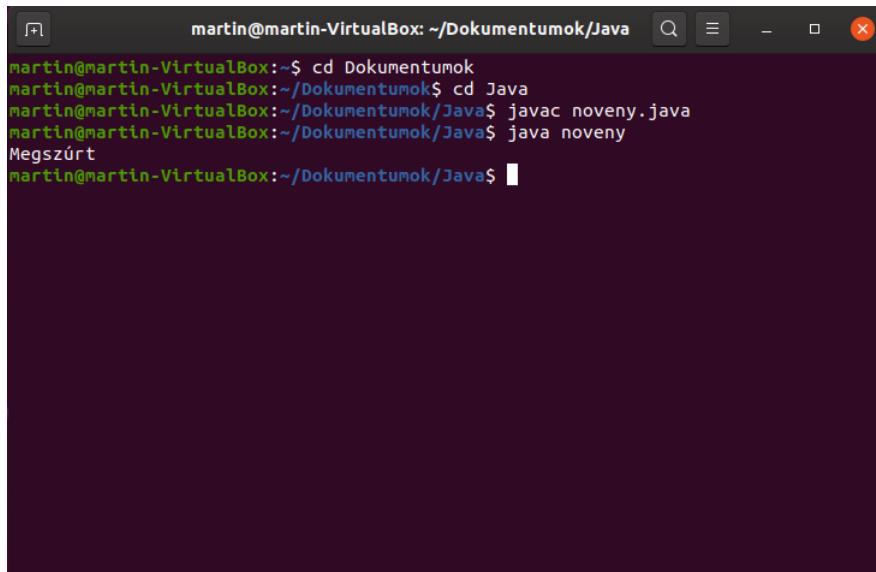
```

class noveny{
public static void main(String[] args){
    Kaktusz r = new Kaktusz();
    Program.fgv(r);
}

class Virag{
    void szur(){
        System.out.print("Megszúrt");
    }
}
class Tulipan extends Virag{
}
class Kaktusz extends Virag{
}
class Program{
    public static void fgv(Virag v){
        v.szur();
    }
}
```

{}

A feladat ugyan az mint az előzőnél volt. Létrehoztuk egy ōs osztályt amin belül kettő darab alosztályt. Mivel köztudottan a Tulipan nem szúr ezért sérteni fogja az elvet.



The screenshot shows a terminal window titled "martin@martin-VirtualBox: ~/Dokumentumok/Java". The terminal output is as follows:

```
martin@martin-VirtualBox:~$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ javac noveny.java
martin@martin-VirtualBox:~/Dokumentumok/Java$ java noveny
Megszűrt
martin@martin-VirtualBox:~/Dokumentumok/Java$
```

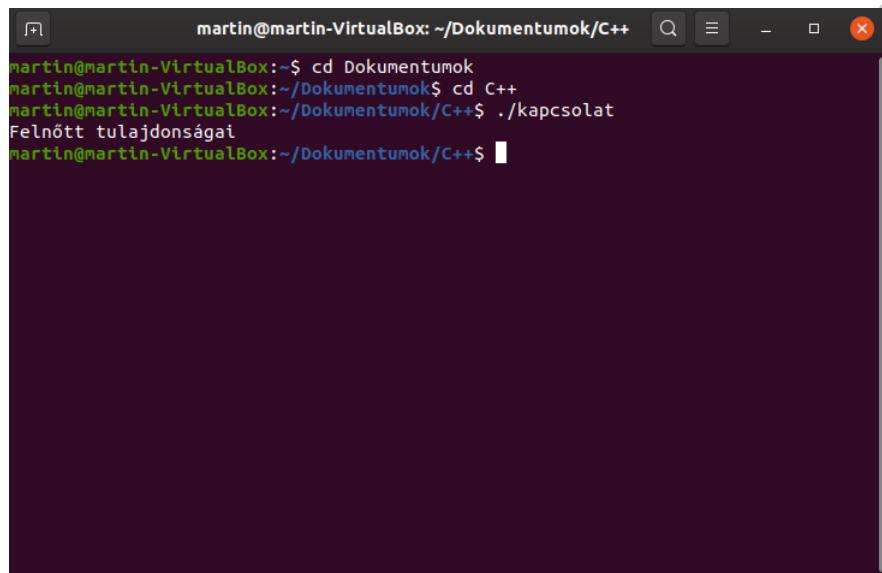
12.2. Szülő-gyerek

Írunk Szülő-gyerek Java és C++ osztálydefiníciót, amelyben demonstrálni tudjuk, hogy az ōson keresztül csak az ōs üzenetei küldhetőek! https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog2_1.pdf (98. fólia)4

A feladat megkezdésénél létre kell hoznunk egy ōs osztály aminek a neve most "Szulo" lesz. Ennek leszármazottjaként készítünk egy "Gyerek" alosztályt. Mint ahogy feljebb már említettem, ilyenkor az alosztály öröklí az ōs osztály tulajdonságait. Kiegészítve ezt még, hogy az ōs osztály viszont nem bővül, tehát egyoldalú csak ez az öröklődés. Miután létrejöttek az osztályok, meghívjuk az alosztályt ōs osztályként. Viszont ekkor egy hibát fogunk kapni mivel ez nem lesz lehetséges. Ennek a logikája hasonlít a matematikába lévő : minden rombusz paralelogramma, de nem minden paralelogramma rombusz. Lássuk is a példát C++ -ban.

```
#include <iostream>
class szulo {
public:
void uzenet()
{
std::cout << "Felnőtt tulajdonságai\n";
}
};
class gyerek : public szulo {
void uzenet1()
{
std::cout << "Gyerek tulajdonságai\n";
}
};
int main()
```

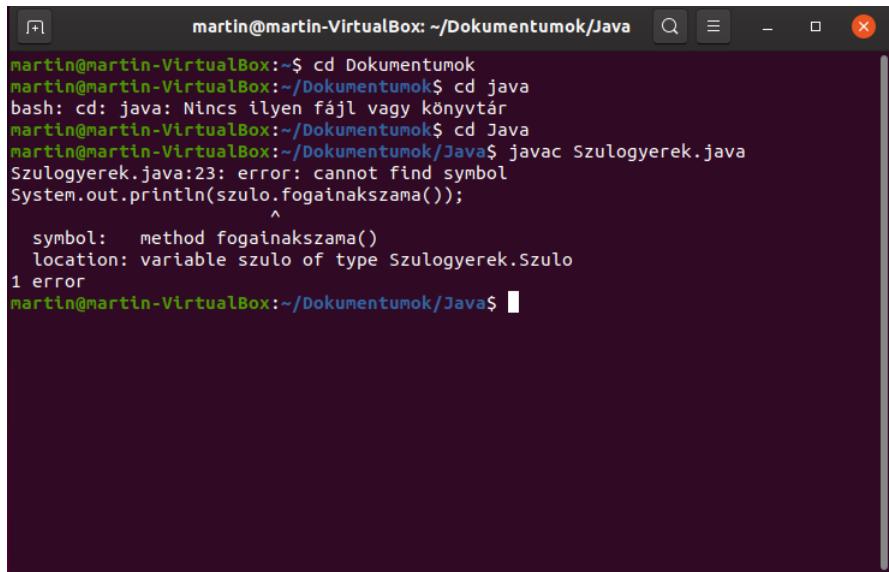
```
{  
szulo * felnott = new gyerek();  
felnott -> uzenet1();  
}  
}
```



Most nézzük JAVA-ban Miután lefuttattuk a programot akkor fogjuk sikeresen megcsinálni a feladatot ha hibát kapunk.

```
class Szulogyerek {  
class Szulo {  
int kor, nem;  
boolean dolgozik;  
Szulo() {  
kor = 32;  
nem = 1;  
dolgozik = true;  
}  
};  
class Gyerek extends Szulo {  
public int fogainakszama;  
Gyerek() {  
fogainakszama = 15;  
kor = 6;  
nem = 2;  
dolgozik = false;  
}  
}  
public static void main(String[] args) {  
Szulogyerek szu = new Szulogyerek();  
Szulo szulo = szu.new Gyerek();  
System.out.println(szulo.fogainakszama);  
}  
};
```

A két alosztályt elláttunk pár tulajdonsággal. A main-be viszont már errorr fogunk kapni mivel nemtudjuk meghívni az alosztályt ős osztályként.



The screenshot shows a terminal window titled "martin@martin-VirtualBox: ~/Dokumentumok/Java". The user runs "cd Dokumentumok", "cd java", and then "javac Szulogyerek.java". The compilation fails with the following error message:

```
martin@martin-VirtualBox:~/Dokumentumok$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd java
bash: cd: java: Nincs ilyen fájl vagy könyvtár
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ javac Szulogyerek.java
Szulogyerek.java:23: error: cannot find symbol
System.out.println(szulo.fogainakszama());
                           ^
      symbol:   method fogainakszama()
     location: variable szulo of type Szulogyerek.Szulo
1 error
martin@martin-VirtualBox:~/Dokumentumok/Java$
```

12.3. Ciklomatikus komplexitás

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

Itt nem egy programot kell írni hanem egy programnak a ciklomatikus komplexitását kell kiszámolni. Konyhanyelvre átvéve a kód bonyolultságát számolja ki nekünk. Elképzelés szerint hasonlíthatjuk egy családfa rajzára is, ahol elágazások vannak amiknek köszönhetően "nő" a bonyolultság, és ezt számolja ki. Kisebb programokra kézzel is kiszámítható egy képlet alapján. $M = E - N + 2P$. M = Ciklomatikus szám E = Az adott gráf éleinek száma N = Az adott gráfban lévő csúcsok száma P = Az összefüggő komponensek száma Viszont ezt már nagyobb programknál nem tehetjük meg ezért hoztak létre több ilyet is. Én jelen esetben a "Lizard.ws" nevezetű szoftvert fogom alkalmazni. Eredménye egy konkrét szám lesz, így könnyen kiolvasható. Ez az egész kiszámítás a gráfelméletre építette McCabe ("feltaláló"). Az adott programom legyen most a "LZWBinFa" JAVA-ban megírt változata. " <http://www.lizard.ws/> " online kalkulátorral számoltam ki az értékeket.

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

13. fejezet

Helló, Mandelbrot!

13.1. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Nevéből adódoan ebben a feladatban UMLben kell megtervezni az osztályokat majd ebből generálunk egy kódot. Természetesen nem az egész és kész kódot fogjuk megkapni hanem csak egy vázat ami szerint eltudunk indulni belőle. Magáról az UML-ről lentebb olvashatunk részletesebben.



```
#include "LZWBinFa.h"

LZWBinFa::LZWBinFa() {
    // TODO - implement LZWBinFa::LZWBinFa
    throw "Not yet implemented";
}

void LZWBinFa::kiir() {
    // TODO - implement LZWBinFa::kiir
    throw "Not yet implemented";
}

int LZWBinFa::getMelyseg() {
    return this->melyseg;
}

double LZWBinFa::getAtlag() {
    return this->atlag;
}
```

```
double LZWBinFa::getSzoras() {
    return this->szoras;
}

void LZWBinFa::kiir(std::ostream& os) {
    // TODO - implement LZWBinFa::kiir
    throw "Not yet implemented";
}

LZWBinFa::LZWBinFa(const LZWBinFa& unnamed_1) {
    // TODO - implement LZWBinFa::LZWBinFa
    throw "Not yet implemented";
}

void LZWBinFa::kiir(Csomopont* elem, std::ostream& os) {
    // TODO - implement LZWBinFa::kiir
    throw "Not yet implemented";
}

void LZWBinFa::szabadit(Csomopont* elem) {
    // TODO - implement LZWBinFa::szabadit
    throw "Not yet implemented";
}

void LZWBinFa::rmelyseg(Csomopont* elem) {
    // TODO - implement LZWBinFa::rmelyseg
    throw "Not yet implemented";
}

void LZWBinFa::ratlag(Csomopont* elem) {
    // TODO - implement LZWBinFa::ratlag
    throw "Not yet implemented";
}

void LZWBinFa::rszoras(Csomopont* elem) {
    // TODO - implement LZWBinFa::rszoras
    throw "Not yet implemented";
}
```

Itt a BINFÁ-ból egy forrást kaptunk viszont ez csak egy váz és rövídebb.

```

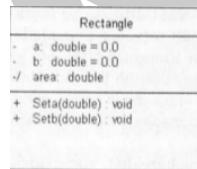
Binfa
- fa : Node*
+ gyoker : Node
- masol(elem : Node*, regifa : Node*) : Node*
+ Binfa() «constructor»
+ Binfa(regi : const Binfa&) «constructor»
+ Binfa( : Binfa) «constructor»
+ operator =(regi : const Binfa&) : Binfa&
+ operator =(: Binfa) : Binfa&
+ operator <<(c : char)
+ preorder(elem : Node*, depth : int)
+ inorder(elem : Node*, depth : int)
+ postorder(elem : Node*, depth : int)
+ destroy_tree(elem : Node*)

```

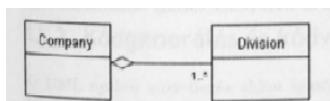
13.2. Egy esettan

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

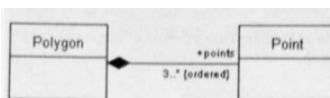
Ebben a feladatban a modellezés és a C++ témát fogjuk kitárgyalni. Legfőképp az UML (Unified Modeling Language) modellezőnyelvvel fogunk találkozni. Ez a nyelv a jelenlegi a szoftverfejlesztés egyetlen szabványos nyelve, ez okból rengeteg eszközöt támogat. Ez a modellezőnyelv legfőbb területe az osztálydiagramok. Ezek a diagramok osztályokat, interfészeket illetve egyéb típusokat, kapcsolatokat jeleníti meg. Jelkészlete a dobozok, vonalak, ikonok, szövegek. Először is taglaljuk kicsit az osztályokat.



Nevéből adódoan ez a legfontosabb elem az osztálydiagramokban. Jelölése egy téglalap. 3 részre oszthatjuk ezt : 1. Osztály név 2. Osztály attribútuma(i) 3. Osztály művelete(i) A legfelső rész minden látszik még az alsó 2 réteget eltudjuk távolítani a megjelenések közül. A középső réteg szintaxisa : Láthatóság név : Típus multiplicitás = Alapértelmezett érték {Tulajdonság(ok)} Neven kívül semmi sem kötelező innen viszont a láthatóságnak négy fajtája van. 1. + azaz public 2. - azaz private 3. # azaz protected 4. ~ azaz package (C++-on kívül) Most jöjjön az asszociációk. Jelölése egy vonal. A vonalra ráírva adjuk meg a nevet. Az asszociáció attribútumait az asszociációs osztály modellezzi, jelölése szaggatott vonal. Két fajta specializált fajtája van névszerint az aggregáció és a kompozíció. Az aggregáció tartalmazást jelöl, jelölése egy üres rombusz.



Nem sűrűn használt fajta mivel szinte semmi jelentőssége nincs. A kompozíció viszont már két fontos dolgot is jelölhet : 1. A tartalmazott objektum PONTOSAN egy tartalmazó objektumhoz tartozik 2. A tartalmazott és a tartalmazó együtt jön létre és együtt szűnik meg. Jelölése egy telt rombusz.



Most az esettan működését mutatom meg. Ebben a feladatban Dékány Róbert tanítványa voltam.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
nagy@nagy-VirtualBox:~$ cd bhax
nagy@nagy-VirtualBox:~/bhax$ cd esettan
nagy@nagy-VirtualBox:~/bhax/esettan$ ./eset
Test1: create inventory and printing it to the screen.
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20191007, Age: 0, Current price: 30000,
2
1.: Type: HardDisk, Name: WD, Initial price: 25000, Date of acquisition: 20191007, Age: 0, Current price: 25000,
Press any key to continue...

Test2: loading inventory from a file (computerproducts.txt), printing it, and then writing it to a file (computerproducts_out.txt)
End of reading product items.The content of the file is:
0.: Type: Display, Name: TFT1, Initial price: 30000, Date of acquisition: 20011001, Age: 6579, Current price: 24000
: 13
1.: Type: Display, Name: TFT2, Initial price: 35000, Date of acquisition: 20060930, Age: 4754, Current price: 28000
: 10
2.: Type: ComputerConfiguration, Name: ComputerConfig1, Initial price: 70000, Date of acquisition: 20060930, Age:
Items:
0. Type: Display, Name: TFT3, Initial price: 30000, Date of acquisition: 20011001, Age: 6579, Current price: 24000
: 13
1. Type: HardDisk, Name: WesternDigital, Initial price: 35000, Date of acquisition: 20060930, Age: 4754, Current
3.: Type: HardDisk, Name: Maxtor, Initial price: 25000, Date of acquisition: 20050228, Age: 5333, Current price: 20000

The content of the inventory has been written to computerproducts_out.txt

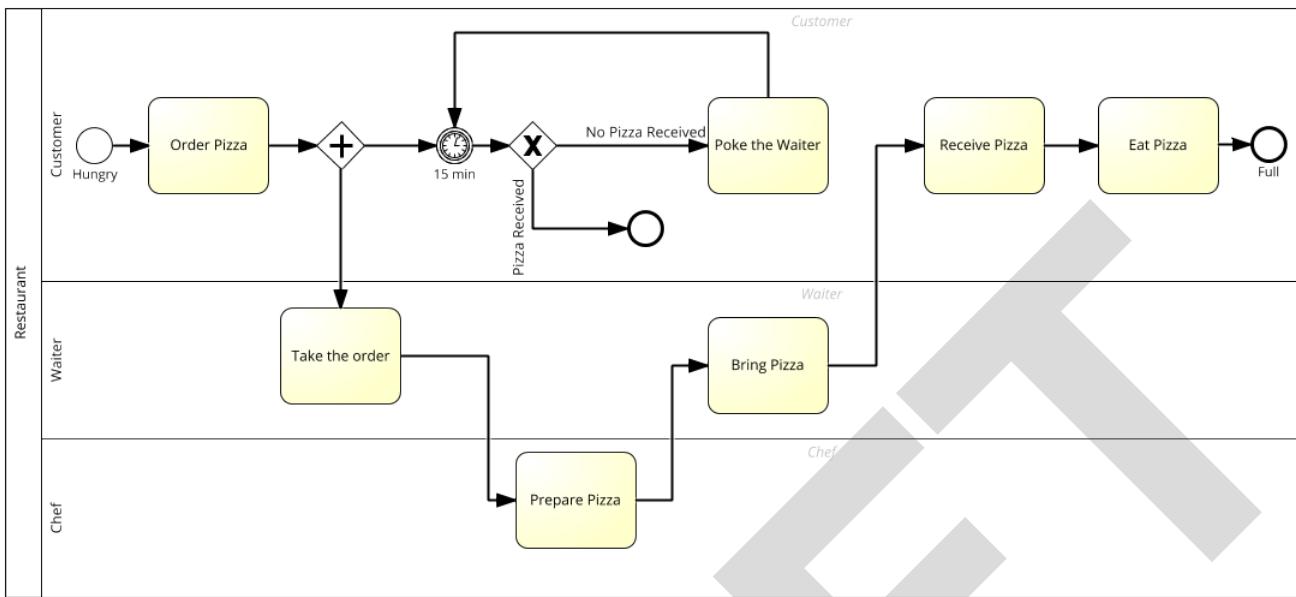
Done.
nagy@nagy-VirtualBox:~/bhax/esettan$
```

```
d TFT1 30000 20011002 12 13
d TFT2 35000 20061001 10 10
c ComputerConfig1 70000 20061001 2
d TFT3 30000 20011002 12 13
h WesternDigital 35000 20061001 7000
h Maxtor 25000 20050301 7000
```

13.3. BPMN

Rajzolunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog> (34-47 fólia)

Maga mint, BPMN (Business Process Model and Notation) nem más mint egy üzleti folyamatmodellezési szabvány, amely grafikus jelölést biztosít. Tehát egy üzleti diagramm. Főbb célja, hogy könnyebben átláthatóak legyenek a bonyolultabb dolgok is. Mint ahogy feljebb írtam üzleti folyamatmodellezési szabvány, ebből adódoan, csak az üzleti folyamatokra alkalmazható modellezés fogalmait veszi elő. Négy fajta "alappillér" található meg : 1. Áramlási objektumok. 2. Objektumot összekapcsolása. 3. Útvonalak. 4. Leletek. Mivel nincsen sok fajta varriáció így ezek a diagrammok kiolvasása egyszerűbb a megszokottnál. Rengeteg ilyen fajta diagramm van a neten amiből egyet válaszottam bemutatás jellegén :



Látszik, hogy 3 fajta sáv van : Vevő,Pincér,Séf. Mindegyiknek más lesz a dolga. A kiinduló pont maga az, hogy éhes a Vevő, ekkor rendel egy pizzát. Ezt követően fog elindulni a diagramm a pincérnél aki felveszi azt, majd tovább adja a séfnek, hogy készítse el. Közben ugyan úgy fut a Vevőnél is a szál és addig ismétlődik a folyamat míg meg nem kapja a pizzát azaz míg nem végez a Pincér és a séf. Teljesen érthető diagramm és kivállóan van szemléltetve is.

14. fejezet

Helló, Chomsky!

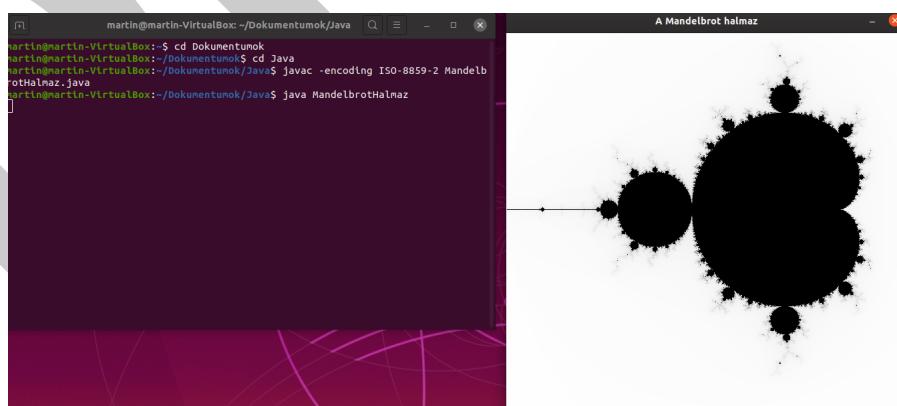
14.1. Encoding

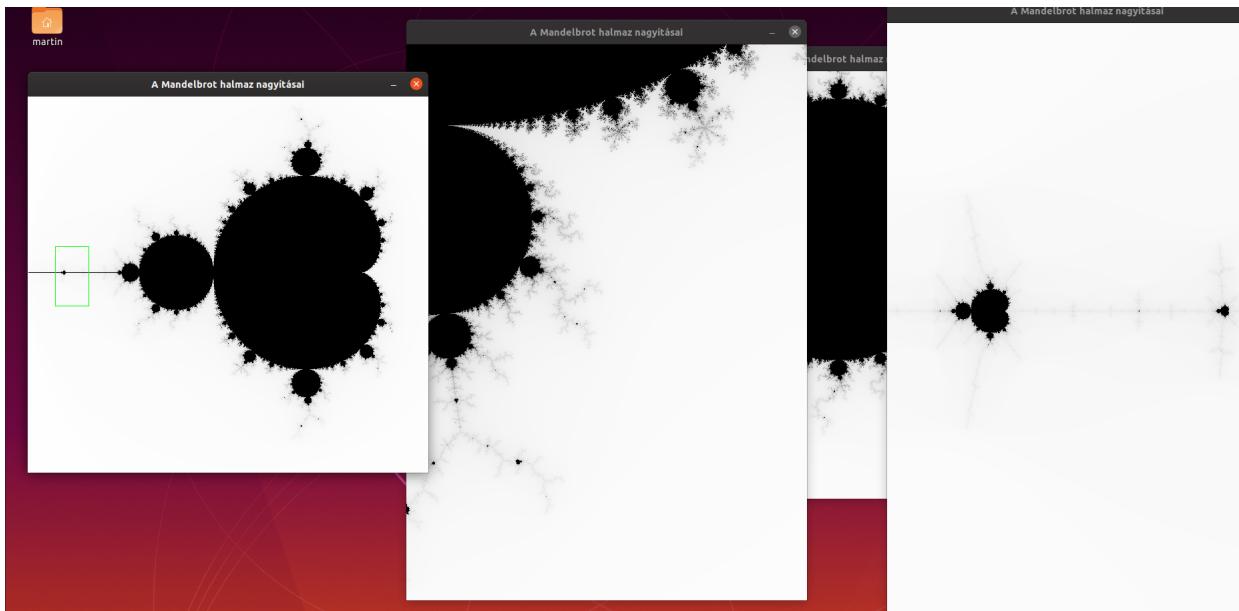
Fordítsuk le és futtassuk a Javat tanítok könyv MandelbrotHalmazNagyító.java forrását úgy, hogy a fájl nevekben és a forrásokban is meghagyjuk az ékezetes betűket! <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/adatok.html>

Maga a feladat egyszerűen értelmezhető. Egy program fordítása és futtatása. Annyi a kitérő benne, hogy a magyar billentyűzetet fogunk használni benne azaz ékezeteket meghagyjuk. Mint ahogy a feladat írta JAVA-ban kell ezt véghez vinni és ez nem véletlen. Más programnyelvekben nem lehetséges / jelentős bonyodalmat okozna ez, de a JAVA kezelni tudja. A enkódolást (encode) kell átállítanunk UTF-8-ra. Ez a lépés nem szükséges amennyiben az alapértelmezett kódolást már alapból ez. Ez inkább egy érdekesebb feladat volt mint hasznos, mivel külföldi programozók-nak feltehetőleg más az alapértelmezett kódolást ezáltal hibákat fog kiírni futtatás során.

```
javac -encoding ISO-8859-2 MandelbrotHalmaz.java  
java MandelbrotHalmaz
```

Így tudjuk lefordítani és futtatni a programot.





14.2. OOCWC lexer

Izzítsuk be az OOCWC-t és vázoljuk a <https://github.com/nbatfai/robocar-emulator/blob/master/justine/rcemu/src/> lexert és kapcsolását a programunk OO struktúrájába!

Maga a szó jelentése Robocar World Championship. Magyarra fordítva a robotautók terjedésének a vizsgálata. Másik fontos dolog a `lexer` szó jelentése. A feladata nem más mint, hogy konkrét szabályok mellett képesek legyünk előállítani szöveges állományokat.

```
%option c++
%option noyywrap
```

Itt látszik, hogy C++ lesz a nyelvünk illetve a `yywrap` nevű függvényre nem lesz szükségünk.

```
% {
#define YY_DECL int justine::robocar::CarLexer::yylex()
#include "carlexer.hpp"
#include <cstdio>
#include <limits>
}

INIT "<init"
INITG "<init guided"
WS [ \t]*
WORD [^-:\n \t()]{2,}
INT [0123456789]+
FLOAT [-.0123456789]+
ROUTE "<route"
CAR "<car"
POS "<pos"
GANGSTERS "<gangsters"
```

```
STAT  "<stat"
DISP  "<disp>"
%%
```

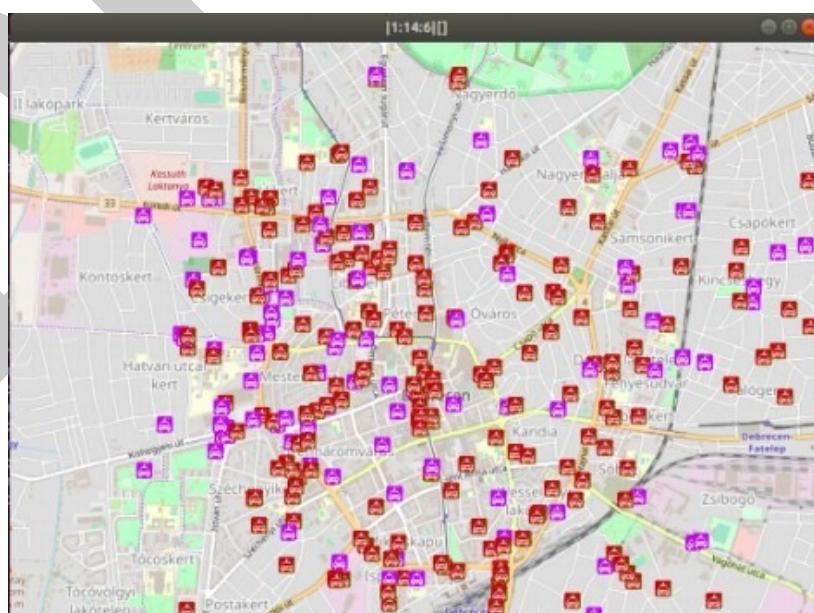
Ez a carlexer.ll definíciós része. Konkrét, előre megadott reguláris kifejezések találhatóak meg itt. Ezek fognak segíteni később a szabályoknál.

```
{DISP}          {
    m_cmd = 0;
}
{POS}{WS}{INT}{WS}{INT}{WS}{INT}  {
    std::sscanf(yytext, "<pos %d %u %u", &m_id, &from, &to);
    m_cmd = 10001;
}
{CAR}{WS}{INT}      {
    std::sscanf(yytext, "<car %d", &m_id);
    m_cmd = 1001;
}
{STAT}{WS}{INT}      {
    std::sscanf(yytext, "<stat %d", &m_id);
    m_cmd = 1003;
}
{GANGSTERS}{WS}{INT}      {
    std::sscanf(yytext, "<gangsters %d", &m_id);
    m_cmd = 1002;
}
{ROUTE}{WS}{INT}{WS}{INT}({WS}{INT})* {
    int size{0};
    int ss{0};
    int sn{0};

    std::sscanf(yytext, "<route %d %d%n", &size, &m_id, &sn);
    ss += sn;
    for(int i{0}; i<size; ++i)
    {
        unsigned int u{0u};
        std::sscanf(yytext+ss, "%u%n", &u, &sn);
        route.push_back(u);
        ss += sn;
    }
    m_cmd = 101;
}
{INIT}{WS}{WORD}{WS}("c"|"g") {
    std::sscanf(yytext, "<init %s %c>", name, &role);
    num = 1;
    m_cmd = 0;
}
{INIT}{WS}{WORD}{WS}{INT}{WS}("c"|"g") {
    std::sscanf(yytext, "<init %s %d %c>", name, &num, &role);
    if(num >200)
```

```
{  
    m_errnumber = 1;  
    num = 200;  
}  
m_cmd = 1;  
}  
  
{INITG} {WS} {WORD} {WS} ("c"|"g") {  
    std::sscanf(yytext, "<init guided %s %c>", name, &role);  
    num = 1;  
    m_guided = true;  
    m_cmd = 3;  
}  
  
{INITG} {WS} {WORD} {WS} {INT} {WS} ("c"|"g") {  
    std::sscanf(yytext, "<init guided %s %d %c>", name, &num, &role);  
    if(num >200)  
    {  
        m_errnumber = 1;  
        num = 200;  
    }  
    m_guided = true;  
    m_cmd = 2;  
}  
.  
{ ; }  
%%  
  
int yyFlexLexer::yylex(){return -1;}
```

Itt találhatóak meg maguk a szabályok. Látszik, hogy egy-egy "Szakaszra" milyen reguláris kifejezések adottak. `sscanf` függvény segítségét kihasználjuk még itt. Az utolsó sorban még egy függvény definíálást láthatunk adott esetben a `yylex`-t.



14.3. I334d1c45

Írj olyan OO Java vagy C++ osztályt, amely leet cipherként működik, azaz megvalósítja ezt a betű helyettesítést: <https://simple.wikipedia.org/wiki/Leet> (Ha ez első részben nem tettet meg, akkor írasd ki és magyarázd meg a használt struktúratömb memória foglalását!)

```
#include<iostream>
#include<fstream>
#include<vector>
#include<string>
#include <algorithm>
#include <cctype>
using namespace std;

class LeetCipher {
public:
    LeetCipher();
    string decipher(string& text);
private:
    vector<vector <string>> leetabc;
    vector<string> abc;
};

LeetCipher::LeetCipher() {
    srand (time(NULL));
    leetabc.resize(26);
    leetabc[0]= {"4", "/-||", "/_||", "@", "/||"};
    leetabc[1]={ "8", "|3", "13", "|}", "|:", "|8", "18", "6", "|B", "|8", "lo", ←
    "|o"};
    leetabc[2]= { "<", "{", "[", "("};
    leetabc[3] = {"|)", "|)", "|]"};
    leetabc[4] = {"3"};
    leetabc[5] = { "|=", "ph", "|#", "||"};
    leetabc[6] = {"[", "-", "+", "6"};
    leetabc[7] = {"4", "|-", "[-", "{-", "|=", "[=", "{="};
    leetabc[8] = {"1", "|", "!", "9"};
    leetabc[9] = {"_|", "|/", "_7", "|)", "|]", "|)"};
    leetabc[10] = {"|<", "1<", "l<", "|{", "l{"};
    leetabc[11] = {"|_", "|", "1", "][]"};
    leetabc[12] = {"44", "|||/|", "^^", "||||/", "/X||", "|||/][", "||V ←
    []","|[||||/][], "(V)", "//.", .||||", "N||"};
    leetabc[13] = {"||||", "/||/", "/V", "]||||["};
    leetabc[14] = {"0", "()", "[]", "{}", "<>"};
    leetabc[15] = {"|o", "|O", "|>", "|*", "|textdegree{}", "|D", "/o"};
    leetabc[16] = {"O_", "9", "(,)", "0,"};
    leetabc[17] = {"|2", "12", ".-", "|^", "12"};
    leetabc[18] = {"5", "$", "S"};
    leetabc[19] = {"7", "+", "7`", "|'||", "'|`", "~|~", "-|-"};
    leetabc[20] = {"|_", "|||_|", "/_/", "|_|/", "(_)", "[_]", "{_}"};
    leetabc[21] = {"||/"};
```

```
leetabc[22] = {"||/||/", "(/||)", "||^/", "||/|||", "||x/", "||||'", "| ←
'//", "VV"};
leetabc[23] = {"%", "*",
"><", "}{"}, ")("};
leetabc[24] = {"`/", "$yen$"};
leetabc[25] = {"2", "7_",
">_"};

abc =
{
    "A",
    "B",
    "C",
    "D",
    "E",
    "F",
    "G",
    "H",
    "I",
    "J",
    "K",
    "L",
    "M",
    "N",
    "O",
    "P",
    "Q",
    "R",
    "S",
    "T",
    "U",
    "V",
    "W",
    "X",
    "Y",
    "Z"
};

string LeetCipher::decipher(string &text) {
    string leet = "";
    string be = text;

    transform(be.begin(), be.end(), be.begin(), ::toupper);

    while(!be.empty())
    {
        bool find=false;
        for (int i=0; i<abc.size(); i++)
        {
            size_t found = be.find(abc[i]);
```

```
if (found==0)
{
    find =true;
    leet+=leetabc[i][rand() % leetabc[i].size()];
    if(be.length() > 1)
        be=be.substr(1);
    else
        be.clear();
}
}

if(!find) {
    leet+=be[0];
    if(be.length()>1)
        be=be.substr(1);
    else
        be.clear();
}
}

return leet;
}

int main(int argc, char * argv[])
{
    ifstream infile (argv[1]);
    fstream outfile (argv[2], ios::out);

    string text = "";
    string temp = "";

    while(getline(infile, temp)) {
        text+=temp;
    }

    LeetCipher* cipher = new LeetCipher();
    string cipheredText = cipher->decipher(text);
    outfile<<cipheredText<<endl;
}
```

Ebben a feladatban a szleng nyelvvel fogunk foglalkozni. Különböző betűket tudunk használni akár másik számmal vagy másik szimbólikus karakterekkel is. Ilyen például az S betű helyett a \$ vagy E betű helyett a 3. Manapság ezek igen elterjedtek a fiatalok körébe, viszont sokan pedig nem értik ezeket a szleng nyelvet. Maga ez egész feladat érthető és könnyen végigvihető. A kódcsípet első részében megvannak adva az adott betűnek a szlengben használt "jelei". Ezután következik, hogy a nagybetűket kisbetűvé alakítja át a kód és átírjam az első részben vett leet karakterekre. A legvégén pedig egyszerűen csak hivatkozunk rá és meghívuk.

The screenshot shows a terminal window on a Linux system. The terminal output is as follows:

```
martin@martin-VirtualBox:~/Dokumentumok/C++$ g++ leet.cpp -o leet
: error: leet.cpp: Nincs ilyen fájl vagy könyvtár
: fatal error: no input files
compilation terminated.
martin@martin-VirtualBox:~/Dokumentumok$ cd C++
martin@martin-VirtualBox:~/Dokumentumok/C++$ g++ leet.cpp -o leet
martin@martin-VirtualBox:~/Dokumentumok/C++$ ./leet in.txt out.txt
martin@martin-VirtualBox:~/Dokumentumok/C++$
```

At the top right of the terminal window, there is a file viewer showing the contents of 'in.txt'. The file contains the string '@|8{|}3|#'. Below the terminal window, there is a status bar with the text 'Egyszerű szöveg ▾ Tabulátorszéle'.

DRAFT

15. fejezet

Helló, Stroustrup!

15.1. String osztály előkészítése

Ebben a feladatban egy string osztály kell létrehoznunk. Másoló és mozgató szemantika lehetséges még fel itt. Ennek a szemantikának az a lényege, hogy amikor új objektumot hozunk létre és annak értékét adunk akkor a mozgató vagy másoló konstruktőr jelenik meg. Viszont ha egy előkészített objektumnak adunk új értéköt akkor a mozgató vagy másoló értékadás van jelen. Nézzük is a kódunkat.

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;
class mystring
{
    char* data;
public:
    char* Data() const {return data;}
    mystring(const char* p)
    {
        cout<<"konstruktor"<<endl;
        size_t size = strlen(p) + 1;
        data = new char[size];
        memcpy(data, p, size);
    }
    ~mystring()
    {
        delete[] data;
    }
    mystring(const mystring& regi)
    {
        cout<<"másoló konstruktor"<<endl;
        size_t size = strlen(regi.data) + 1;
        data = new char[size];
        memcpy(data, regi.data, size);
    }
}
```

```
mystring& operator=(const mystring& regi)
{
    cout<<"másoló értékkadás"<<endl;
    delete[] data;
    size_t size = strlen(regi.data) + 1;
    data = new char[size];
    memcpy(data, regi.data, size);
    return *this;
}
mystring (mystring&& regi )
{
    cout<<"mozgató konstruktor"<<endl;
    data = regi.data;
    regi.data = nullptr;
}
mystring& operator=(mystring&& regi)
{
    cout<<"mozgató értékkadás"<<endl;
    swap(data, regi.data);
    return *this;
}

char operator[](int i)
{
    size_t size = strlen(data) -1;
    if( i > size )
    {
        cout << "Out of range" <<endl;
    }
    return data [i];
};

ostream& operator<<(ostream& os,const mystring& x)
{
    return os<<x.Data();
}
int main()
{
    mystring a("szoveg helye");
    cout<<a[5]<<endl;
}
```

Itt látható egy operátor túlterhelés. Régebben ezt már kitárgyaltuk így nem írnám le újra. Ahogy utána látszik megnézzük, hogy hol "van távolságon túl" azaz ha nagyobb indexű értéket szeretnénk visszakérni mint amekkora maga a data pointer (*data) akkor sajnos hibát fogunk kapni(Out of range). Viszont ha kisebb akkor egy referenciát dob vissza ami a helyes indexű karakterre mutat. A következő sorokban hasonló szemantikával csak az operátor túlterhelés utána van írva, hogy "const". Ez annyit takar, hogy visszatérési értékként egy másolatot kapunk az i-edik elemről ezáltal nem történik módosítás.

The screenshot shows a code editor with the file `String.cpp` open. The code defines a class `mystring` with various operators and a `main` function. The terminal window to the right shows the output of running the program, which includes a constructor call and the value 9.

```

String.cpp ✘
1  data = regi.data;
2  regi.data = nullptr;
3 }
4 mystring& operator=(mystring&& regi)
5 {
6     cout<<"mozgató értékkedés"<<endl;
7     swap(data, regi.data);
8     return *this;
9 }
10
11 char operator[](int i)
12 {
13     size_t size = strlen(data) -1;
14     if( i > size )
15     {
16         cout << "Out of range" << endl;
17     }
18     return data [i];
19 }
20
21 ostringstream& operator<<(ostream& os, const mystring& x)
22 {
23     return os<<x.Data();
24 }
25 int main()
26 {
27     mystring a("szöveg helye");
28     cout<<a[5]<<endl;
29 }

```

```

/home/martin/Dokumentumok/C++/String - konstruktor
9
Process returned 0 (0x0)   execution time : 0.002 s
Press ENTER to continue.

```

15.2. Hibásan implementált RSA törése

Készítsünk betű gyakoriság alapú törést egy hibásan implementált RSA kódoló: <https://arato.inf.unideb.hu/batfai.n> (71-73 fólia) által készített titkos szövegen.

Először az RSA törés fogalmát kéne tisztázni. Az eljárás nyílt kulcsú, titkosító algoritmus, mely 1976-ban látott napvilágot Ron Rivest, Adi Shamir és Len Adleman jóvoltából. Maga az eljárás során van egy nyílt és egy titkosított kulcs. A nyílt kulcshoz mindenki hozzá tud férfi míg a titkoshoz nevéből adódoan nem és utóbbi szükséges a nyílt kulccsal kódolt üzenet "visszafejtésére". Most magát a matematikai hátterét írnám le. Első lépésben választanunk kell két darab prímszámot(minél nagyobb annál biztonságosabb). Egyik pírmszámot "p"-nek, míg a másikat "q"-nak fogjuk jelölni. Ezt követően egy képlet segítségével kikell számolni N-t, ahol az $N = p \cdot q$, két pírmszámot összeszorozva kapjuk meg a modulusa. (Modulus méretei : 512, 768, 1024, 2048, 4096) $N = p \cdot q$, két pírmszámot összeszorozva kapjuk meg a modulusust. Miután ez megvan kikell számolnunk Euler féle Fí függvény értékét N-re. (Euler féle Fí = Egész számokon értelmezett egész értékű számelméleti függvény) $F(N) = (p-1)(q-1)$, egyik prímszámból elvéve egyet és a másik prímszámból is elvéve egyet majd ezeket összeszorozva kapjuk meg a Euler féle Fí függvény N értékét. Következő lépésben újra választanunk kell egy egész számot amit "e"-vel fogunk jelölni. ("e" = nyílvános kulcs kitejőve, nyílvános) Három kritériumnak kell végbe mennie a kiválasztás során : 1. Nagyobbnak kell lennie "e"-nek mint 1 | 2. "e"-nek kisebbnek kell lennie, mint az előbb kapott Euler féle Fí függvény N értéke. | 3. Az előbb említett N érték és "e"-nek a legnagyobb közös osztójának 1-nek kell lennie. Még egy "d" is kelleni fog amit nem megadni fogunk hanem kikell számítani. ("d" = titkos kulcs kitejőve, titkos) $d \cdot e \equiv 1 \pmod{F(N)}$ Jelenesetben az a lényeg, hogy elosztva a két számot 1-et kapunk maradékna. A nyílvános kulcs N modulusból illetve "e" kitevőből áll. A titkos kulcs N modulusból illetve "d" kitevőből áll. Fontos megjegyezni, hogy a "p" illetve "q" prímszámokat érdemes titokban tartani, mivel ha ezek kitudódnak onnantól könnyű visszavezetni "d"-re, N megvan, "e" nyílvános tehát az is megvan és innen az előző képletbe visszahelyettesítve megkaphatjuk egy gyors számolással a "d"-t azaz a titkos kulcs kitevőjét. A kulcsgenerálásához szükséges idő lehet akár eléggé nagy is(percek), ezért ahol jobban

számít a gyorsaság sajnos nem annyira alkalmazható. Ha egy RSA kódolás megfelelően gondolkodással van megírva (azaz minél nagyobb a két prímszám szorzata), ez esetben nem lehetetlen a feltörés viszont eléggyé sok időben telik. Nincs bizonyítva, hogy nem létezhet kellő gyorsaságú algoritmus erre a probléma megoldásra viszont jelenleg még nem ismert ez. Összefoglalva az RSA algoritmus széles körben terjedt el akár elektronikus kereskedelmi protokollokban akár máshol és igen hasznos illetve biztonságos tud lenni, viszont matematikailag nem bizonyított, hogy feltörhetetlen. Most pedig nézzük meg a kódot.

```
public class RSA{
    public static void main(String[] args)
    {
        int meretBitekben = (int) (700 * (int) (java.lang.Math.log( (double) 10)) / java.lang.Math.log((double) 2));
        System.out.println("Méret bitekben: ");
        System.out.println(meretBitekben);
        java.math.BigInteger p_i = new java.math.BigInteger(meretBitekben, 100, new java.util.Random());
        System.out.println("p_i");
        System.out.println(p_i);
        System.out.println("p_i hexa");
        System.out.println(p_i.toString(16));
        java.math.BigInteger q_i = new java.math.BigInteger(meretBitekben, 100, new java.util.Random());
        System.out.println("q_i");
        System.out.println(q_i);
        java.math.BigInteger m_i = p_i.multiply(q_i);
        System.out.println("m_i");
        System.out.println(m_i);
        java.math.BigInteger z_i = p_i.subtract(java.math.BigInteger.ONE).multiply(q_i.subtract(java.math.BigInteger.ONE));
        System.out.println("z_i");
        System.out.println(z_i);
        java.math.BigInteger d_i;
        do {
            do {
                d_i = new java.math.BigInteger(meretBitekben, new java.util.Random());
            } while(d_i.equals(java.math.BigInteger.ONE));
        } while(!z_i.gcd(d_i).equals(java.math.BigInteger.ONE));
        System.out.println("d_i");
        System.out.println(d_i);
        java.math.BigInteger e_i = d_i.modInverse(z_i);
        System.out.println("e_i");
        System.out.println(e_i);
    }
}
```

Itt jönnek létre a kulcsok amik hibásan vannak implementálva.

```
class KulcsPar {
    java.math.BigInteger d, e, m;
```

```
public KulcsPar() {
    int meretBitekben = 700 * (int) (java.lang.Math.log((double) ←
        10)
        / java.lang.Math.log((double) 2));

    java.math.BigInteger p =
        new java.math.BigInteger(meretBitekben, 100, new java.←
            util.Random());
    java.math.BigInteger q =
        new java.math.BigInteger(meretBitekben, 100, new java.←
            util.Random());

    m = p.multiply(q);

    java.math.BigInteger z = p.subtract(java.math.BigInteger.ONE).←
        multiply(q.subtract(java.math.BigInteger.ONE));

    do {

        do {
            d = new java.math.BigInteger(meretBitekben, new java.←
                util.Random());
        } while (d.equals(java.math.BigInteger.ONE));

        } while (!z.gcd(d).equals(java.math.BigInteger.ONE));

        e = d.modInverse(z);

    }
}
```

"BigInteger" nevéből adódóan ahoz kell, hogy minél nagyobb számokkal tudjunk számolni(ezáltal hatékonyabban is). Itt lászik, hogy a m(modulus) értékét úgy kapjuk meg, hogy a p és a q prímszámokat összeszorozzuk. Megfigyelhető, hogy itt kapjuk még meg a kitevőt illetve annak az inverzét is.

```
public class RSAPelda {
    public static void main(String[] args){

        KulcsPar jszereplo = new KulcsPar();

        String tisztaSzoveg = "Szia, Martin!";

        //kódol
        byte[] buffer = tisztaSzoveg.getBytes();
        java.math.BigInteger[] titkos = new java.math.←
            BigInteger[buffer.length];

        for (int i = 0; i < titkos.length; ++i){
            titkos[i] = new java.math.BigInteger(new byte[] { ←
```

```
        buffer[i]);
titkos[i] = titkos[i].modPow(jSzereplo.e, jSzereplo ←
    .m);
}

//dekódol
for (int i = 0; i < titkos.length; ++i){
    titkos[i] = titkos[i].modPow(jSzereplo.d, jSzereplo ←
        .m);
    buffer[i] = titkos[i].byteValue();
}

System.out.println(new String (buffer));
}

}
```

Itt végződik el a kódolás illetve a dekódolás is. "modPow" függvény abban segít nekünk, hogy megadja a hatványkitevőt és a modulust.

```
martin@martin-VirtualBox:~$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ gedit
^C
martin@martin-VirtualBox:~/Dokumentumok/Java$ javac RSA.java
martin@martin-VirtualBox:~/Dokumentumok/Java$ java RSA
Méret bitekben:
2019
p_i
33753026168010929794108731999251817385382450780457469112917104940129167691445391279896782827085178201166571910392630303318077655651365201193241773782
42468462264488879731309796787452207157431905991338140275109996914910608086475235645144819268478466089281713098687000945280913016527630920259993501787
45997645603417922498965934275956018164789670113720437665938718884610954699087670504915816569318742464493024179015092591508006860611614953165617699150
p_i hexa
47c5e704b0ce6d699baa6ca9356bfab9545180a7d0b9b3a270ddf4e3a67d810616528020ba4a4c5cd83fc591520da5fbe0f7861bbc2f65668a882b6be211dfcb79e0a68ddd0fd6087e2
f12d292b2bfcff7788f5e5e9efcf4cc8b2bfbcc776aaa0efbc8da09eec535eefbad01428b8280276896e2588324596d1ec76534eb4f946f4728ffa754bc38ae09700247140c22232bae3ea
794d42b81a653888b612d6a19281c5e7f5f1e30508d120c51ab522283f766612f054b3186abc830591c11a9209368500d
q_i
4058920617392687583759883881852018794047717661288899753115415053071604070174538599931188304307388092787447120125236732483413819755208350660063038503
0474135208161892582057863245878093045829150108237291219755400947405956342893751351450319672866529383989827500152629263155466761648087858209
53197354236509084258793023258247625039070082308430444124391756089504887836744847338471786419581991518744343671576827055758193630249808993369297168
n_i
137000853812734463113147841938722991801724213399458419546802589245022088958273228171204185998985120689894530517325081972375948577194835284707150622
09836758322478892393078225433387162533213121825560279047165547590359814661106782940366690817062530384766204875094342135408287366436350752337283155018
01117743479805788527883798995184428197348689765222827520279051448288752494964542764337396193953178524014799258309285654034260273505402792771234857012
560473886639275283104382176475742463632415816216139481406966216495639842133884721189922134377853066023823128457618490878987681875056954735661781
00922048998237045651370878626808111708517620476221518669731648097121359687583391663511388731892752053607927250759951464121913167758107962204525748609
92689925371852762001777471241501229259705353024953062303624898890625807182425903904805543677295438431197932266990498663619312821817033723520375299137
z_i
137000853812734463113147841938722991801724213399458419546802589245022088958273228171204185998985120689894530517325081972375948577194835284707150622
09836758322478892393078225433387162533213121825560279047165547590359814661106782940366690817062530384766204875094342135408287366436350752337283155018
01117743479805788527883798995184428197348689765222827520279051448288752494964542764337396193953178524014799258309285654034260273505402792771234857012
337239692858712123473003992262483850089308116955173235587646193194778491459019240767828774844014699303526142181699940381108323203906612796864
0277858702018391197477054589486332808487479208399503844778695894669989930079800270180416362755899270792505531804971286991163367812888139025776722
426915261025293464631323858565707745599598497753198867726044383059003263127238136192198629405087831351270540029631302563674368068660273041412102614
d_i
3015438136631173185600174750001437379343622127432430110556701072858373005523380263765210590854034408710372239136318261278586142415304300179511289330
14342989830697217626184234928526916196429726483672283199215598943391091400925850529795797624738810281250398756297871677994340783254362672178945314413
217550038769355451173331015498132162072612436934735542579451034157849530003443991605216354196644230307006978446746836070085489950287000755660946348
e_i
40890035099099670867231974840542648801013295426070894581157347516831856539730192955972689871117877910748077296060042355767709573917375767496692237953
4233039138836648692358843442783840125556306566353431369525313440563939242653828938497933449637548667930119326269411968647570692680932912945388930884
3421316344413187007415178987739405698012675490156624029697354673811468837589689922411981978643505233043955184448541417887592627448994924643377012072
1164491373732505686674033724993321789542477654759861904287527307610919266803445061570072505506795918704464277836557357836980891558149671207242266
24465684343596124805967953467485632195602392141010011310225708880464573163349401137132991423546664503026263363713904355699224000341971051219132740505
27221132729971906500574445446474604934199710381101332862395926662202989591922527912021833456637328933499269499201787313890315876672623705585648566388
```

```
martin@martin-VirtualBox:~/Dokumentumok/Java$ javac RSApelda.java
martin@martin-VirtualBox:~/Dokumentumok/Java$ java RSApelda
Szia, Martin!
martin@martin-VirtualBox:~/Dokumentumok/Java$
```

15.3. Összefoglaló

Az előző 4 feladat egyikéről írj egy 1 oldalas bemutató „esszé szöveget!

Ezt a feladatot összefűztem az "Hibásan implementált RSA törése" feladattal.

DRAFT

16. fejezet

Helló, Gödel!

16.1. Gengszterek

Gengszterek rendezése lambdával a Robotautó Világajánokságban <https://youtu.be/DL6iQwPx1Yw> (8:05-től)

Magát az OOCWC (Robocar World Championship) működését már egyszer leírtuk, de felelevenítve annyi a lényege, hogy ezen a platformon keresztül kutassák a forgalomirányítás algoritmusokat illetve nevéből adódóan a robotautók terjedését is. Ebben a feladatban lambda segítségével kell rendezni benne. C+11 óta használható `std::sort()` függvényt fogjuk használni ennél a feladatnál. Ennek a függvénynek többféle használata létezik, mint például 2 vagy akár 3 paraméteres típusok. Itt a 3 paraméteres függvényt fogjuk alkalmazni :

```
std::sort ( gangsters.begin(), gangsters.end(), [this, cop] ( ←
    Gangster x, Gangster y )
{
    return dst ( cop, x.to ) < dst ( cop, y.to );
} );
```

`gangsters.begin()`, `gangsters.end()`, ez lesz az első illetve második paraméter amiből leolvasható, hogy mettől-meddig menjen a rendezés. `[this, cop] (Gangster x, Gangster y)`, ez lesz a harmadik paraméter ami a rendezéshez egy függvényt ad meg. Megvizsgálja a "gangsters" tömböt és amennyiben x Gangster közelebb van a cop-hoz, mint y akkor igaz-ként tér vissza a függvény. Amikor végig nézte az egészet eredményül a cop-hoz való legközelebbi gangster-ek lesznek legelől.

16.2. STL map érték szerinti rendezése

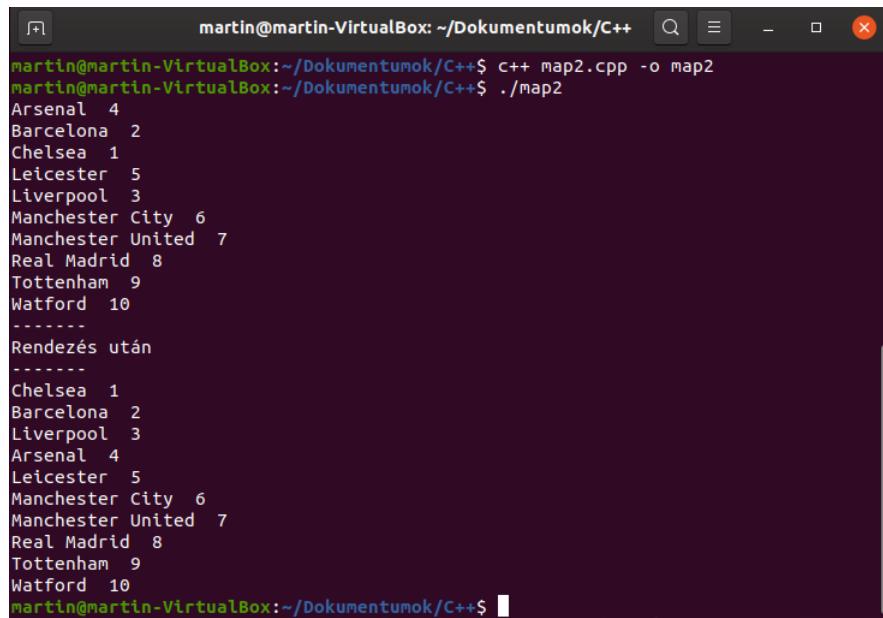
Például: <https://github.com/nbatfai/future/blob/master/cs/F9F2/fenykard.cpp#L180>

Először is az STL fogalmát szeretnémm tisztázni. Standard Template Library a teljes neve és ezenbelül találhatóak a tárolók. Ebben a feladatban a STL tároló map-ot fogjuk használni. Ezek a map tárolók asszociatív tömbök amelyek kulcs-adat párokat tárolnak. Ahogy a feladat kérte a STL mapot érték szerint kell ebben a feladatban rendezni nem pedig kulcs alapján. Most pedig lássuk a kódunkat :

```
#include <iostream>
#include <map>
#include <set>
#include <functional>
using namespace std;
int main() {
    map<string, int> Map;
    Map.insert(pair<string, int>("Chelsea", 1));
    Map.insert(pair<string, int>("Real Madrid", 8));
    Map.insert(pair<string, int>("Barcelona", 2));
    Map.insert(pair<string, int>("Manchester United", 7));
    Map.insert(pair<string, int>("Manchester City", 6));
    Map.insert(pair<string, int>("Liverpool", 3));
    Map.insert(pair<string, int>("Tottenham", 9));
    Map.insert(pair<string, int>("Arsenal", 4));
    Map.insert(pair<string, int>("Leicester", 5));
    Map.insert(pair<string, int>("Watford", 10));
    typedef function<bool(pair<string, int>, pair<string, int>)> Hasonlito;
    Hasonlito hasonlit =
        [](pair<string, int> else ,std::pair<string, int> masodik)
    {
        return else.second < masodik.second;
    };

    set<pair<string, int>, Hasonlito> sortedMap(Map.begin(), Map.end(), ←
        hasonlit);
    for (pair<string, int> elem : Map)
        cout << elem.first << " " << elem.second << endl;
    cout << "-----" << endl;
    cout << "Rendezés után" << endl;
    cout << "-----" << endl;
    for (pair<string, int> elem : sortedMap)
        cout << elem.first << " " << elem.second << endl;
}
```

Pár focicsapat felsorolva random kitalált számokkal, ezeket rendezni ez a kód. Láthatjuk, hogy először egy stringet fog kérni aztán pedig egy int-et ami szerint fogja rendezni a mapunkat. Mint ahogy látszik ez után történik meg az összehasonlítás kér pár között. Miután ez megtörtént akkor fogja rendezni a mapunkat. Itt adhatjuk meg, hogy melyik osztály használja a program a rendezéshez illetve a map kezdetét és végét és a hasonlít metódust is. Mikor megtörtént a rendezés nincs más dolgunk, mint szépen kiíratni.



```
martin@martin-VirtualBox:~/Dokumentumok/C++$ c++ map2.cpp -o map2
martin@martin-VirtualBox:~/Dokumentumok/C++$ ./map2
Arsenal 4
Barcelona 2
Chelsea 1
Leicester 5
Liverpool 3
Manchester City 6
Manchester United 7
Real Madrid 8
Tottenham 9
Watford 10
-----
Rendezés után
-----
Chelsea 1
Barcelona 2
Liverpool 3
Arsenal 4
Leicester 5
Manchester City 6
Manchester United 7
Real Madrid 8
Tottenham 9
Watford 10
martin@martin-VirtualBox:~/Dokumentumok/C++$
```

16.3. Alternatív Tabella rendezése

Mutassuk be a https://progpater.blog.hu/2011/03/11/alternativ_tabella a programban a java.lang Interface Comparable T szerepét!

Ebben a feladatban egy alternatív tábla rendezést fogunk létrehozni az adott NB1-es bajnokságra. Ez a tábla és a rendes bajnoki tabella között az a különbség, hogy míg a bajnoki tabella pontok alapján rangsorolja a csapatokat addíg a az alternatív megnézni az adott csapattal játszott csapatok rangsorát illetve a meccs kimenetelét és ennek függvényében ad vissza egy értéket. Tehát nem mindegy, hogy valaki az utolsó három csapat ellen nyert vagy az első három ellen. Számos fajtája van ezeknek az alternatív táblázatoknak mint például a PageRank.

```
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;
        }
    }
}
```

Elsősorban létrehozunk egy "Csapat" osztályt. Ezután olvasva a kódot látszik, hogy ez implementálja a "Comparable" interfést. Maga az osztály rendelkezni fog névvel illetve értékkel. Még ami kivehető ebből a kódcsipetből a "compareTo" függvény ami az összehasonlításban játszik szerepet. Az összehasonlítás menete igen egyszerű. Ha a csapat az adott objektumhoz képest nagyobb akkor minusz egyet kap, ha kisebb akkor plussz egyet ha pedig ugyanakkora akkor nullát fog kapni.

DRAFT

17. fejezet

Helló, !

17.1. FUTURE tevékenység editor

Javítsunk valamit a ActivityEditor.java JavaFX programon! <https://github.com/nbatfai/future/tree/master/cs/F6>
Itt láthatjuk működésben az alapot: <https://www.twitch.tv/videos/222879467>

Mentorom ennél a feladatnál : Mester Ákos

Ebben a feladatban módosítanunk kell valamit az adott programban. A jelenlegi programban a jobb klick lenyomásával vagy egy új tevékenység jön létre vagy pedig egy props fájl. Mivel ez nem a legmegfelelőbb nekünk ezért erre írt(am) egyfajta megoldást. A CTRL + B kombinációkombóra egy új mappa létrehozását hozunk létre.

```
javafx.scene.Scene scene = FileTree.this.getScene();  
  
KeyCombination kc1 = new KeyCodeCombination(KeyCode.B, KeyCombination.  
    ←  
    CONTROL_DOWN);  
Runnable rn1 = ()-> {  
    System.out.println("Ctrl B");  
    javafx.scene.control.TreeItem<java.io.File> item = FileTree.this.  
        .←  
        getSelectionModel().getSelectedItem();  
    System.out.println(item.getValue());  
    java.io.File file = item.getValue();  
    java.io.File f = new java.io.File(file.getPath() + System.  
        getProperty("file ←  
        .separator") + "Új altevékenység");  
    if (f.mkdir())  
    {  
        javafx.scene.control.TreeItem<java.io.File> newAct  
            = new FileTreeItem(f, new javafx.scene.image.ImageView(←  
                actIcon));  
        item.getChildren().add(newAct);  
    } else {  
        System.err.println("Cannot create " + f.getPath());  
    }  
}
```

```
};  
scene.getAccelerators().put(kc1, rn1);
```

Ezenkívül még egy modósítást fogok létrehozni. CTRL + V kombinációra egy új props-ot fog létrehozni. Ez szinte ugyan úgy kell létrehozni.

```
KeyCombination kc2 = new KeyCodeCombination(KeyCode.V, ←  
    KeyCombination. ←  
    CONTROL_DOWN);  
Runnable rn2 = ()-> {  
    System.out.println("Ctrl V");  
    javafx.scene.control.TreeItem<java.io.File> item = FileTree.this ←  
        . ←  
        getSelectionModel().getSelectedItem();  
    System.out.println(item.getValue());  
    java.io.File file = getItem().getValue();  
    java.io.File f = new java.io.File(file.getPath() + System. ←  
        getProperty("file ←  
            .separator") + "Új altevékenység tulajdonságok");  
    try {  
        f.createNewFile();  
    } catch (java.io.IOException e) {  
        System.err.println(e.getMessage());  
    }  
    javafx.scene.control.TreeItem<java.io.File> newProps  
        = new FileTreeItem(f, new javafx.scene.image.ImageView( ←  
            actpropsIcon));  
    item.getChildren().add(newProps);  
};  
scene.getAccelerators().put(kc2, rn2);
```

Ezenkívül lehet még színeket állítani is a programon vagy egyéb kinézeti tulajdonságokat a style.css segítségével.

17.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocaremulator/blob/master/justine/ro>

Itt a megszokható OOCWC-ben lévő scanf szerepét fogjuk megnézni. Magát a program lényegét már leírtam pár fejezettel hamarabb az ott megtalálható illetve az egész kód is. Itt csak a függvény használatára fogok kitérni. Nézzük is a kódcsípetet ahol látszani fog a használata :

```
while ( std::sscanf ( data+nn, "<OK %d %u %u %u>%n", &idd, &f, &t, &s, &n ) ←  
    == 4 )  
{  
    nn += n;
```

```
    gangsters.push_back ( Gangster {idd, f, t, s} );  
}
```

Magára a függvényre azért van szükségünk, hogy fájleíróból tudunk olvasni. `while` ciklust van az elején ami addig fut le míg sikeres volt beolvasni 4 paramétert. A függvény stringet vár bemenetül. A "data" segít nekünk abban, hogy olvasni tudjuk a beolvasatlan adatokat. A string kezdete "<OK" míg folytatásban egy egész majd 3 előjel nélküli egész. Utolsó paraméterként találhatjuk meg ahol a függvény a beolvasott bajtok számát határozza meg. ("n") Ahogy látszik még a kódjából, hogy az "nn" változóhoz hozzáadjuk az "n"-et ami ugye a beolvasott bajtok száma. Legvégül pedig a "gangsters" tömbhöz adunk egy gangstert a paramétereivel együtt.

17.3. BrainB

Mutassuk be a Qt slot-signal mechanizmust ebben a projektben: <https://github.com/nbatfai/esporttalents-search>

Maga a BrainB nevezetű programmal már találkoztunk régebben. Azt méri, hogy meddig tudunk fókusznal egy adott játékban. Nehezítés képpen több karakter illetve jelenség próbálja ezt megnehezíteni. Maga a "játék" időtartama 10 perc. Most viszont nem az egész kódcsípet kell nekünk hanem csak egy kis részlete, amiben bekell mutatni a Qt slot-signal mechanizmust. Először is a Qt slot-signal mechanizmust szeretném tisztázni. A Qt egyfajta keretrendszer ami cross-platformos. Itt kell megértenünk a slot-signal mechanizmust ami objektumok közötti kommunikációra ad segítséget. Ezzel átláthatóbb és szébb lesz a kódunk. Egy adott történésre egy signal jön létre. A slot pedig egy függvény ami megfelelő signál esetén hívódik meg. Szemantikailag minden meegyezik. Most pedig lássuk a kivett kódcsípetet :

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

Itt láthatjuk még a `connect` függvényt ami az összekapcsolást valósítja meg a SLOT és a SIGNAL között. Itt látszik, hogy két függvényt futtattunk, egyik a : `heroesChanged` ami a lényegében a karakterváltás a signal bekövetkezésekor illetve másik a `endAndStats` ami pedig a futtatási idő lejárata függvény.

18. fejezet

Helló, Schwarzenegger!

18.1. Port scan

Mutassunk rá ebben a port szkennelő forrásban a kivételkezelés szerepére! <https://www.tankonyvtar.hu/hu/tartalom/tanitok-javat/ch01.html#id527287>

Ennek a feladatnak a témája a kivételkezelés lesz. JAVA nyelvben egy kiemelt fontosságot kap. Maga a kivételkezelés egy programozási mechanizmus mely szándékosan vagy nem szándékosan(error esetén), de megállítja a programot. Az objektum orientált nyelvekben a szintaxisa a try, catch játszadozással jelenik meg. try-nak a lényege, hogy oda tesszük meg a vizsgálandó kód-ot, míg a catch-ben vizsgáljuk a kívételt vagy kívételeket. Egy port szkennelő forrást kell megnéznünk. Maga a feladat a való életben is jelen van. Egyik megfelelője amikor rendszergazdák vagyunk ebben az esetben a feladatunk, hogy a hálózaton keresztül ne férjen más hozzá az adatainkhoz. Egy másik példa pedig amikor, pont ezért kapunk pénzt, hogy hibás részeket keressünk egy adott rendszeren. A port szkennelést nem saját gépen való használat esetén nem nagyon ajánlott mivel nem annyira legális. Lássuk is a vizsgált kódrészletet :

```
public class KapuSzkenner {
    public static void main(String[] args) {

        for(int i=0; i<1024; ++i)

            try {

                java.net.Socket socket = new java.net.Socket(args[0], i);

                System.out.println(i + " figyeli");

                socket.close();

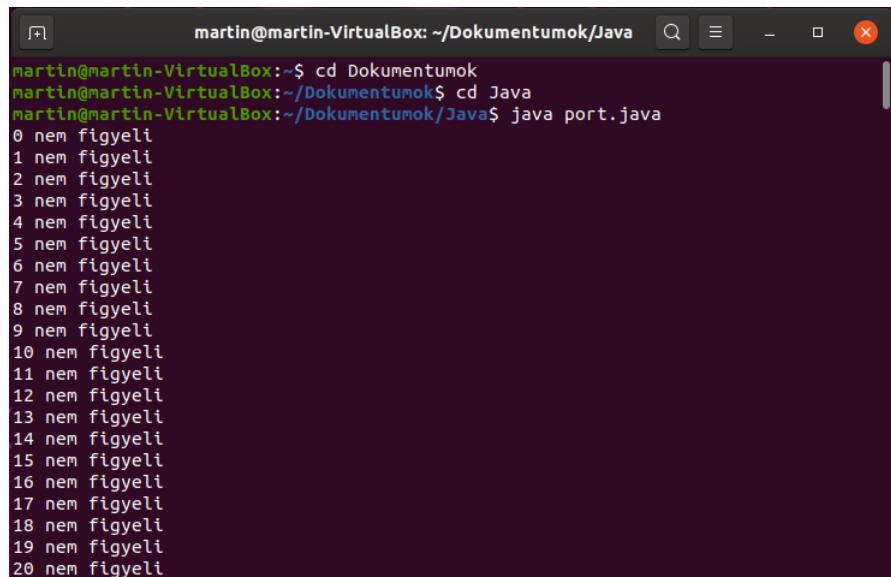
            } catch (Exception e) {

                System.out.println(i + " nem figyeli");

            }
    }
}
```

{

Ahogy látszik csak 1024 portig megy el a program. Ezután ahogyan láthatjuk a `try`, `catch` fog szerepet kapni. minden portra egy `socket`-et próbálunk "ráadni". Ez próbál kapcsolatot létrehozni a porttal. Amennyiben ez sikerült akkor kapcsolat jött létre, viszont ha nem akkor jön a `catch` kivételkezelés.



The screenshot shows a terminal window titled "martin@martin-VirtualBox: ~/Dokumentumok/Java". The command entered was "java port.java". The output of the program is a series of numbers from 0 to 20, each followed by the string "nem figyeli", indicating that no connection was established for any of the ports tried.

```
martin@martin-VirtualBox:~/Dokumentumok$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ java port.java
0 nem figyeli
1 nem figyeli
2 nem figyeli
3 nem figyeli
4 nem figyeli
5 nem figyeli
6 nem figyeli
7 nem figyeli
8 nem figyeli
9 nem figyeli
10 nem figyeli
11 nem figyeli
12 nem figyeli
13 nem figyeli
14 nem figyeli
15 nem figyeli
16 nem figyeli
17 nem figyeli
18 nem figyeli
19 nem figyeli
20 nem figyeli
```

18.2. AOP

Szój bele egy átszövő vonatkozást az első védési programod Java átiratába! (Sztenderd védési feladat volt korábban.)

Ebben a feladatban az AspectJ programozási nyelv fog segítségünkre kelleni. Ebben a nyelvben a már kész JAVA kódokat tudjuk használni vagy akár egységesebbé tenni. Ennek a programozási nyelvnek az a lényege, hogy egy adott kódot úgy módosítsunk, hogy ne kelljen a forráskódon módosítást végrehajtanunk. Az AspectJ segítségével megtudunk változtatni egy függvényt vagy akár hogy előtte / utána változzon-e valami. Ez a programozási nyelv vezette be a csatlakozási pontot (joint point). Ez felel a kereszthívások definilásáért. Még itt érdemes megemlíteni a vágási pontot (cut point). Csatlakozási pontok és az értékük szortirozásáért felelős. A harmadik fogalom amit tisztázni kell az a " tanács ". Itt történik a lényeg amiből következően itt történik a program módosítása. Most pedig lássuk a feladatot:

```
public class Test
{
    public static class testtest
    {
        public void testfn()
        {
            System.out.println("test");
        }
    }

    public static void main(String[] args)
    {
```

```
testtest test1 = new testtest();
test1.testfn();
}
}
```

Ez egy egyszerű kis program ami a kiíratáshoz lesz majd szükséges.

```
public aspect BeforeAfter
{
    public pointcut fnCall(): call(public void testfn());

    before(): fnCall()
    {
        System.out.println("before> test");
    }

    after(): fnCall()
    {
        System.out.println("after> test");
    }
}
```

Itt történik a módosítás az AspectJ segítségével. A vágási pont (pointcut) fogja megmutatni, hogy hol fog végebe menni a szövés.

```
ajc Test.java BeforeAfter.aj
java -cp /usr/share/java/aspectjrt.jar:. Test
```

Itt pedig láthatjuk a fordítást és a futtatást.



```
martin@martin-VirtualBox: ~/Dokumentumok/Java
martin@martin-VirtualBox:~$ cd Dokumentumok
martin@martin-VirtualBox:~/Dokumentumok$ cd Java
martin@martin-VirtualBox:~/Dokumentumok/Java$ ajc Test.java BeforeAfter.aj
martin@martin-VirtualBox:~/Dokumentumok/Java$ java -cp /usr/share/java/aspectjrt
.jar:. Test
before> test
test
after> test
martin@martin-VirtualBox:~/Dokumentumok/Java$
```

18.3. Junit teszt

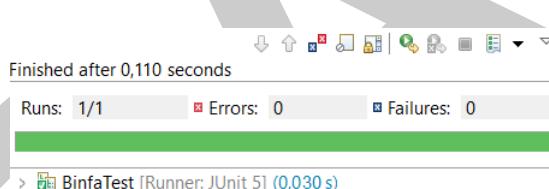
A https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat_poszt_kézzel_számított_mélységét_és_szórását_dolgozd_be_egy_Junit_tesztbe (sztenderd védési feladat volt korábban).

A JUnit-teszt egy keretrendszer JAVA-hoz. Ennek a segítségével tudunk különböző kódokat tesztelő osztályokat. Ellenőrzésképpen tökéletes, hogy jól fut-e le a kód. Az `equals` metódus vizsgálja meg, hogy a két eredmény ugyan az-e. Ebben a feladatban a kézzel kiszámított Binfa átlag,szórás és mélységét értékeit "tesztelés alá" vegyük. Először is kikell számolni ezeket manuálisan. Junit-tools Plugijnt fogok használni ehez a feladathoz. Nem kaptunk error-t, jól futott le a program.

```
package binfa;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
class BinfaTest {
    LZWBinFa binfa = new LZWBinFa();

    @Test
    void test() {
        for (char c: "011110010010010000111".toCharArray()) {
            binfa.egyBitFeldolg(c);
        }

        assertEquals(4, binfa.getMelyseg());
        assertEquals(2.75, binfa.getAtlag());
        assertEquals(0.957427, binfa.getSzoras(), 0.0001);
    }
}
```



19. fejezet

Helló, Calvin!

19.1. MNIST

Az alap feladat megoldása, +saját kézzel rajzolt képet is ismerjen fel, https://progpater.blog.hu/2016/11/13/hello_sbol Háttérként ezt vetítük le: <https://prezi.com/0u8ncvvoabcr/no-programming-programming/>

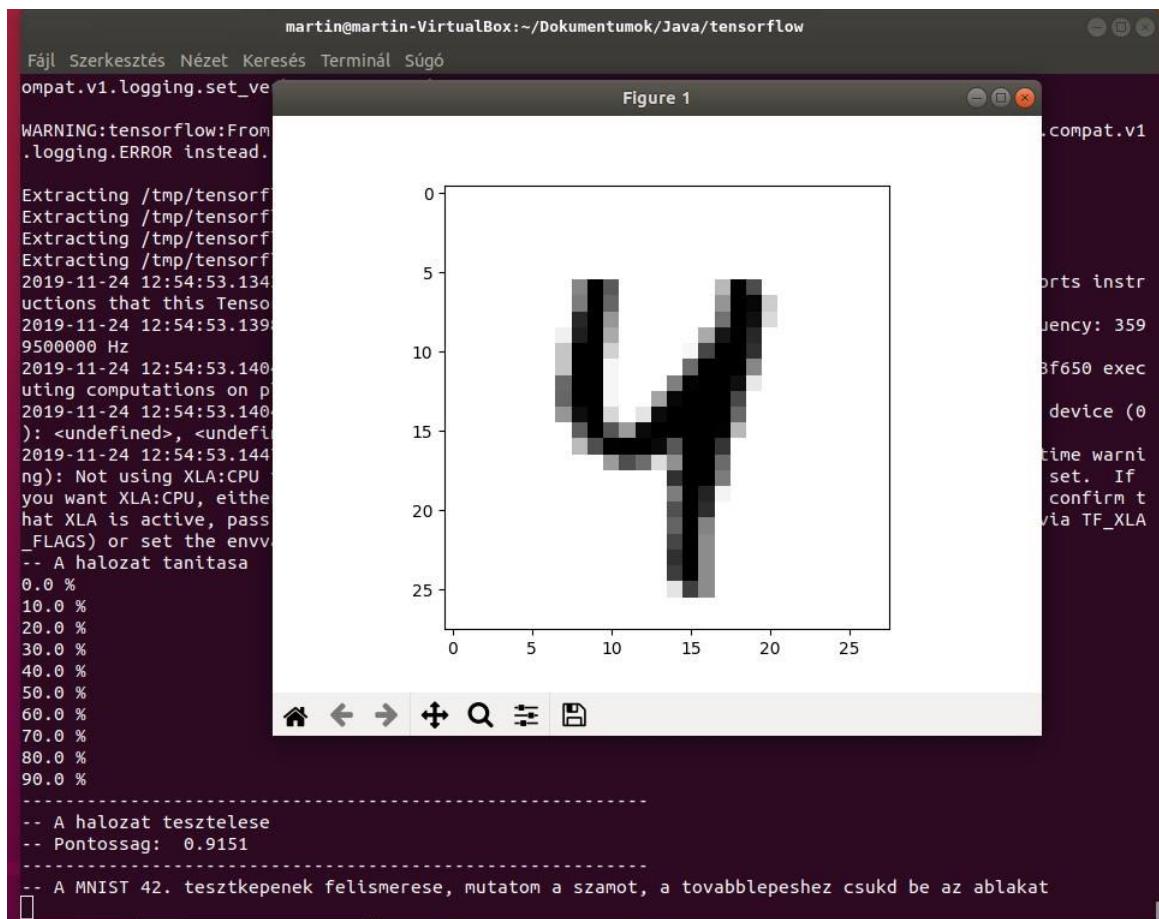
Célunk nem más mint a tensorflow segítségével feléleszteni a programot. Gép tanulást fogunk használni ennél a feladatnál. A Softmax függvényt képletét használja ami valószínűségelosztást ad meg eredményül. $y = \text{softmax}(Wx + b)$ itt látható a képlet. A "W" jelenti a súlyokat. A súlyok döntik el, hogy amit látunk tárgy benne van-e a jó halmazban. A "X" jelenti a bemenő értéket míg a "b" (bias) pedig egyfajta torzítási érték. Az elve hasonlít a neurális and or xor kapura. Ahoz, hogy biztosabb legyen a program minél több hidden réteg szükséges. Jelen esetben 28*28 pixeles képen lévő szám megtanulására tanítjuk a gép modellt. Amikor létrehozzuk a számunkat meg kell adni az "x" illetve "y" értékeket.

```
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("-----")
```

Itt azt figyelhetjük meg, hogy 1000 kép segítségével tanulja meg felismerni. Viszont 100 képenként kapunk tájékoztatást a tanulás folyamatáról.

```
img = readimg()
image = img.eval()
image = image.reshape(28*28)
```

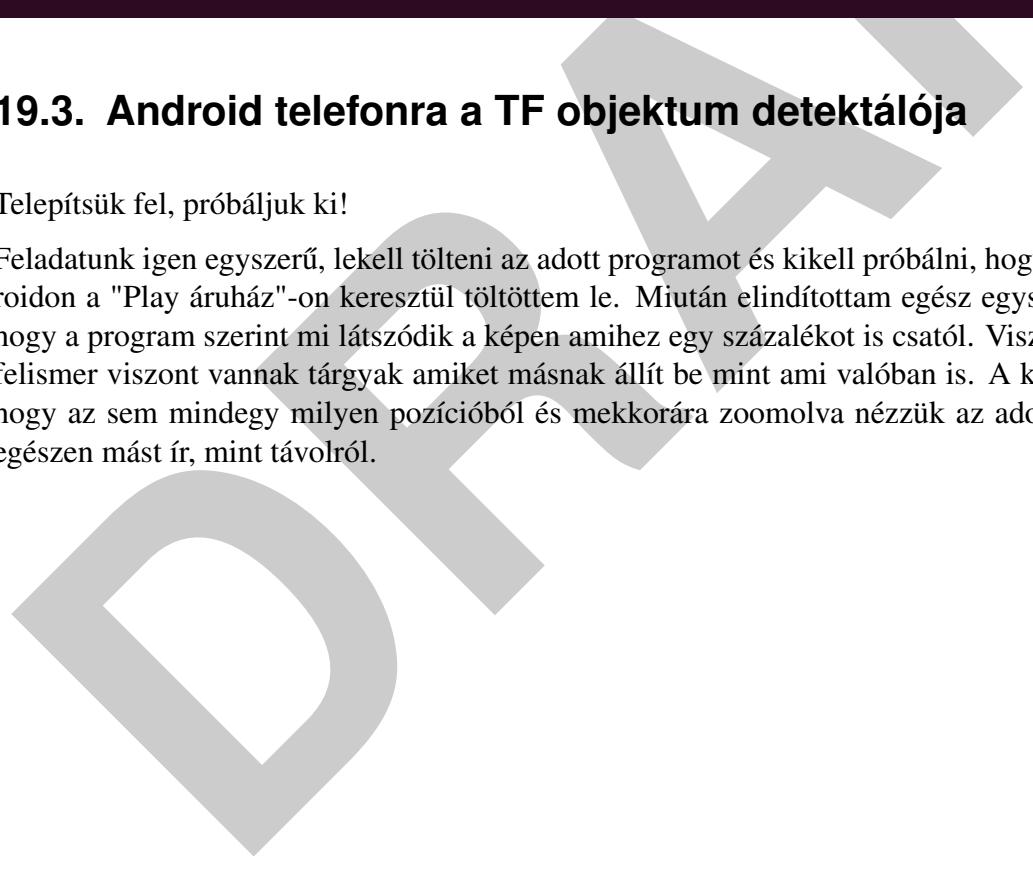
Itt kapcsolódik be a feladat legfőbb része, hogy saját képet nézzen. Később a Matlab segítségével tudjuk kirajzolni a kapott képet.



19.2. CIFAR-10

Az alap feladat megoldása, +saját fotót is ismerjen fel, https://progpater.blog.hu/2016/12/10/hello_samu_a_cifar-10_tf_tutorial_peldabol

A CIFAR-10 dataset egy képgyűjtemény, amit gépi tanuló algoritmusok tanítására szoktak használni. 60000 32x32-es képet tartalmaz, amiket 10 különböző csoportba sorolhatunk, úgy mint: autók, repülők, macskák, kutyák, madarak, szarvasok, teherautók, békák, lovak és hajók. A kicsi felbontás miatt gyorsan tesztelehetik a kutatók, hogy melyik algoritmussal milyen hatásos a tanulás. A példában a képeket binárisá alakítjuk, úgy hogy a képet „ellapítjuk” (lsc.: cifar10_2bin.py), azaz egy egysoros mátrixot képezünk ami magába foglalja a 32 sort, mindenzt 3x, mert színes képeket használunk. A rendszer tanítása (viszonylag) hosszú időt vesz igénybe. A GPU-s futtatás nagyobb teljesítményt eredményezett volna, már a 2 GPU-s futtatás is lehetséges, de sajnos nekem nem állt rendelkezésre megfelelő GPU. A tanítási folyamat befejeztével alkalmaztam a belinkelt tutorial módosításait, érdekes tapasztalat, hogy a pythonban a behúzásoknak mekkora jelentősége van. A tutorial módosításán kívül szükség volt még a logit tömb argumentumként átadására itt. Ha saját képen szeretnénk tesztelni, akkor a képet 32*32-es méretben futtatási argumentumként a cifar10_2bin.py-nek átadjuk.



```
martin@martin-VirtualBox:~/Dokumentumok/cifar
Fájl Szerkesztés Nézet Keresés Terminál Súgó
(cifar) martin@martin-VirtualBox:~/Dokumentumok/cifar$ python3 cifar10_bin.py auto2.png input.bin
(cifar) martin@martin-VirtualBox:~/Dokumentumok/cifar$ python3 cifar10_eval.py --run Once True 2>/dev/null
[-0.25024414  2.2070312 -0.36254883  0.66503906 -0.88720703 -0.31860352
 0.21679688 -0.52783203  0.23950195  0.19897461]
automobile
(cifar) martin@martin-VirtualBox:~/Dokumentumok/cifar
```

19.3. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Feladatunk igen egyszerű, lekell tölteni az adott programot és kikell próbálni, hogy hogyan működik. Androidon a "Play áruház"-on keresztül töltöttem le. Miután elindítottam egész egyszerű a használata, kiírja, hogy a program szerint mi látszódik a képen amihez egy százalékot is csatol. Viszonylag elég sok minden felismer viszont vannak tárgyak amiket másnak állít be mint ami valóban is. A kipróbálás során rájöttem, hogy az sem mindegy milyen pozícióból és mekkorára zoomolva nézzük az adott tárgya, mivel közelről egészen mást ír, mint távolról.





DRAFT

20. fejezet

Helló, Berners-Lee!

20.1. C++ és Java összehasonlítás

C++: Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven

Java: Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II

Két darab könyvet fogok most összehasonlítani. Egyik a „Benedek Zoltán, Levendovszky Tihamér Szoftverfejlesztés C++ nyelven”, míg a másik „Nyékyné Dr. Gaizler Judit et al. Java 2 útikalauz programozóknak 5.0 I-II”. A Java egy objektumorientált eszközökészletből építkező programozási nyelv amely szabványokat, eszközöket foglal egy rendszerbe. A Java előtti objektumorientált programozási nyelv viszont a C+. Míg magát a Javát széles körben web-alapú, vállalati illetve mobilalkalmazásoknál használják addig a C++ nyelvet „rendszer programozás”-nál általában. Magában a C++ nyelvben van operátor túlterhelés viszont a Javánál ez nem található meg. A pointereknél már kicsit más a helyzet. C++ nyelvben egyértelműen támogatva vannak a pointerek amit akár bele is írhatsz egy adott programban. Javánál nem ilyen egyszerű ez, korlátozott pointer támogatással lehet csak beleírni. Maga a C++ nyelv csak fordítót használ, átkonvertálja a forráskódot gépi kóddá, ezért van az, hogy ez a nyelv ahogy említettem is platformfüggő. A Javánál ez bonyolultabb egy fokkal. Először is átkonvertálja a forráskódot bájtkódra. Ezt követően a „tolmács” futtatja ezt a bájtkódot majd kiad egy kimenetet. Ugyan úgy ahogy említettem ezért platformfüggetlen ez a programozási nyelv. A C++ nyelv támogatja a hívásértéket illetve a hívás referenciát ellenben a JAVA csak a hívásértéket támogatja. C++ ahogy megszokhattuk támogatja a struktúrákat míg a JAVA egyáltalán nem. Viszont a „menettámogatásban” megfordul a kocka. C++-nál nincs beépített segítség erre, egy könyvtárra támaszkodva tudjuk csak ezt véghez vinni. JAVA-ban viszont támogatva van és nem kell külön könyvtár rá. A C++ támogatja a „virtuális kulcsszó” használatát ameddig az összehasolítandó JAVA nem teszi ezt meg. C++ nyelven az osztályok adattagjai előtt megadhatjuk a static kulcsszót, jelezve azt, hogy ezeket a tagokat megosztva használják az osztály objektumai. Java nyelv nem „áll” olyan közel a Hardwarerel, mint a C++. Java nem támogatja az alapértelmezett argumentumokat míg a C++ nyelv igen. Javában nem lellhetők fel a header fájlok, viszont a C++-ban igen. Előzőnél a kulcsszavak különféle osztályokat és módszereket tartalmaz. A JAVA nem támogatja a destrukturákat míg a C++ abban is „otthon van”. A Java sokkal hasznosabb memória-használat szempontjából mivel ha megpróbálok értéket rendelni az adott tömbön kívül akkor azonnal hibát kap vissza a programozó. C++-nál ez nem így van, ő megengedi az érték hozzárendelést viszont ezálltal több memóriát használva, de ez később futási időnél visszacsaphat bugokkal illetve összeomlásokkal. Maga a Java lassabnak minősül a C++ nyelvnél. Mivel a C++ ahogy említettem bináris fájlokba át kódolja a forráskódot így az szinte azonnal lefut míg a JAVA fordít és „tolmácsol” is. Maga a szintaxis minden kettő nyelvben elégé hasonló, operátorok (operátor túlterhelés nem),

osztályok, meghatározó változok között elég sok hasonlóságot lehet felfedezni a két programnyelv között. Míg a C++ nyelvben a beviteli mechanizmus „in » SZÖVEG” és a kimenet „cout« SZÖVEG” addig a JAVA-nál a bemenet „System.in” és a kimenet pedig „System.out.println(SZÖVEG)”. Maga az egész C++ nyelv a C nyelvre épül. Sok olyan C-s program van ami C++ nyelven is lefut viszont akad jótár is ami már nem vagy esetleg máshogyan. A JAVA-nál ilyen nincs. Nincs előzetes nyelv viszont ahogy említettem a szintaxisa nagyban felfedezhető a C illetve C++ nyelvekből. Maga a felülete a C++ nyelvnél engedi a közvetlen system könyvtárak hívását ameddig a JAVA csak a JAVA.native felületén keresztül hívhat. A „többszörös öröklés” fellelhető a C++ nyelvnél. Ha esetleg még hiba lépne fel ez során akkor azért vannak a kulcsszavak(pl.:Többszörös öröklésnél). Javánál nincs ilyen. A kivételkezelés Java-ban eltérő mivel nincsenek destruktorkok. Meg kell határozni a try / catch funkciót ha a funkció deklarálja, hogy kivételt okozhat. C++ nyelvnél nincs ilyen még ha a funkció előre fel is veti a hibát. A könyvtáraiknál is eltérés van mivel a C++-nál legtöbbször alacson szintű a „funkcionalitás”. Ezzel szemben a JAVA-nál maga szintűek a szolgáltatások. JAVÁ-nál az összes funkció és adat osztályon belül létezik, míg a C++-nál osztályon kívül is. Maga a JAVA nyelvet leginkább android applikációknál használják. C++ nyelv legtöbbször „rendszer programozás”-nál fordulhat elő.

20.2. Python

Python: Forstner Bertalan, Ekler Péter, Kelényi Imre: Bevezetés a mobilprogramozásba. Gyors prototípusfejlesztés Python és Java nyelven (35-51 oldal), a kijelölt oldalakból élmény-olvasónapló

Maga a Python nyelv alapjait Guido Van Rossum tette le még 1990-ben. Ez egy magas szintű, dinamikus, objektumorientált és a JAVÁ-hoz hasonlóan platformfüggetlen azaz elérhető akár Windows, Unix, MacOS X vagy akár iPhone-nal is. Magát a fordítás fázist kilehet hagyni elég ha csak a forrást adjuk meg. Maga a nyelv az egy köztes nyelv azaz nincs szükség se fordításra se pedig linkelésre. A Python kódcsipetjei sokkal rövidebbek és tömörebbek mint például a JAVA vagy C++ nyelvnél. Ennek több oka is van például a magas szintű adattípusok lehetővé teszik, hogy összetett kifejezéseket írunk le egy rövid állításban vagy a kódcsoporthoz egyszerű tagolással törnénik nincs szükség nyitó és záró jelzésekre. Egyik legfőbb jellemzője a Pythonnak, hogy behúzásalapú a szintaxis. Egy adott blokk végét egy kisebb behúzású sor jelzi, ezért lehet üres sor a blokkon belül. A másik két nyelvttől eltérően egy utasítás a sor végéig tart(nem kell „;” jel), ha több sorba akarjuk ezt akkor a végén kell használnunk egy „\” jelet. A token különböző fajtái : azonosító, kulcsszó, operátor, delimiter, literál. A kis illetve nagybetűket Pythonba meg kell különböztetnünk. Jótár kulcsszó van jelen itt mint például : and, if, elif, else, in, break, try, or, while vagy még sorolhatnám. Megjegyzésekkel itt is lehetünk a „#” jel segítségével. Adattípusok lehetnek : számok, sztringek, ennesek(tuple), listák, szótárak. A nem létező változóra való hivatkozás futás közbeni kivételt okoz. A „print” segítségével írathatunk ki tetszőleges sztringet vagy más változót a konzolra. A nyelv támogatja az „if” (if, elif, else) elágazást is. Ciklusokra térté a jól megismert „for”, „while”, „range” is támogatja. Használatuk a C-hez hasonló. Pythonban a függvényeket a „def” kulcsszóval definiáljuk. Ahogy említettem ez a nyelv támogatja a klasszikus, objektumorientált fejlesztési eljárásokat is. Definiálhatunk osztályokat is. Osztályok lehetnek : objektumok illetve függvények. Egyes osztályok örökölhetnek más osztályokból is. A nyelv a fejlesztés megkönnyítése érdekében sok szabványos modult tartalmaz. Sok modul van mint például : felhasználói felület kezelés, hálózatkezelés, kamera kezelés. Ezenfelül a kivételeket is támogatja. Összefoglalva a Python kód részlete kevesebb mint a JAVA vagy akár a C++-nál használt kódokkal viszont tömörsegíleg ugyan ott van. Moduljai miatt nagyobb problémákat is meglehet oldani, nem hiába nevezik magas szintű programozásnak ezt a nyelvet.

IV. rész

Irodalomjegyzék

DRAFT

20.3. Általános

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

20.4. C

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

20.5. C++

[BMECPP] Benedek, Zoltán & Levendovszky, Tíhamér, *Szoftverfejlesztés C++ nyelven*, Bp., Szak Kiadó, 2013.

20.6. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPORG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésekért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPORG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségen született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.