

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. május 8, v. 0.0.6

Copyright © 2019 Dr. Bátfai Norbert, Tóth Attila

Copyright (C) 2019, Norbert Bátfai Ph.D., batfai.norbert@inf.unideb.hu, nbatfai@gmail.com, Attila Tóth, atoth1571@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>

DRAFT

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY	Bátfai, Norbert ÁCs Tóth, Attila	2019. szeptember 16.

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-05-02	Feladatok kész.Utómunkállatok szükségesek még,bővítés és ellenőrzés.	T.Attilla
0.0.6	2019-05-08	Ábrák felcímzése, ábrajegyzék kész.	T.Attilla

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?	7
2.3. Változók értékének felcserélése	9
2.4. Labdapattogás	11
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	13
2.6. Helló, Google!	14
2.7. 100 éves a Brun tétele	17
2.8. A Monty Hall probléma	18
3. Helló, Chomsky!	20
3.1. Decimálisból unárisba átváltó Turing gép	20
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	21
3.3. Hivatalos nyelv	23
3.4. Saját lexikális elemző	24
3.5. l33t.l	26
3.6. A források olvasása	29
3.7. Logikus	33
3.8. Deklaráció	34

4. Helló, Caesar!	38
4.1. double ** háromszögmátrix	38
4.2. C EXOR titkosító	39
4.3. Java EXOR titkosító	40
4.4. C EXOR törő	41
4.5. Neurális OR, AND és EXOR kapu	41
4.6. Hiba-visszaterjesztéses perceptron	43
5. Helló, Mandelbrot!	44
5.1. A Mandelbrot halmaz	44
5.2. A Mandelbrot halmaz a std::complex osztállyal	45
5.3. Biomorfok	46
5.4. A Mandelbrot halmaz CUDA megvalósítása	47
5.5. Mandelbrot nagyító és utazó C++ nyelven	51
5.6. Mandelbrot nagyító és utazó Java nyelven	52
6. Helló, Welch!	54
6.1. Első osztályom	54
6.2. LZW	56
6.3. Fabejárás	60
6.4. Tag a gyökér	62
6.5. Mutató a gyökér	71
6.6. Mozgató szemantika	79
7. Helló, Conway!	80
7.1. Hangyaszimulációk	80
7.2. Java életjáték	81
7.3. Qt C++ életjáték	83
7.4. BrainB Benchmark	83
8. Helló, Schwarzenegger!	85
8.1. Szoftmax Py MNIST	85
8.2. Mély MNIST	86
8.3. Minecraft-MALMÖ	87

9. Helló, Chaitin!	90
9.1. Iteratív és rekurzív faktoriális Lisp-ben	90
9.2. Gimp Scheme Script-fu: króm effekt	90
9.3. Gimp Scheme Script-fu: név mandala	92
10. Helló, Gutenberg!	94
10.1. Programozási alapfogalmak	94
10.2. Programozás bevezetés	96
10.3. Programozás	97
III. Második felvonás	101
11. Helló, Berners Lee!	103
11.1. C++ és Java	103
11.2. Python	103
12. Helló, Arroway!	104
12.1. OO szemlélet	104
12.2. Homokozó	105
12.3. "Gagyí"	107
12.4. Yoda	109
12.5. Kódolás from scratch	109
13. Helló, Liskov!	110
13.1. Liskov helyettesítés sértése	110
13.2. Szülő-gyerek	110
13.3. Anti OO	110
13.4. Hello Android!	111
13.5. Ciklomatikus komplexitás	111
14. Helló, Mandelbrot!	112
14.1. Reverse engineering UML osztálydiagram	112
14.2. Forward engineering UML osztálydiagram	112
14.3. Egy esettan	112
14.4. BPMN	113
14.5. TeX UML	113

15. Helló, Chomsky!	114
15.1. Encoding	114
15.2. OOCWC Lexer	114
15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció	114
15.4. Paszigráfia Rapszódia LuaLaTeX vizualizáció	115
15.5. Perceptron osztály	115
16. Helló, Stroustrup!	116
16.1. JDK osztályok	116
16.2. Másoló-mozgató szemantika	116
16.3. Hibásan implementált RSA törése	116
16.4. Változó argumentumszámú ctor	117
16.5. Összefoglaló	117
17. Helló, Gödel!	118
17.1. Gengszterek	118
17.2. C++11 Custom Allocator	118
17.3. STL map érték szerinti rendezése	118
17.4. Alternatív tabella rendezése	119
17.5. Prolog családfa	119
17.6. GIMP Scheme hack	119
18. Helló, !	120
18.1. FUTURE tevékenység editor	120
18.2. OOCWC Boost ASIO hálózatkezelése	120
18.3. SamuCam	120
18.4. BrainB	121
18.5. OSM térképre rajzolása 6	121
19. Helló, Schwarzenegger!	122
19.1. Port scan	122
19.2. AOP	122
19.3. Junit teszt	122

20. Helló, Calvin!	123
20.1. MNIST	123
20.2. Deep MNIST	123
20.3. Android telefonra a TF objektum detektálója	123
20.4. SMNIST for Machines	124
20.5. Minecraft MALMO-s példa	124
21. Helló, Berners Lee!	125
21.1. C++ és Java	125
21.2. Python	125
IV. Irodalomjegyzék	126
21.3. Általános	127
21.4. C	127
21.5. C++	127
21.6. Lisp	127

Ábrák jegyzéke

2.1. Végtelen ciklus 100%-os használattal	6
2.2. Végtelen ciklus az összes mag 100%-os használattal	6
2.3. Végtelen ciklus 0%-os használattal	7
2.4. Változócsere	10
2.5. Labdapattogtatás	12
2.6. Szóhossz	13
2.7. Szóhossz jó megoldás	14
2.8. Pagerank hibásmegoldás	16
2.9. Pagerank helyes megoldás	16
2.10. A konstans közelítése	18
2.11. Monty Hall teszt	19
3.1. Turing gép	21
3.2. Hibaüzenet	24
3.3. Lexikális elemző	26
3.4. 1337 5P34CH	29
3.5. Deklaráció	36
4.1. Memória példa	38
4.2. Alsó háromszög mátrix	39
4.3. Exortörés	41
4.4. Signum függvény	42
4.5. Perceptron	43
5.1. Mandelbrothalmaz	45
5.2. Mandelbrot halmaz komplex osztállyal	46
5.3. Biomorf	47

5.4. Mandel CUDA	51
5.5. Mandelbrot nagyító	52
5.6. Java Mandelbrot nagyító	53
6.1. Polargen random	56
6.2. Inorder bejárás	61
6.3. Preorder bejárás	61
6.4. Postorder bejárás	62
7.1. Hangszimuláció	81
7.2. UML osztálydiagramm	81
7.3. Sejtautomata	82
7.4. Életjáték	83
7.5. BrainB benchmark	84
8.1. Softmax mnist	86
9.1. Nemesis króm effektel	91
9.2. Nemesisborder króm effektel	92
9.3. Név mandala	93
12.1. Polargen random	105
12.2. Binfa Servlet indexpage	106
12.3. Binfa Servlet after run	107
12.4. Végtelen ciklus 128-nál	108
12.5. Leállás 127-nél	109

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 gyerekeknek, mert velük készítem az iniciális változatot. Ám tervezem felhasználását az egyetemi programozás oktatásban is. Ahogy szélesedni tudna a felhasználók köre, akkor lehetne kiadása különböző korosztályú gyerekeknek, családoknak, szakköröknek, programozás kurzusoknak, felnőtt és továbbképzési műhelyeknek és sorolhatnánk...

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogy lássuk másit 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
'banax-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 csemegéznek a C nyelvi szabvány [ISO/IEC 9899:2017](#) kódcsipeteiből is.

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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!

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/vegtelen>

Végtelen ciklust a legkönnyebben 3 féle képpen tudunk írni(ezek a standard formák):

```
#include <stdio.h>
int main() {
    while(1); //végtelen ciklus while-al
    for(;); //végtelen ciklus for-al
    do{
        while(1); //végtelen ciklus do while segítségével
    return 0;
}
```

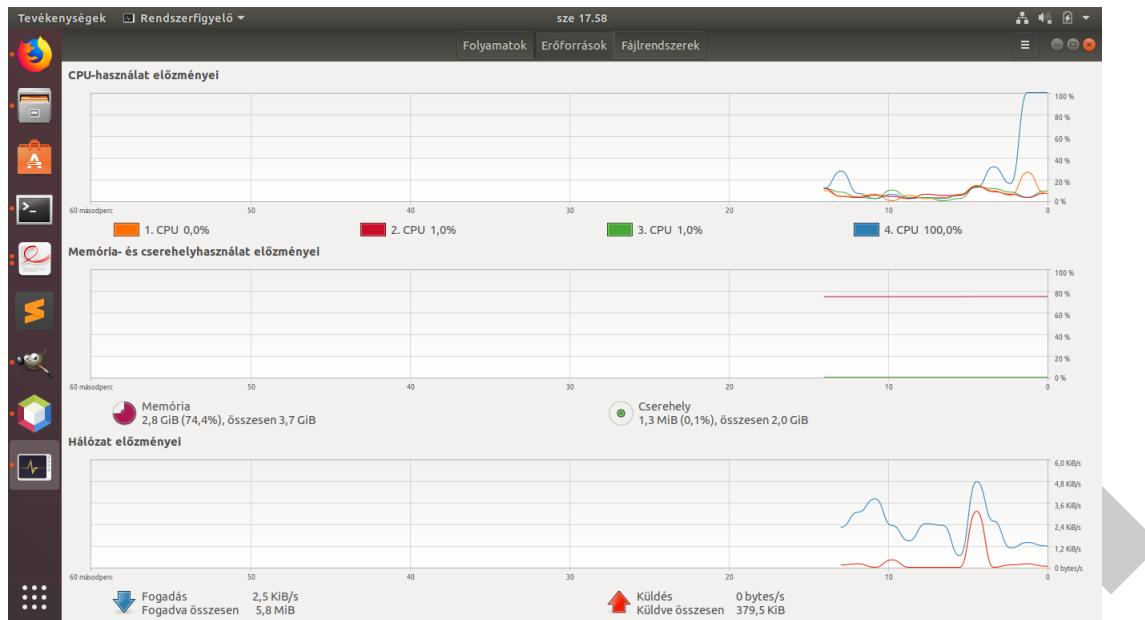
A végtelen ciklus következhet hibából, de van úgy, hogy szándékosan használunk végtelen ciklusokat például a programok menüjénél, de a program futása is egy végtelen ciklus, amelyet az X gombra kattintás szakít meg. Ha simán írunk egy végtelen ciklust az egy szálat fog kihasználni 100%-on, mindaddig amíg nem párhuzamosítjuk, ezt a:

```
#pragma omp parallel
```

segítségével érjük el. Ekkor a program már minden szálat képes kihasználni 100%-on. Fordítani pedig a következőképpen tudjuk:

```
gcc vegtelen3.c -o vegtelen3 -fopenmp
```

Végtelen ciklus 1 mag 100%-os használattal:



2.1. ábra. Végtelen ciklus 100%-os használattal

Végtelen ciklus az összes mag kihasználásával:



2.2. ábra. Végtelen ciklus az összes mag 100%-os használattal

Ha azt akarjuk, hogy 0%-ot használjon a processzorból akkor azt a:

```
sleep();
```

használatával tudjuk elérni, amely lehetővé teszi, hogy a meghívott szálat egy meghatározott ideig "sleepeltesse". Az időt a () között adhatjuk meg másodpercben, ha 0-t adunk meg, akkor végétlen időre sleepeltethetünk. A sleep függvényt az:

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

könyvtár tartalmazza. Tehát a használatához meg kell hívnunk.



2.3. ábra. Végtelen ciklus 0%-os használattal

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

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

Megoldás videó:

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

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

```
        return false;
    }

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

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

{}

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

Ha a T100 as függvény létezne és megkapná a P-t paraméternek akkor igazat ad vissza. De ha a T100 asnak a T100 ast adjuk meg tehát rekurzívan hívjuk meg a T100 ast akkor azt írná ki, hogy a T100 asban nincs végtelen ciklus, pedig a bemenő argumentuma egy végtelen ciklus. Ha létezne ilyen program nem lenne szükség a teszterekre. Mellesleg ez ellentmondást ad vissza.

Tanulság nem lehet jelenleg olyan programot írni, amely normálisan eldönti egy másik programról, hogy kifog -e fagyni avagy sem.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: <https://github.com/Savitar97/Prog1/blob/master/valtcser/valtcser.c>

Két változó értékének felcserélése többféle módon is történhet a legalapvetőbb a segédváltozó használata. Ekkor a 2 változóhoz behozunk egy ideiglenes segédváltozót, amiben valamelyik változó értékét letároljuk, majd az első változó értékét egyenlővé tesszük a másodikkal, majd a második értékét egyenlővé tesszük az ideiglenesben letárolt első változó értékével. Ez itt látható:

```
#include <stdio.h>
int main() {
    int elseo=5,masodik=3,temp;
    temp=elseo;
    elseo=masodik;
    masodik=temp;
}
```

De ezen a módszeren kívül, lehetséges összeadás-kivonással,szorzás-osztással,vagy logikai kizárá vagy művelet segítségével felcserélni két változó értékét.

Legyen két változónk a és b. Összeadással az a-ba összeadjuk a-t és b-t. Majd az a-ból kivonjuk a b-t és ezt letároljuk b-be.Ekkor b értéke egyenlő lesz az a változó kezdeti értékével,majd az a-ból kivonjuk a b változót ekkor a értéke egyenlő lesz b kezdeti értékével, tehát felcserélődtek az értékek.Szorzás-osztással, ugyan így működik.

Két változó értékét logikai operátorral a kizárá vaggyal is megcserélhetjük.

```
#include <stdio.h>
int main(){
    int elso=5,masodik=3;
    elso=elso^masodik;
    masodik=elso^masodik;
    elso=elso^masodik;
}
```

A kizárvagy(xor) lényege, hogy csak akkor igaz ha az egyik igaz.Ez binárisan azt jelenti,hogy akkor 1 es ha ugyan azon a biten lévő érték az egyik változónál 1 es a másikban 0 ás.Az elso változó binárisan 0101 a masodik 0011 kizáró vagyot végre hajtva az elso értéke 0110 lesz, aminek az értéke 6. Majd újra kizáro vagyot végre hajtva a masodik értéke 0101 lesz, ami 5 tehát megkapta az elso ertékét. Ezután még egyszer kizárvagy-ot használunk ekkor az elso értéke 0011 lesz, ami 10 es számrendszerbe 3. Tehát a két változó értéke felcserélődött.

The screenshot shows a Sublime Text interface with two open files: `bhax_mandala9.scm` and `valtcser.c`. The terminal window displays the following session:

```
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/bevprog$ ./valtcser
Elso változó:5
Második változó:3
Elso változó:3
Második változó:5
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/bevprog$
```

2.4. ábra. Változócsere

2.4. Labdapattogás

**Tutor**

Ebben a feladatban tutoráltam Ádám Petrát.

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:<https://github.com/Savitar97/Prog1/tree/master/labda>

Labdapattogtatás if-el: c-ben megadtam egy maximális méretet a pályának ez az x és y változó. A labda kezdetleges koordinátáit a labdax és labday-ban tárolom. Ezen kívül kell még 2 változó, amely a labda mozgásáért felelős ez a tempx és tempy. Magát a labdát karakterként a labda változóban tárolom.

```
#include <stdio.h>
int main(){
    char labda='o';
    int x=80,y=15,labdax=1,labday=1,tempx=1,tempy=1;
```

A labdapattogtatást a for(;;) végtelen ciklus és egy rajzol eljárás folytonos meghívása szolgálja. A labda mozgását a koordináták temp-el való növelése szolgálja. Az if-ek segítségével érem el, hogy ha a labda eléri a pálya szélét, akkor a temp előjele változzon, így az érték csökkeni kezd, majd csökkenés után ha újra eléri a pálya szélét a -1-szeres szorzással újra pozitívba vált. A késleltetett kiírást az usleep éri el, az értéket microsec-be kell megadni és az unistd.h könyvtár tartalmazza ezt a függvényt.

```
#include <stdio.h>
#include <unistd.h>
int main(){
    for(;;)
    {
        labdax+=tempx;
        labday+=tempy;
        if(x-1<=labdax)
        {
            tempx*=-1;
        }
        else if(y-1<=labday)
        {
            tempy*=-1;
        }
        else if(labdax<0)
```

```
{  
    tempx*=-1;  
}  
else if(labday<0)  
{  
    tempy*=-1;  
}  
rajzol(labdax,labday,labda);  
usleep(100000);  
}  
}
```

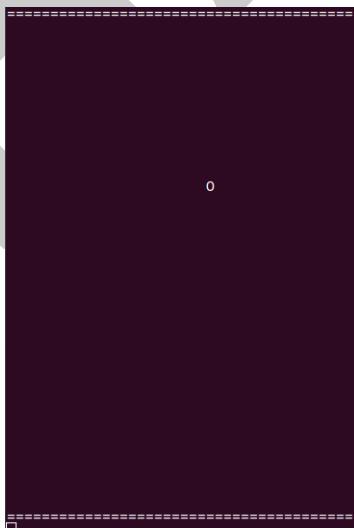
If nélkül azt, hogy a labda vissza pattanjon a maradékos osztás végzi el és az abszolút érték.

```
x=abs(szelesseg-lepteto%(2*szelesseg));  
y=abs(tmagassag-lepteto%(2*tmagassag));  
lepteto++;
```

És persze szükséges mellette egy változó aminek az értékét folyamatosan növeljük és a pálya méretének 2x esével osztjuk el maradékosan, majd kivonjuk a pálya méretéből és ez határozza meg a labda koordinátáját. Tehát mondjuk egy 50 es pálya méretnél $50-1\%100=49$... így csökken egészen 0-ig majd mikor a léptető eléri az 51 et $50-51=-1$ el, de ennek az abszolút értéke 1, tehát újra növekedni fog.

Tapasztalat:C-ben van egy kis hiba mivel, amikor eléri a pálya tetejét a labda nem egyből pattan vissza, ez csak if-nél jelentkezik. If nélkül a két abszolút értékes függvénytel nem jelentkezik ez a hiba.

A program futás közben:



2.5. ábra. Labdapattogtatás

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/blob/master/szohossz/bitshift.cpp>

A szóhossz megnézéséhez a bitenkénti léptetés operátort használjuk:

```
while (szam <=1) {  
    cout<<szam<<' \n' ;  
    counter++;  
}
```

Ez annyit jelent hogy az egyest egyre jobban balra toljuk és jobbról 0-ákkal pótoljuk.

Ez azt eredményezi, hogy 2-nek hatványait kapjuk és amikor eléri az int maximális méretét utána 0-t kap eredményül, mivel a bitsorozat teljesen ki 0-ázódik. Tehát a 32. lépéstre, nem lesz olyan bit, amin képesek leszünk ábrázolni az 1 est.

```
2  
4  
8  
16  
32  
64  
128  
256  
512  
1024  
2048  
4096  
8192  
16384  
32768  
65536  
131072  
262144  
524288  
1048576  
2097152  
4194304  
8388608  
16777216  
33554432  
67108864  
134217728  
268435456  
536870912  
1073741824  
2147483648  
Az egy 31 lepes utan lesz 0.
```

2.6. ábra. Szóhossz

Igazából 32 bites a szóhossz csak az elsőt nem számolja szóval 31+1.

De ezen könnyen javíthatunk ha, a while helyett do while-t használunk:

```
do{
    cout<<szam<<' \n';
    counter++;
} while (szam <=1);
```

```
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/bevprog$ g++ bitshift.cpp -o bitshift
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/bevprog$ ./bitshift
1
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
2147483648
Az egy 32 lepes után lesz 0.
```

2.7. ábra. Szóhossz jó megoldás

2.6. Helló, Google!

Tutor

Ebben a feladatban tutoráltam Ádám Petrát és Duszka Ákos Attilát.

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

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/blob/master/pagerank/pagerank.c>

Tanulságok, tapasztalatok, magyarázat...

A pagerank algoritmust a google találta ki azért, hogy megkönnyítsék a weben való keresést. Maga a pagerank egy számba sűrít a weblap értékét. A lényege, hogy minél több oldal mutat a weblapunkra annál értékesebb. Ez azért van mert, úgy gondolták a google alapítói, hogy a weboldal készítői, azért linkelnek be oldalakat mert hasznosnak találják.

Ez felfogható úgy is, mint egy választás. És minden oldal képes szavazni és az, hogy valaki a mi linkünket használja olyan mintha ránk voksolna és akinek több a szavazata az van előrébb a rangsorban.

Első lépésként megadjuk a kapcsolati gráfot. Tehát, hogy melyik oldal melyik oldalra mutat. Ezt egy mátrixban tároljuk le, mivel 4 honlappal dolgozunk, ezért egy 4x4 es mátrix lesz:

```
#include <stdio.h>
#include <unistd.h>
int main() {
    double graf[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}
    };
}
```

Azért double mivel a pagerank nem feltétlenül csak egész szám lehet.

Ezután létrehozunk még 2 db egydimenziós tömböt. Az egyikben a végleges pagerankot tároljuk míg a másikban az ideiglenest. Az ideiglenes vektorban minden oldal pagerankját 1/4-re állítjuk mivel 4 oldal van.

Majd indítunk egy végleges ciklust, amely addig fut, amíg a pagerank kisebb nem lesz, mint a dumping faktort az az a csillapító értéket, most ez 0.00001-ben lett meghatározva

A végtelen ciklus elején áttöltjük az ideiglenes pagerankból az értékeket a végleges pagerank tömbünkbe.

Ezt követően indítunk egy egybeágyazott for ciklust, amely a letárolt kapcsolati gráfos tömb(a szétoztott szavazatokat tárolja) sorait megszorozza a jelenlegi pagerankkal és a sorok összegét, mármint egyessével egy értékbe tömöríti és azt betölti az ideiglenes pagerankba és ez addig folytatódik, amíg kinem lép a végtelen ciklusból.

```
#include <stdio.h>
#include <unistd.h>
#define dumping_factor 0.0001

int main() {
    while(1)
    {
        for(i=0;i<4;i++)
        {
            PR[i] = PRt[i];
        }
        for (i=0;i<4;i++)
        {
            double temp=0;
            for (j=0;j<4;j++)
                temp+=graf[i][j]*PR[j];
            PRt[i]=temp;
        }
    }
}
```

```
        if ( dif(PR,PRT, 4) < dumping_factor)
break;
}
}
}
```

A távolság függvény paraméterként megkapja a végleges pagerankot és az ideiglenest és, hogy hány db oldal van. Majd kivonja a pagerank i-edik eleméből az ideiglenes pagerank i-edik elemének értékét és ezek abszolút értékét összeadja a tav nevű változóban, amely a függvény visszatérési értéke lesz.

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

double dif(double pagerank[],double pagerank_temp[],int db)

{
double dif= 0.0;
int i;
for(i=0;i<db;i++)
{
    dif +=fabs(pagerank[i] - pagerank_temp[i]);
}
return dif;
}
```

Érdekesség ha az egyik oldal nem mutat semmire.Tehát ha az utolsó oszlopot mondjuk teljesen ki 0-ázzuk akkor a pagerank is kinullázódna ha 2 tizedes jegyik néznénk az értéket.

```
PageRank [0]: 0.000010
PageRank [1]: 0.000047
PageRank [2]: 0.000026
PageRank [3]: 0.000010
```

2.8. ábra. Pagerank hibásmegoldás

A helyes megoldás:

```
PageRank [0]: 0.090908
PageRank [1]: 0.545453
PageRank [2]: 0.272730
PageRank [3]: 0.090908
```

2.9. ábra. Pagerank helyes megoldás

2.7. 100 éves a Brun téTEL



Tutorált

Ebben a feladatban tutorált Duszka Ákos Attilát.

Írj R szimulációt a Brun téTEL demonstrálására!

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

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

Prímnek nevezük azokat a számokat, amelyek csak 1-el és önmagukkal oszthatók. Ikerprímek azok a prím-számok, amelyek különbsége 2.

A program egy megadott x értékig kikeresi a prímeket. Majd megnézi a köztük lévő differenciát (diff), ahol ez a differencia 2, annak az indexét egy tömbben(idx) tárolja (de csak az ikerprímpár első tagjának indexét, ezért kell a t2primes-nál a +2) tehát a prímek közül kiszűri, hogy melyek ikerprímek.

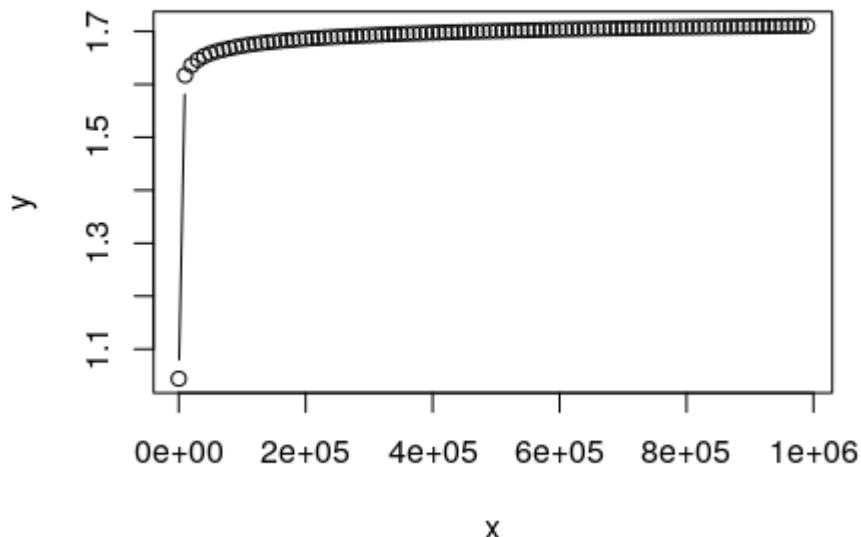
```
primes = primes(x)
diff = primes[2:length(primes)]-primes[1:length(primes)-1]
idx = which(diff==2)
t1primes = primes[idx]
t2primes = primes[idx]+2
```

Majd az rt1plus2-ben összeadjuk ezeknek a reciprokát végül a függvényünk visszatérési értéke az rt1plus2-nek az összege.

A seq függvény hasonló a for ciklushoz seq(from, to, by=), from, hogy mettől(13) to, hogy meddig(1000000) és a by, hogy milyen lépésszámmal(10000). Ez határozza meg az x tengely beosztását.

A sapply függvény az x ekhez rendeli egyessével az stp függvényben kapott értékeket y-ként.

Végül a plot kirajzolja a függvényt.



2.10. ábra. A konstans közelítése

A képen látható, hogy a párosprímek reciprokának összege egyre jobban tart a 2 felé, tehát egy véges értékhez konvergál, amely a Brun konstans azaz a Brun téTEL teljesül.

De ezzel még mindig nem tudjuk eldönteni, hogy végtelen vagy véges számú prímszám van mert úgysem lépik át ezt a határt csak megközelítik.

2.8. A Monty Hall probléma

Tutorált

Ebben a feladatban tutorál Duszka Ákos Attilát.

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...

A Monty Hall probléma lényegében 3 ajtó közül kell kiválasztanunk a nyerteset viszont ha nem találjuk el akkor újra kezdhetjük.

Annak a valószínűsége, hogy egyből a jó ajtót találjuk el 1/3 és az, hogy rosszat 2/3. Viszont a választás után a 3 ajtó közül a műsorvezető kinyit egyet, amelyik mögött nem a nyeremény van.(A játékvezető tudja melyik ajtó mögött van a nyeremény) Majd ezután megkérdei a játékos, hogy szeretne -e változtatni a választásán. Elvileg az ajtó nyitása után a nyerési arányunk redukálódik 1/2-re, hogy nyerünk és 1/2-re, hogy vesztünk. A nagy kérdés itt az, hogy megéri -e váltanunk.Ez a program pont ezt szimulálja.

A kísérletek száma változóban definiáljuk, hogy hányszor fusson le a kísérlet.Azaz a minták száma.

A kiserlet és a jatekos tömbök, amelyeket 1 és 3 közé eső számokkal tölt fel a sample. A műsorvezető egy vektor amelyet ugyan olyan méretűre deklarálunk mint a kísérletek száma.

Egy for ciklussal bezárjuk a tömböt és ha a játékos eltalálja, hogy melyik ajtó mögött van akkor a mibol tömbbe a másik két ajtó lehetősége kerül.Ha nem találja el akkor csak egyetlen érték az az ajtó ami mögött nincs semmi,de nem választotta a játékos.

Ezután a műsorvezető úgymond kinyit egy ajtót tehát választ egyet a mibol tömbből.

Majd lefut egy feltétel, amely megmutatja hányszor nyerne a játékos ha nem változtat.Tehát a tömbbe azok az indexek kerülnek amikor a jatekos és a kiserlet megegyezik.

Létrehozunk egy új vektort amiben megváltoztatjuk a választást úgy, hogy kivételként adjuk a műsorvezető által kinyitott ajtót és a játékos által választottat.

Végül lefuttatunk egy ugyan ilyen feltételes vizsgálatot, hogy mi lett volna ha minden változtatunk. És itt is ugyan úgy letároljuk egy tömbbe, hogy mely indexknél nyert a játékos. És kiiratjuk a statisztikát, amely megmutatja, hogyan járnánk jobban ha minden változtatnánk vagy ha tartózkodnánk az először választott ajtóhoz.

Egy példa a program futására:

```
> valtoztatesnyer = which(kiserlet==valtoztat)
>
>
> sprintf("Kísérletek száma: %i", kiserletek_szama)
[1] "Kísérletek száma: 100"
> length(nemvaltoztatesnyer)
[1] 34
> length(valtoztatesnyer)
[1] 66
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.5151515
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 100
>
```

2.11. ábra. Monty Hall teszt

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

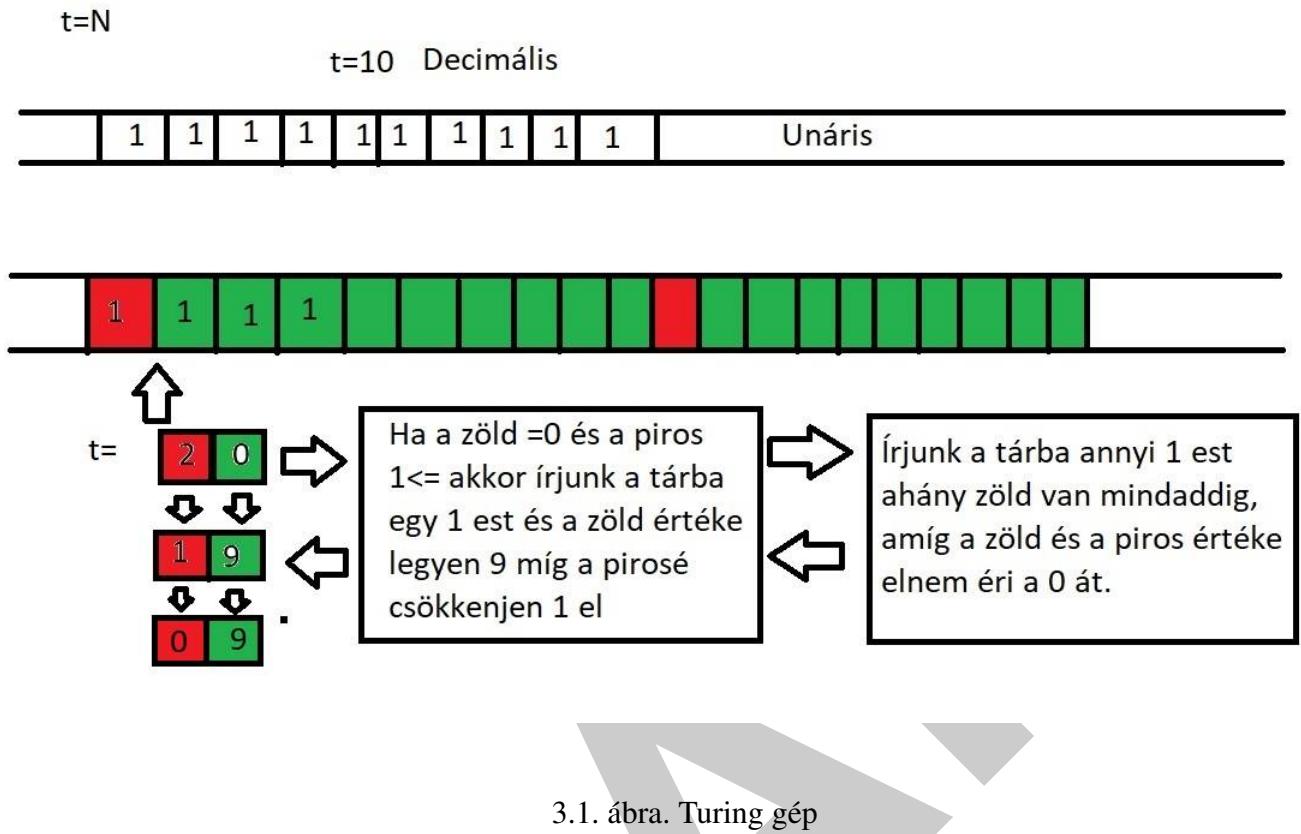
Megoldás videó:

Megoldás forrása: Az előadás fóliája.

Tanulságok, tapasztalatok, magyarázat...

Decimálisból unárisba, úgy váltunk, hogy annyi 1 est írunk le, amennyi a szám értéke vagy másképp fogalmazva amilyen messze van a 0-tól. Pl.: $n=90$ esetén 90 db 1 est kell leírnunk.

Tehát itt pozitív számokat tudunk ábrázolni. A megvalósítás 2 féle lehet vagy indítunk egy for ciklust 0-tól és minden egyes lépésnél egy stringbe összefűzzük az egyeseket. Vagy pedig a számtól indítunk egy ciklust 0-ig és ugyan ezt tesszük.



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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A környezetfüggő grammátika olyan szabályok összessége, amely segítségével a nyelvben minden jelsorozatot képesek vagyunk előállítani.

- A grammátika forrása a fólia: A, B, C „változók” a, b, c „konstansok” $A \rightarrow aAB$, $A \rightarrow aC$, $CB \rightarrow bCc$, $cB \rightarrow Bc$, $C \rightarrow bc$ S-ból indulunk ki. $S \rightarrow aC$ $aC(C \rightarrow bc)$ abc
- $S \rightarrow aAB$
- $aAB(A \rightarrow aAB)$
- $aaABB(A \rightarrow aAB)$
- $aaaABBB(A \rightarrow aAB)$
- $aaaaABBBB(A \rightarrow aC)$

- aaaaaCBBBB(CB → bCc)
 - aaaaabCcBBB(cB → Bc)
 - aaaaabCBcBB(CB → bCc)
 - aaaaabbCccBB(cB → Bc)x2
 - aaaaabbCBccB(CB → bCc)
 - aaaaabbbCcccB(cB → Bc)x3
 - aaaaabbbCBccc(CB → bCc)
 - aaaaabbbbCcccc(C → bc)
 - aaaaabbbbbcccc
 - Ez a grammaтика biztosan ezt a nyelvet generálja.
- S, X, Y „változók” a, b, c „konstansok” S → abc, S → aXbc, Xb → bX, Xc → Ybcc, bY → Yb, aY → aaX, aY → aa S-ből indulunk ki A grammaтика forrása a fólia.
- S(S → aXbc)
 - aXbc(Xb → bX)
 - abXc(Xc → Ybcc)
 - abYbcc(bY → Yb)
 - aYbbcc(aY → aaX)
 - aaXbbcc(Xb → xB)2*
 - aabbXcc(Xc → Ybcc)
 - aabbYbcc(bY → Yb)
 - aaYbbbccc(aY → aaX)
 - aaaXbbbccc(Xb → bX)3*
 - aaabbbXccc(Xc → Ybcc)
 - aaabbbYbccccc(bY → Yb)3*
 - aaaYbbbbcccc(aY → aa)
 - aaaabbbbcccc
 - Környezetfüggő! Az abc-nek bárhanyadik hatványa előállítható.

3.3. Hivatkozási nyelv

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

Megoldás video:

Megoldás forrása: <https://github.com/Savitar97/Prog1/blob/master/hivatkozas/hivatkozas.c>

<https://hu.wikipedia.org/wiki/Backus%E2%80%93Naur-forma>

Tanulságok, tapasztalatok, magyarázat...

A C utasítások a C nyelv kulcsszavai. A C nyelv tartalmazza a többsoros utasítás blokkokat, iterációkat (for, while, do-while), vezérlő szerkezeteket (if, switch), operátorok (++,-,!=, stb), deklarációk.

Backus normal form egy általánosított leírása a programozási nyelveknek. Nyelv független. Vagyis ez a séma ráilleszthető a legtöbb programozási nyelvre és használható a nyelvekben írt programok leírására.

```
Forrás:https://arato.inf.unideb.hu/batfai.norbert/UDPROG/ ←
    deprecated/Prog1_1.pdf?fbclid= ←
    IwAR1eImHN5PINGTMnhpJItlqA_PtEcfnGKqndS6nt5wNTFr_X- ←
    hcSdiVr5iQ
```

A backus leírás röviden:

```
<szimbólum> ::= <kifejezés a szimbólumra>
    van egy szimbólum aztán a ::= után van egy formai leírása
<egész szám> ::= <előjel><szam>
<előjel> ::= [-+]
<szam> ::= <szamjegy>{<szamjegy>}
<szamjegy> ::= 0|1|2|3|4|5|6|7|8|9
    /*A forrás az előadás pptjéről származik.*/
```

A c89-ben még nem lehet egysoros kommenteket írni (//) és szintén nem lehet a for ciklus fejében változót deklarálni, amit a c99 már enged. A különböző változatoknál a fordítást a -std kapcsolóval érjük el ez, így néz ki a gyakorlatban:

gcc -o hivatkozas -std=c89 hivatkozas.c

gcc -o hivatkozas -std=c99 hivatkozas.c

```
#include <stdio.h>
int main(){
    int i;
    for(i=0;i<10;i++)
        /*Ez így lefog futni c89-ben is.
        Viszont
        for(int i=0;i<10;i++)
            ez nem.
    */
```

```
    return 0;  
}
```

A következő hibaüzenetet kapjuk:

```
hivatkozas.c: In function 'main':  
hivatkozas.c:5:2: error: C++ style comments are not allowed in ISO C90  
  //ahahahah  
  ^  
hivatkozas.c:5:2: error: (this will be reported only once per input file)  
hivatkozas.c:6:2: error: 'for' loop initial declarations are only allowed in C99  
or C11 mode  
  for(int i=0;i<10;i++)  
  ^~~  
hivatkozas.c:6:2: note: use option -std=c99, -std=gnu99, -std=c11 or -std=gnu11  
to compile your code
```

3.2. ábra. Hibaüzenet

Emellett van olyan ami a c89-ben működik, de c99 ben nem. Ilyen a következő kódcsipet:

```
#include <stdio.h>  
  
int main()  
{  
  
    int restrict=1;  
    if (restrict) printf("restricted");  
    return 0;  
}
```

Azért fordulhat le mivel a restrict még nem kulcsszó c89-ben, viszont c99-ben már igen. A restrict megadja, hogy mely pointerek férhetnek hozzá az adott memória területhez, ezeket a pointereket nem lehet módosítani.

3.4. Saját lexikális elemző

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/real>

```
%{  
#include <stdio.h>  
int counter = 0;  
%}  
digit [0-9]  
%%  
{digit}*(\.{digit}+)? {++counter;  
    printf("[realnum=%s %f]", yytext, atof(yytext));}  
%%  
int  
main ()  
{  
    yylex();  
    printf("Valós számok száma:[ %d]\n", counter);  
    return 0;  
}
```

A lexernél az egyes részeket %%-jelek választják el. Az első résznél jön a könyvtár hivatkozás és a deklarációk és az első rész végén definiálunk(ilyen itt a digit amiben a számokat definiáljuk) a definícióknál lehet megadni a karakter csoportokat, amelyeket keresni akarunk a beolvasott szövegből.

A következőben jöhetnek a formázási szabályok itt mondhatjuk meg, hogy mi történjen ha megtalálja az adott karakter sorozatot vagy karaktert a lexer. Itt már használhatjuk a definíciókat.

Az első kapcsos zárójelben megadtuk, hogy számot keresünk, ezután a csillag azt jelenti, hogy bármennyi-szer előfordulhat 0 vagy akárhány. Majd egy pontnak kell követni azután a + miatt jönnie kell egy számnak legalább vagy többnek. A kérdőjel viszont azt jelzi, hogy nem muszáj pontnak következnie és utána számnak ez azért kell mivel az egész számok is beletartoznak a valós számokhoz.

Majd azt adjuk meg ha találunk ilyen karaktersorozatot akkor a countert növeljük-1 el. És írjuk ki ezt a karakter sorozatot -között az atof függvény pedig ezt a karaktersorozatot valós számmá konvertálja.

A programot a következő képpen kell fordítanunk:

lex -o real.c real.l

Ilyenkor a lexer megírja a c programot, majd a létrehozott .c fájlt gcc-vel fordítjuk.

gcc -o real real.c -lfl

Az utolsó részben jön a program main része itt meghívjuk a yylex() függvényt és kiirassuk, hogy hány db valós számot találtunk.

A programot a **Ctrl+D**-vel tudjuk leállítani.

```

Tevékenységek ┌ Terminál ─
Fájl Szerkesztés Nézet Keresés Terminál Lapok Súgó
nemesis@nemesis-Aspire-A515-51G: ~/Letöltések/netbeans/netbeans/bin × nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/Prog1/real ×
vedes3
vedes3.cpp
vedes4
vedes4.cpp
vedes5
vedes5.cpp
vegtelen4.c
xd
xd.cpp
z
z3
z3a7.cpp
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/bevprog$ cd ..
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas$ cd Prog1/real/
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/Prog1/real$ ls
real real.c real_l
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/Prog1/real$ ./real
21312412.23131 adasdadawaras
[realnum=21312412.23131] adasdadawaras
xsaxasdawd12312414251241213123
xsaxasdawd[realnum=12312414251241213123 12312414251241213952.000000]
213123xwd123123
[realnum=213123 213123.000000]xwd[realnum=123123 123123.000000]
d13ed4tf312e1dd445f5t6z6
d[realnum=13 13.000000]f[realnum=4 4.000000]f[realnum=32 32.000000]e[realnum=1 1.000000]d[realnum=1 1.000000]dd[realnum=445 445.000000]f[r
ealnum=5 5.000000]t[realnum=66 66.000000]z[realnum=6 6.000000]
e1e1be138ze81hs1sh1397sh1
e[realnum=1 1.000000]e[realnum=1 1.000000]be[realnum=138 138.000000]ze[realnum=81 81.000000]h[realnum=9 9.000000]s[realnum=1 1.000000]h[realnu
m=9 9.000000]sh[realnum=1397 1397.000000]sh[realnum=1 1.000000]
sis139sihs881s13s13s13
s[realnum=1 1.000000]s[realnum=139 139.000000]s[realnum=13 13.000000]hs[realnum=8 8.000000]s[realnum=1 1.000000]s[realnum=13 13.000000]s[realn
um=13 13.000000]s[realnum=13 13.000000]
sish139sijs0s1s19
s[realnum=1 1.000000]sn[realnum=19 19.000000]s[realnum=1 1.000000]jsö[realnum=1 1.000000]s[realnum=1 1.000000]s
Valós számok száma:[ 35 ]
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/Prog1/real$ 
```

3.3. ábra. Lexikális elemző

3.5. I33t.I

Lexelj össze egy I33t cipher-t!

Megoldás videó:

Megoldás forrása:<https://github.com/Savitar97/Prog1/tree/master/leet>

```

/*
Fordítás:
$ lex -o lexer.c lexer.l

Futtatas:
$ gcc lexer.c -o lexer -lfl
(kilépés az input vége, azaz Ctrl+D)

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

#define LEXERSIZE (sizeof lexer / sizeof (struct cipher))

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

```
    } lexer [] = {  
  
{'0', {"Ω", "○", "°", ""}},  
{'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<LEXERSIZE; ++i)  
    {  
  
        if(lexer[i].c == tolower(*yytext))  
        {  
  
            int r = 1+(int) (100.0*rand() / (RAND_MAX+1.0));  
  
            if(r<91)  
                printf("%s", lexer[i].leet[0]);  
            else if(r<95)  
                printf("%s", lexer[i].leet[1]);  
            else if(r<98)  
                printf("%s", lexer[i].leet[2]);  
            else  
                printf("%s", lexer[i].leet[3]);  
  
            found = 1;  
            break;  
        }  
  
        if(!found)  
            printf("%c", *yytext);  
  
    }  
%%
```

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

Elsőként definiáljuk a lexer struktúra méretét. Ezt a define LEXERSIZE adja meg vagyis a sorok számát ez 36.

Ezután létrehozunk egy struktúrát. Amiben definiálunk egy karakter változót és egy 4 elemű mutató tömböt. Ha több variációt akarunk behelyettesítésre, akkor növeljük ennek a számát.

Ezután létrehozzuk a struktúra fő részét itt az első elem karakter típusú, amelyet majd vizsgálunk, a 2. elem egy 4 elemű tömb, amelyben a karakter helyettesítési lehetőségeit tároljuk.

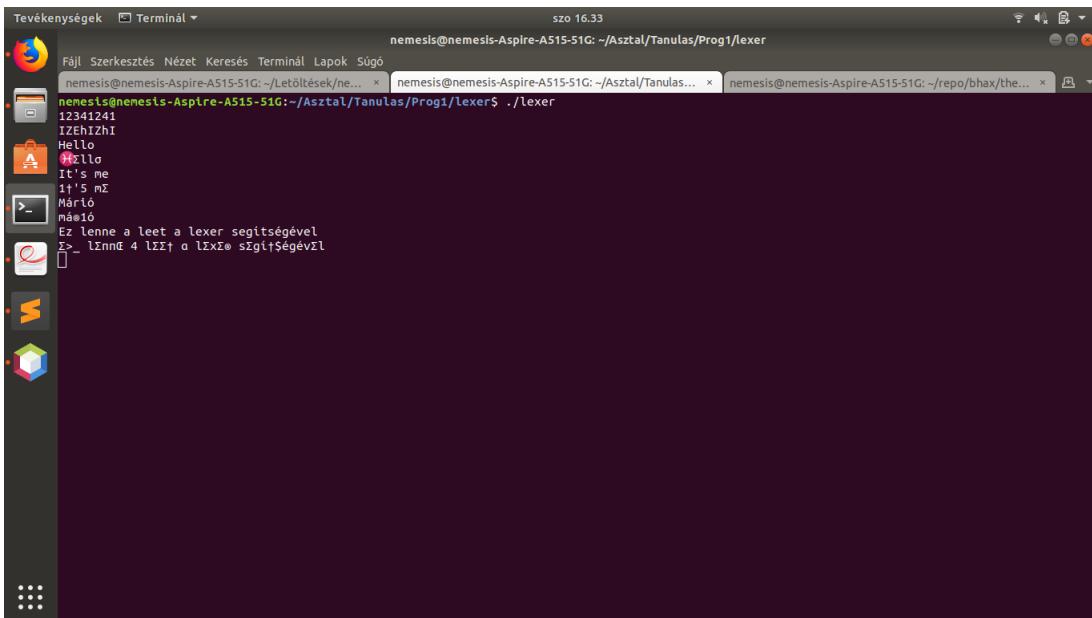
Ez volt az első része a lexernak ahol definiálunk és könyvtárakat hívunk meg. Most a definíciókat kihagyjuk.

Következőnek az utasítás része jön a lexernak. Itt behozunk egy változót, amely azt jelzi, hogy megtalálta -e a karaktert a struktúrában ha nem akkor visszaadja majd a lent lévő if magát a karaktert. Majd indítunk egy fort, amely átnézi a struktúrát keresve a beolvastott karaktert, amelyet kisbetűre alakítunk, hogy ne kelljen külön kezelni a kis és nagy betűket.

Ha megtaláltuk a karaktert akkor egy random számot generálunk, amely segít, hogy véletlenszerűen válasszunk a 4 opciónál közül, amelyet a 4 if segítségével érünk el és visszaadjuk, hogy megtaláltuk a karaktert found.

Az utolsó részben a mainben találjuk az srandomot, amely a randomot hívja ehhez a **time.h** szükséges. A random generálásához az időt használja és hozzáadja a getpidet, amely az `unistd.h` könyvtárban van, ez azért szükséges, hogy jobban generáljon random számokat, vagyis nagyobb legyen a számok randomitása. Majd végül meghívjuk a yylex() függvényt.

A program futása során a lexer cseréli a beírt karakter sorozatot és ez 1337 5P34CH.



```
nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/Prog1/lexer$ ./lexer
12341241
IZEHIZHI
Hello
It's me
it's m2
Máról
máris
Ez lenne a leet a lexer segítségével
Σ>_ linne 4 lzzt a lizzi sZglfSégevzl
```

3.4. ábra. 1337 5P34CH

3.6. A források olvasása

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

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

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

Bugok

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

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása:<https://github.com/Savitar97/Prog1/tree/master/3.6>

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

```
signal.c:2: Include file <unistd.h> matches the name of a POSIX library, ←
but
the POSIX library is not being used. Consider using +posixlib or
+posixstrictlib to select the POSIX library, or -warnposix to suppress ←
this
message.
Header name matches a POSIX header, but the POSIX library is not selected ←
.
(Use -warnposixheaders to inhibit warning)
signal.c: (in function jelkezelo)
signal.c:5:20: Parameter a not used
A function parameter is not used in the body of the function. If the ←
argument
is needed for type compatibility or future plans, use /*@unused@*/ in the
argument declaration. (Use -paramuse to inhibit warning)
signal.c: (in function main)
signal.c:16:4: Return value (type [function (int) returns void]) ignored:
    signal(SIGINT, S...
Result returned by function call is not used. If this is intended, can ←
cast
```

```
result to (void) to eliminate message. (Use -retvalother to inhibit ←
warning)
signal.c:5:6: Function exported but not used outside signal: jelkezelő
A declaration is exported, but not used outside this module. Declaration ←
can
use static qualifier. (Use -exportlocal to inhibit warning)
signal.c:8:1: Definition of jelkezelő
```

A signalis program ignorálja(SIG_IGN) vagy másnépp elkapja a signalokat ilyen például a **Ctrl+C**. De nem minden signalt tud ignorálni. A SIGINT itt magát a signalt jelzi a 2 es signal neve SIGINT. A program a signal kezelést átadja a jelkezelésnek, tehát innentől nem a signal hajtódiik végre hanem a jelkezelő.

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

Egy for ciklus, amely 0-tól meg 5-ig. A `++i` jelentése pre-increment, ez azt jelenti, hogy a művelet lefutása előtt megnöveli a változó értékét egyel.

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

For ciklus amely 0-tól megy 5-ig. Az `i++` az post-increment, vagyis előbb hajtódiik végre a művelet és csak utána növeli az `i` értékét 1 el.

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

Splint error:

```
forforth.c: (in function main)
forforth.c:7:24: Expression has undefined behavior (left operand uses i,
modified by right operand): tomb[i] = i++
Code has unspecified behavior. Order of evaluation of function parameters ←
or
subexpressions is not defined, so if a value is used and modified in
different places not separated by a sequence point constraining ←
evaluation
order, then the result of the expression is unspecified. (Use -evalorder ←
to
inhibit warning)
```

```
Finished checking --- 1 code warning
```

A tömbnek az elemét egyenlővé teszi az `i`-nek az értékével kivéve az első elemét.

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

Elindít egy for ciklust, amely addig fut amíg az i kisebb mint n, ezen kívül d és s egy tömb mutatója és azokat a tömb elemeket, amelyekre a d mutat kicseréli azokra, amelyekre az s mutat.

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

splint által elkapott error:

```
forsix.c: (in function main)
forsix.c:10:24: Argument 2 modifies a, used by argument 1 (order of ←
    evaluation
        of actual parameters is undefined): f(a, ++a)
Code has unspecified behavior. Order of evaluation of function parameters ←
    or
subexpressions is not defined, so if a value is used and modified in
different places not separated by a sequence point constraining ←
    evaluation
order, then the result of the expression is unspecified. (Use -evalorder ←
    to
inhibit warning)
forsix.c:10:32: Argument 1 modifies a, used by argument 2 (order of ←
    evaluation
        of actual parameters is undefined): f(++a, a)
forsix.c:10:19: Argument 2 modifies a, used by argument 3 (order of ←
    evaluation
        of actual parameters is undefined): printf("%d %d\n", f(a, ++a), f(++a, ←
            a))
forsix.c:10:30: Argument 3 modifies a, used by argument 2 (order of ←
    evaluation
        of actual parameters is undefined): printf("%d %d\n", f(a, ++a), f(++a, ←
            a))
forsix.c:2:5: Function exported but not used outside forsix: f
A declaration is exported, but not used outside this module. Declaration ←
    can
use static qualifier. (Use -exportlocal to inhibit warning)
forsix.c:5:1: Definition of f

Finished checking --- 5 code warnings
```

Az első függvénynél az a értéke 2 vel nő, míg a 2. nál 1 el.

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

Splint:

```
forseven.c:2:5: Function exported but not used outside forseven: f
```

```
A declaration is exported, but not used outside this module. Declaration ↵
can
use static qualifier. (Use -exportlocal to inhibit warning)
forseven.c:5:1: Definition of f
```

Kiirja a függvényel módosított a és az a értékét.

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

Splint error:

```
foreight.c:6:2: Parse Error. (For help on parse errors, see splint -help
parseerrors.)
```

Konvertálási hiba a pointer értékeket %d helyett a %p-vel kell kiíratni printf-el. Tehát egy memória címet egészként akart kiíratni.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

Tanulságok, tapasztalatok, magyarázat...

1. Bármely x-hez létezik olyan y, hogy az y nagyobb mint x és y prím.
2. Bármely x-hez létezik olyan y, hogy y nagyobb mint x és y prím és y+2 is prím.
3. Létezik olyan y, hogy bármely x esetén ha x prím akkor az x kisebb mint y.
4. létezik olyan y, hogy bármely x esetén ha y kisebb mint x akkor x nem prím.

Az első állítás azt mondja ki, hogy a prímszámok száma végtelen. Míg a második azt jelenti, hogy végtelen sok ikerprím létezik. Itt az SSy a successor function vagy másnéven a rákövetkező függvény, tehát **S(S(y))=(y+1)+1**.

A 3. állítás ennek az ellenkezőjét fejezi ki, tehát azt, hogy a prímszámok száma véges. A negyedik állítás ezzel ekvivalens, mivel azt mondja ki, hogy létezik olyan y amitől nincs nagyobb prímszám.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- `int a;`
- `int *b = &a;`
- `int &r = a;`
- `int c[5];`
- `int (&tr)[5] = c;`
- `int *d[5];`

- ```
int *h ();
```
- ```
int *(*l) ();
```
- ```
int (*v (int c)) (int a, int b)
```
- ```
int (*(*z) (int)) (int, int);
```

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/deklaracio>

Tanulságok, tapasztalatok, magyarázat...

Az int a létrehoz egy változót, amely int típusú az az egész, a néven. Egy változónak van neve, típusa, hatásköre, mérőcímé ahol tárolódik, értéke.

int *b létrehoz egy mutatót, amely a-nak a memória címére hivatkozik.

Az r rekurzívan hivatkozik a értékére. Vagyis ugyan arra a memória területre hivatkozik mint az a. Tehát ha r értékét változtatjuk akkor a-nak az értéke is változik. Szemléltetésként a következő kód szolgál:

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

using namespace std;

int main()
{
    int a=5;
    int &r=a;
    int *d=&a;
    int *b=&r;
    cout<<a<<' \n' <<r<<' \n' <<d<<' \n' <<b<<' \n' ;
    r=r+2;
    cout<<a<<' \n' <<r<<' \n' <<d<<' \n' <<b<<' \n' ;
    return 0;
}
```

Futtatva következő eredményt kapjuk:

```
5  
5  
0x7ffc06a097c  
0x7ffc06a097c  
7  
7  
0x7ffc06a097c  
0x7ffc06a097c
```

3.5. ábra. Deklaráció

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

A c deklarál egy 5 elemű tömböt.

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

A tr tömb rekurzívan hivatkozik a c tömbre.

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

Létrehoz egy 5 elemű mutató tömböt.

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

Olyan függvény ami egy egészre mutatót ad vissza.

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

Egy egészre mutatót ad vissza a függvényt.

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

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

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

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

```
int (*sumormul (int c)) (int a, int b)
{
    if (c)
        return mul;
    else
        return sum;

}
int
main ()
{
    int (*f) (int, int);
    int (*(*g) (int)) (int, int);
    g = sumormul;
    f = *g (0);
    return 0;
}
```

Itt az f egy olyan pointer ami egy int típusú függvényre mutat.Tehát egyszerűen meghívhatunk függvényeket vele, ha egyenlővé tesszük egy 2 egészet kapó függvénnyel.Itt például a szummal ami 2 egészet kap. int sum(int a,int b)

Az f-nek megadjuk a g pointert.Viszont a g már a sumormulra mutat.Tehát egy olyan függvényre, ami egy egész számot kér és egy két egész számot kapó függvényre hivatkozik.Ha a g-nek 0 át adunk akkor a sum, ha ettől eltérő értéket akkor a mul fog végrehajtódni.Amit így már az f(int,int)-el tudunk hivatkozni és attól függ,hogy melyik függvényre mutat, hogy a g-nek az értéke alapján a g melyik függvényre mutat.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

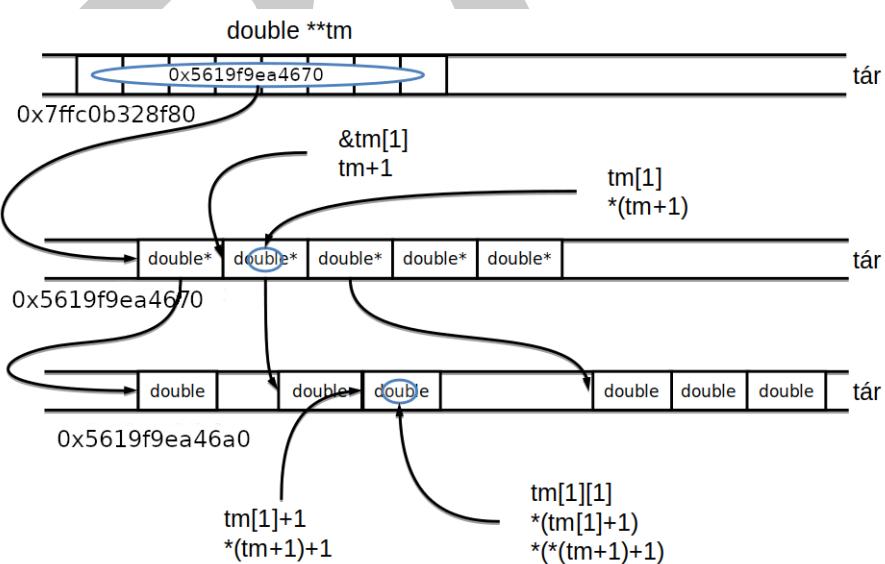
Írj egy olyan malloc és free párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

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

Megoldás forrása: https://gitlab.com/Savitar97/bhax/blob/master/thematic_tutorials/bhax_textbook/Caesar-tm.c

Tanulságok, tapasztalatok, magyarázat...

Az alsó háromszög mátrix lényege, hogy a főátló fölött csupa 0-érték helyezkedik el. Az alsó háromszög mátrixokat sorfolytonosan szoktuk ábrázolni, ha $M[i,j]$ a $j > i$ akkor a j értéke 0.



4.1. ábra. Memória példa

Az nr-ben inicializáljuk, hogy hány soros és oszlopos legyen a mátrixunk. Ezután létrehozunk(deklarálunk) egy double típusú mutatóra mutatót, amely egy 2 dimenziós tömb lesz, ez látható a fenti képen.

Majd kiiratjuk a címét ennek a mutatónak. Itt még nincs érték adva neki. Ezután a tm-nek a malloc típuskényszerítve double-re visszaad egy pointert a dinamikusan lefoglalt terüettel. Egy pointernek lefoglalt hely függ, hogy hány bites a rendszerünk mivel általában 64 bitesek a rendszerek ezért ez 8 bájtos lesz, itt most megszorozzuk az nr el, tehát 40 bájtot foglal le a malloc. Ha nem tudod helyet foglalni akkor visszaad valamilyen hibát, itt a hibakezelést egy egyszerű return -1 oldja meg.

Ezzel lefoglaltuk a sorokat azonosító mutatóknak a helyet(5 mutatónak a helyét).

Ezután a következő mallockal a sorokban elhelyezkedő elemekre foglalunk le helyet. Mivel a double mérete is 8 bájt ezért az első sorba 1*8bájtot a következőben 2*8 bájtot és így tovább foglalunk le. Miután ez lefut az elemeknek a helye is le lesz foglalva.

Majd kiiratjuk az első sorra hivatkozó mutató memória területét.

Majd feltöltjük a mátrixunkat elemekkel ez a forcikus 0-14 ig fogja feltölteni.

A következő értékkedésekkel kicséréljük a 4.sor elemeit a megadott elemekre csak különböző hivatkozásokkal van felírva, de minden ugyan azt jelenti.

A program végén a free felszabadítja a malloc által lefoglalt de nem használt memória területeket.

```
nemesis@nemesis-Aspire-A515-51G: ~/repo/bhax/thematic_tutorials/bhax_textbook/Caesar$ ./tm
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
0.000000, 10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
```

4.2. ábra. Alsó háromszög mátrix

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/exor>

Tanulságok, tapasztalatok, magyarázat...

Az exor titkosítás egy viszonylag egyszerű, de mégis hatékony titkosítási módszer tud lenni.

Ha törekszünk arra, hogy a kulcs minél hosszabb legyen. Ugyanis a program működése, hogy a szöveg karakterein egyessével bitenkénti xor műveletet hajtunk végre. Viszont ha a szöveg hosszabb, mint a kulcs akkor a kulcs ismétlődésével érjük el a titkosítást. Tehát például 8 karakterű kulcsnál a szöveg első karakterét a kulcs első karakterével exorozzuk, a másodikat a másodikkal és így tovább.

A programban elsőként definiáltuk a max kulcsméretet és a max buffer méretet. Deklarálunk 2 char tömböt ezek segítségével, majd deklarálunk és inicializáltunk 2 változót ez a kulcs_index és az olvasott_bajtok.

Majd az int kulcs_meretbe megadjuk, hogy mekkora a kulcsunk ezt az argv[1] karakter tömbjének mérete adja. Ugyanis az argv az a futtatáskor bemenő egységeket nézi az argv[0] maga a futtatás parancsa a terminálta a ./fájlnév. Az 1 itt a kulcs amit megadunk. Ezek char mátrixok. Azt, hogy hány bemenet van az az ilyen char tömb az az argc-ben van letárolva ami ezeknek a számát kapja értékül.

A strncpy-vel másoljuk át a bemenetben megadott kulcsot az argv[1]-tömbből a kulcs tömbbe. A while beolvassa a txt-t az első karaktertől a végéig (a buffer méretéig), a read visszatérési értéke a beolvastott bajtok száma. A for ciklus végig megy a szövegen és ez titkosítja a maradékos osztással éri el a program, ha kisebb a kulcs mint a szöveg, akkor a kulcs ismétlődésével titkosítunk.

A write és a read is ha negatív értéket kap akkor hibát fog kiírni.

A readnél a 0 azt jelenti, hogy a standard inputról olvasson. A writenál az 1 es, hogy a standard outputba írjon azaz ez oldja meg, hogy a megadott szövegfájlt olvassa be és a megadott fájlba írja ki.

Fordítani a szokásos módon tudjuk. Futtatási segédlet a következő minta:

```
./exor kulcs <bemenőszöveg.txt>titkosítottszöveg.txt
```

4.3. Java EXOR titkosító



Tutorált

Ebben a feladatban tutorált Molnár Antal.

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/exor>

Tanulságok, tapasztalatok, magyarázat...

A programunk úgy kezdődik h létrehozunk egy Exor osztályt. Majd létrehozunk egy stringet, amelyben a kulcsot tároljuk és nyitunk két csatornát egy bejövőt és egy kimenőt az olvasás íráshoz. A throws a hibakezeléshez kell ha nem sikerül a beolvasás vagy kiiratás akkor hibát dob. Ezután definíálunk egy byte változót a buffernek és egy kulcs indexet, amely a kulcs első karakterére hivatkozik kezdetben és egy olvasott bajtokat, amely a beomenetről beolvasott szöveg hosszával egyenlő. Aztán a while-al olvassuk be a szöveget és letároljuk a bufferben közben a méretét az olvasott bajtokba. Aztán a forban titkosítunk a maradékos osztás azért szükséges, hogy ha a szöveg hosszabb mint a kulcs akkor a kulcs index ismét 0-tól kezdődjön mivel karakterenként történik a titkosítás. Majd a végén a write-al a megadott kimenetre írunk.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:<https://github.com/Savitar97/Prog1/tree/master/ex01>

Tanulságok, tapasztalatok, magyarázat...

Az exor törés az exor visszafejtése. Ez olyan mint a brutal force, addig próbálhatjuk a kulcs kombinációkat, amíg vissza nem kapjuk a szöveget.

Annak meghatározásához, hogy jó -e a szöveg most két függvényt írtunk egyik szempont az átlagos szóhosszak figyelembe vétele a másik pedig, a mondatokban gyakran előforduló szavak.

Az átlagos szóhosszt úgy kapjuk, hogy megszámoljuk hány darab space van a szövegben, majd a bemenő szöveg hosszát elosztjuk a szóközök számával.

Az exorban ugyan azt csináljuk mint a titkosításnál csak most vissza fejtjük a titkos szöveget.

Az exortörésben meghívjük az exor eljárást majd az exorozott szöveget átadjuk a tiszta lehetnek vizsgálatra, ha passzol akkor majd a brutal force-s forban az if igaz lesz és kiirja a terminálra a kulcsot és a megfejtett szöveget.

A mainbe szintén definiáljuk a kulcsot és a titkos szöveget,a p-titkossal megkapjuk a szöveg méretét. A while-ban hívjuk be a titkos szöveget. A while-t követő forban pedig minden 0-ázzuk a bufferben a maradék helyet. Ezután jönnek a kulcspróbálgatásos for ciklusok, amelyek a törést végzik, itt párhuzamosítva a gyorsabb működés érdekében.Ha nem állt le a for akkor újra exorozunk,így nincs szükség újabb meg újabb bufferre.

Így néz ki a program futás közben:

A screenshot of a terminal window titled "Terminál" (Terminal) at the top. The window contains several tabs, each showing command-line history. The tabs are labeled with file paths such as "/nemesis/nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/bevprop", "/nemesis/nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/bevprop", and so on. The terminal itself shows a series of commands being entered and executed, primarily involving file operations like copying and pasting text from a clipboard (indicated by 'ctrl + v'). The background of the terminal window features a dark theme with light-colored text.

4.3. ábra. Exortörés

4.5. Neurális OR, AND és EXOR kapu

**Tutor**

Ebben a feladatban tutoráltam Molnár Antalt.

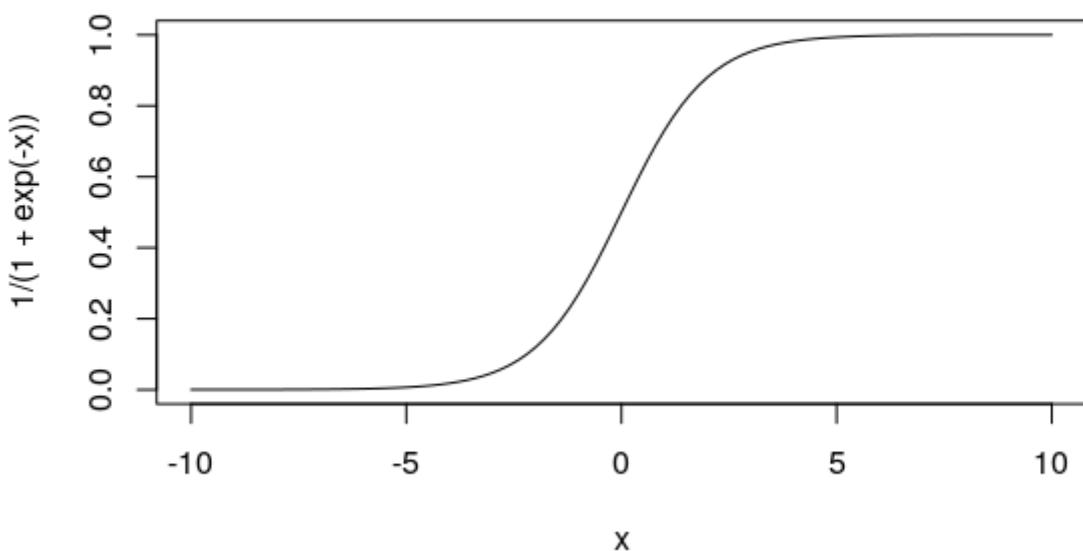
Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

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

Tanulságok, tapasztalatok, magyarázat...

A program futására az R-hez hozzá kell adnunk a neuralnetwork library-t. Ezután megadjuk neki az első mintát, ami alapján majd tanulni fog a program, tehát megoldunk egy minta feladatot. Tehát a program előre tudja mi lesz a bemenet és a kimenet. A program automatikusan választ súlyokat majd az értéket szorozzuk ezzel a súlyjal és összeadjuk őket és hozzáadja az eltolás mértékét. Majd behelyettesít egy $1/(1+\exp(-x))$ függvénybe. A kapott eredmény minden 0 és 1 közé fog esni. A neural net függvénynél elsőnek megadjuk, hogy milyen értéket kell kapnunk ez az OR-nak az értéke Ha több kimeneti értéket számolunk akkor +t használunk a felsoroláshoz, majd a ~-al adjuk meg, hogy miből kell ezt az eredményt kapnia, azaz a bemenetet. Következőnek megadjuk, hogy honnan vegye a bemenő adatokat. (Pl.: or.data, orand.data)

Ha növeljük a rejtett neuronok számát akkor pontosodik az érték és kevesebb lépésből képes meghatározni az eredményt az az több mintát készít. A stepmax meghatározza a maximum lépések számát. A threshhold a küszöbüfüggvény, ez amolyan leállási kritérium. Majd a compute kiadásával már számol a megtanult módon itt ellenőrizhetjük, hogy megtanulta-e a program a számítást. Lényegében a program próbál olyan értékpárokat találni súlynak és eltolási értéknek, amivel egy megközelítőleg hasonló értéket kap mint a mintában neki végeredményként megadott adat.



4.4. ábra. Signum függvény

4.6. Hiba-visszaterjesztéses perceptron

C++

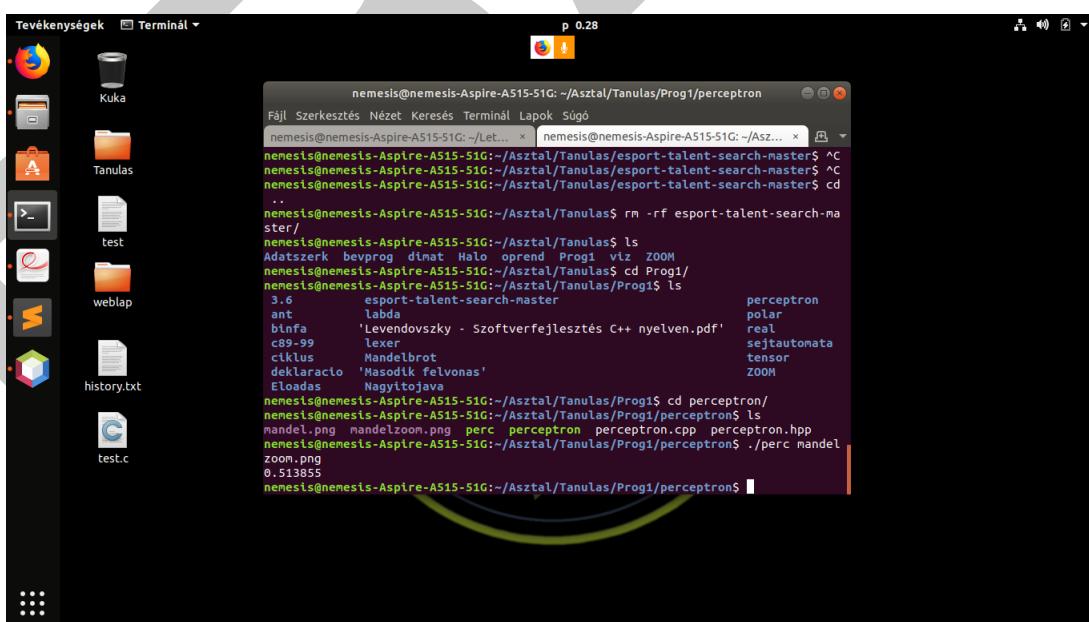
Megoldás video:

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

<https://github.com/Savitar97/Prog1/tree/master/perceptron>

Tanulságok, tapasztalatok, magyarázat...

A hiba-visszaterjesztéses perceptron(back-propagation) a rejtett rétegekben fellépő hibákat is a tudtunkra hozza. Vagyis ami a színfalak mögött történik. Ugyanis a végeredmény kiszámításánál létrejövő hibákért már a rejtett rétegen kialakult hibák a felelősek. Az ilyen visszaterjesztési perceptronokat csak olyan neurális hálóknál lehet alkalmazni ahol van hidden réteg. A visszaterjesztés a legutolsó rejtett neurontól kezdődik ellentétesen mint az alap számolás, tehát a kimenet előtti neurontól. A visszaterjesztés módszer lényege, hogy frissített súlyjal megbecsüljük, hogy az előző neuron mennyire hibázott a kívánt értéktől ez jó iránymutatást adhat a súlyok javításán. A program bekér egy képet. Majd a size-ban definiáljuk a kép méretét (szélesség*magasság). Majd a for ciklusban végig megyünk a kép pixelein. Példányosítjuk a perceptron osztályunkat, amely majd annyi neurális szintet képez ahány bemenet van, és annyi az argumentumok amiket kap generálja az egyes rétegeken a neuronok számát mint az R es példánál ha a hidden=c(2,3,2)-t használtuk például akkor azt jelentette hogy az első rétegen 2, a 2.-on 3 a 3.on megint 2 neuron legyen. Az utolsó érték azért 1 mert végül egy neuronon kell összekapcsolni minden értéket, amely majd a kimenethez csatlakozik. A perceptron a szigmoid függvényt használja $1.0 / (1.0 + \exp(-x))$. Az unitsba tároljuk majd le a súlyjal megszorzott értékünket. Amelyről majd a sigmoid megmondja, hogy jó -e vagy sem. A learning eljárásban történik a visszaterjesztés. Annak kiszámítása, hogy mekkora volt az eltérés úgy történik, hogy a sigmoid-al kiszámolt értéket kivonjuk az 1.0-ból az az a felső határól majd frissítsük a súlyokat és újra megnézzük. A perceptron kimenete egyetlen egy érték lesz, amely egy pixel, a visszakapott érték pedig egy 1 és 0 közé eső szám.



4.5. ábra. Perceptron

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

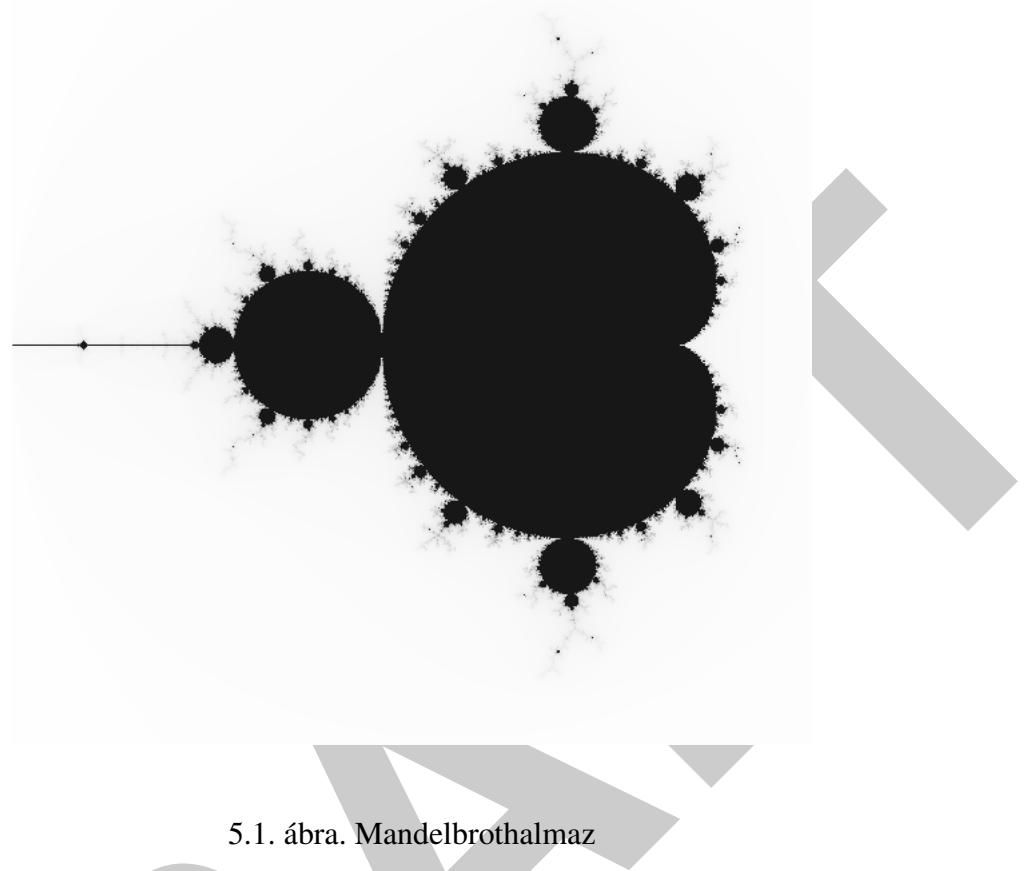
Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/mandelcomplex>

A program elején definiáljuk a kép méretét és az iterációs határt mivel végtelen számra nem tudjuk megnézni ezért kell valamilyen korlát.

A mainben most használjuk az argc és argv-t, ez csak azért kell, hogy megadhassuk hogy milyen néven mentse el a kimenetet. Ha nem adunk meg fájl nevet az első if fog hibaüzenetet dobni nekünk.

Ezután létrehozunk egy 2 dimenziós tömböt a kép méreteivel. Ezután a mandel függvénynek átadjuk ezt a tömböt. Itt vannak a futási időhöz való számítást segítő változók. De ami lényeg az a számoláshoz tartozó változók és, hogy az adott komplex szám a halmaznak eleme -e, ez akkor lehetséges ha a z kisebb mint 2 vagy elértek a 255. iterációt. Majd feltöltjük a tömböt. Majd létrehozunk egy új képet kép néven és pixelenként bezárjuk és ami benne van a halmazból elem azon a helyen a képkocka színét átszínezzük. Majd a write(argv[1])-el a megadott fájlnévvel készítünk egy képet. Az így kapott ábra a mandelbrot halmaz grafikus megjelenítése, amely egy fraktál az az egy végtelenül komplex alakzat lesz. Vagyis minél jobban rá nagytunk ismétlődésként megfog jelenni ez az ábra.



5.1. ábra. Mandelbrothalmaz

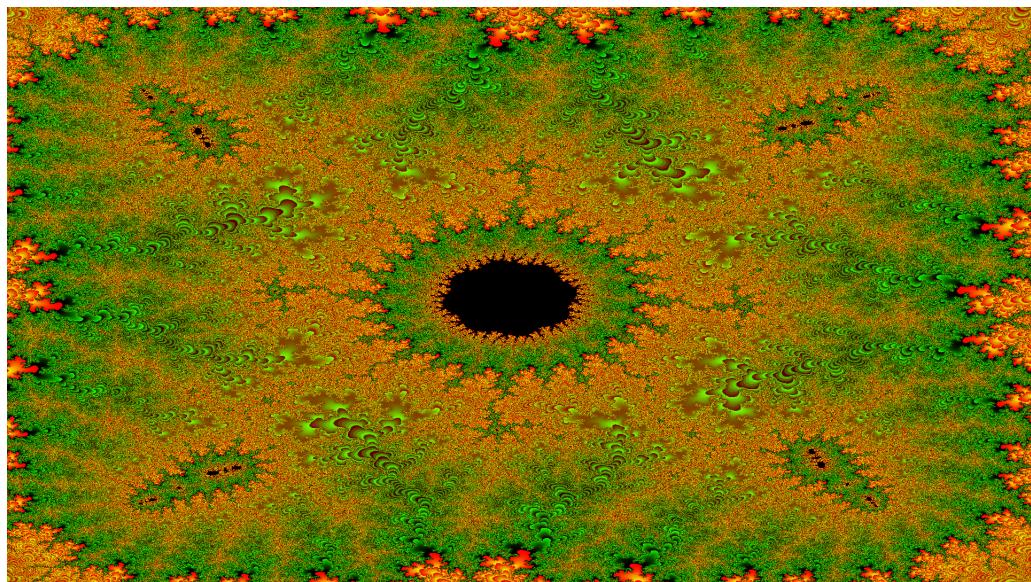
5.2. A Mandelbrot halmaz a std::complex osztálytal

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/mandelcomplex>

Ez a program ugyan az mint az előző csak most a complex osztályt fogjuk használni. Elsőként felvesszük a kezdetleges változókat vagyis, hogy mekkora legyen majd a kép mérete, hányszor fogunk iterálni és az intervallumot amin ábrázolni szeretnénk. Ezeket majd konzolról kérjük be de inicializáljuk, majd az if megvizsgálja megvan -e a kellő bemeneti adat, ha nincs akkor hibaüzenetet dob a program használtára.

Ezután létrehozunk egy üres képet a mérettel és a szélességgel. Ezután a dx és dy al megadjuk, hogy minden egyes lépéssel mennyit megyünk előre azaz a lépésközöket. A következő for ciklussal végig megyünk a képnek a képpontjain. Ezután kiszámoljuk C-nek a valós és imaginárius részét és ezeket átadjuk a komplex C számnak. Majd létrehozunk egy z_n komplex számot és inicializáljuk. Ezután jön egy while ciklus ami addig megy amíg a z_n abszolút értékben kisebb mint 4 vagy pedig elérte az iterációs határt. Majd a while ciklus törzsében kiszámoljuk a z_n értékének és növeljük az iterációs határt. Majd ha kilép a while ciklusból a képnek az adott sorában és oszlopában lévő pixel színét átállítjuk. Az int százalékkal a feldolgozás állását közvetítjük a consolra. Végén kiiratjuk a képet a megadott fájlba.



5.2. ábra. Mandelbrot halmaz komplex osztályal

5.3. Biomorfok

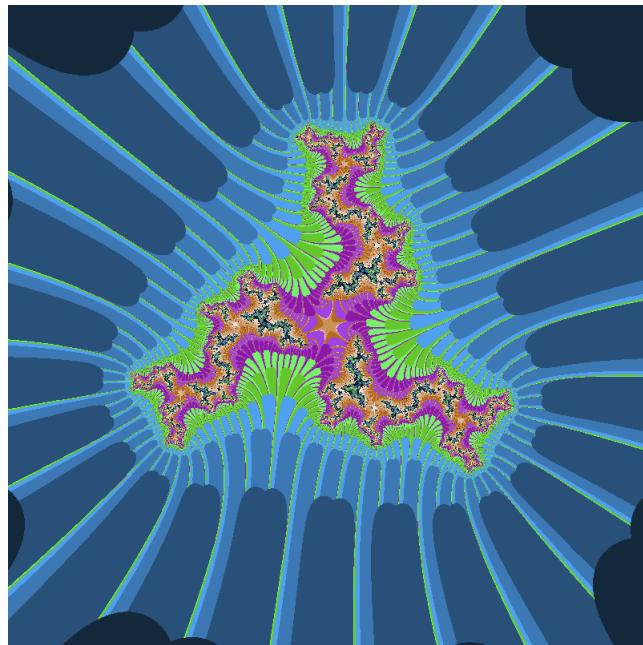
Megoldás videó: <https://youtu.be/IJMbgRzY76E>

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

Tanulságok, tapasztalatok, magyarázat...

A biomorpos program abban különbözik az előzőtől, hogy most több argumentumot tudunk megadni tehát adhatunk kezdőértéket egy komplexszámnak cc-nek, amelyet majd z-hez minden hozzáadunk. A bekért karaktereket az atoi intté alakítja míg az atof lebegőpontos számmá. Az előzőhöz képest ahol egy while-t futtattam, hogy eldöntsem mely elemeket tartoznak a halmazba és ez a feltétel az volt hogy abszolút értékbe a z komplex szám kisebb mint négy, vagy elértek -e az iterációs határt. Ez a feltétel most annyiban változott, hogy R ben megadhatjuk hogy mekkora érték felett kell legyen a z valós vagy z imaginárius részének és csak akkor növeljük az iterációk számát(ez egy küszöbérték), vagyis az iteráció azt az értéket fogja megkapni a 0- és az iterációs határ között, amelyre még teljesül a feltétel. Emellett még lényeges változtatás, hogy eddig csak az iterációt osztottuk maradékosan a kép színeihez, de mostmár konstansokkal szorozzuk meg a különböző színeket előállító képletet. Ez színesebb képet fog eredményezni és a több argumentum nagyobb szabadságot nyújt a felhasználónak, hogy különböző képeket alkossan. A biomorpos képek az egysejtűekre hasonlítanak, elég érdekes formákat lehet alkotni a program segítségével.

Az általam létrehozott biomorf amely szerintem egész jól néz ki:



5.3. ábra. Biomorf

5.4. A Mandelbrot halmaz CUDA megvalósítása

**Tutorált**

Ebben a feladatban tutorált Molnár Antal.

Megoldás videó:

Megoldás forrása: https://gitlab.com/Savitar97/bhax/tree/master/attention_raising/CUDA
<https://www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf>

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

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

#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
```

```
// Végigzongorázza a CUDA a szélesség x magasság rácsot:  
// most eppen a j. sor k. oszlopban vagyunk  
  
// számítás adatai  
float a = -2.0, b = .7, c = -1.35, d = 1.35;  
int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;  
  
// a számítás  
float dx = (b - a) / szelesseg;  
float dy = (d - c) / magassag;  
float reC, imC, reZ, imZ, ujreZ, ujimZ;  
// Hány iterációt csináltunk?  
int iteracio = 0;  
  
// c = (reC, imC) a rács csomópontjainak  
// megfelelő komplex szám  
reC = a + k * dx;  
imC = d - j * dy;  
// z_0 = 0 = (reZ, imZ)  
reZ = 0.0;  
imZ = 0.0;  
iteracio = 0;  
// z_{n+1} = z_n * z_n + c iterációk  
// számítása, amíg |z_n| < 2 vagy még  
// nem értük el a 255 iterációt, ha  
// viszont elértek, akkor úgy vesszük,  
// hogy a kiinduláci c komplex számra  
// az iteráció konvergens, azaz a c a  
// Mandelbrot halmaz eleme  
while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)  
{  
    // z_{n+1} = z_n * z_n + c  
    ujreZ = reZ * reZ - imZ * imZ + reC;  
    ujimZ = 2 * reZ * imZ + imC;  
    reZ = ujreZ;  
    imZ = ujimZ;  
  
    ++iteracio;  
  
}  
return iteracio;  
}  
  
/*  
__global__ void  
mandelkernel (int *kepadat)  
{  
  
    int j = blockIdx.x;
```

```
int k = blockIdx.y;

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

}

*/



__global__ void
mandelkernel (int *kepadat)
{

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

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

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

}

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

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

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

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

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

}

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

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

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

int kepadat [MERET] [MERET];

cudamandel (kepadat);

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

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

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

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

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

}
```

Lényegében azt kell megfigyelnünk, hogy mennyivel gyorsabban dolgoznak a cuda magok a processzornál. A processzornál ugyebár a képpontokat szekvenciálisan számoljuk tehát a processzor egyessével számolja a képpontokat. Ez a cudánál egyszerre történik most az az minden egyes képpontot egy szál fog számolni a számolás egyszerre történik mint a párhuzamos programoknál. Ez hatalmas sebességbeli növekedést eredményez konkrétan egy pillanat alatt végez a program ha elég erős a videókártyánk, mivel itt már a gpu számol nem a cpu. Ez azért lehetséges mivel a GPU-ban sokkal több szál van mint a processzorban. A programban a cudaMallocal foglalunk helyet a GPU-n dinamikusan történik, amely a kép méretei alapján és az int mérete alapján történik. A dim3 adja meg a blokkok és szálak méretét, ezzel 3 dimenzióban szoktuk

megadni, de itt most csak 2-t használunk. Külön figyelmet kíván a:

```
<<<
    és
>>>
    operátor.
```

Ezekkel használjuk a kernelt azt hogy hány szálat szeretnénk futtatni és hány blokkra osztjuk fel ezeket a szálakat azaz ez a párhuzamos kernelünk, a grid a legfelsőbb szintje az az ez olyan mint egy keret(rács), amely összefogja a blokkokat és szálakat, ezen kívül a tgrid a blokk méretét számolja ki. Ezeket adjuk át a kernelnek. A global típusú függvények az eszközön futnak és a processzoron futó program hívja meg, míg a device kódokat a processzor nem tudja meghívni ezek csak a videókártyán használhatók. A cudaMemcpyvel tudunk adatokat mozgatni a videókártyán számolt adat és a processzor között. Mivel dinamikusan forgaltunk memóriát itt is fel kell szabadítani ezt a cudaFree-vel érjük el.

```
nemesis@nemesis-Aspire-A515-51G:~/repo/bhax/attention_raising/CUDA$ ./mandel man
del.png
mandel.png mentve
30
0.306424 sec
nemesis@nemesis-Aspire-A515-51G:~/repo/bhax/attention_raising/CUDA$ ./mandelpngt
mandel.png
1936
19.3534 sec
mandel.png mentve
```

5.4. ábra. Mandel CUDA

Először cudával lett futtatva majd azután C-ben processzorral.

5.5. Mandelbrot nagyító és utazó C++ nyelven



Tutor

Ebben a feladatban tutoráltam Kun-Limberger Anettet és Duszka Ákos Attilát.

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

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/Nagyito>

Megoldás videó:

Újra a mandelbrot halmazzal fogunk foglalkozni, de most felhasználjuk a qt-t hogy létrehozzunk egy grafikus interfacet. A programban a tartományt ugyan úgy az a, b, c és d változó határozza meg A képlet most is ugyan az mint a legelsőnél, amivel számoljuk az iterációt: $z_{(n+1)} = z_n * z_n + c$, ez a számolás a

fraksal.cpp-ben van. A számításokat soronként küldjük vissza a frakablaknak, amely majd elvégzi a színezést. A változó deklarációja és inicializálása a számításokhoz a frakablak.h-ban található. A frakablak.cpp-ben definiáljuk, hogy mit csináljon a program az egérmozgására és, hogyha kijelölünk egy területet az egérrel akkor arra a területre nagyítson rá. Tehát a mousepressevent letárolja a kattintásunk koordinátáit, míg a mousemove a szélességet és a magasságot tehet, hogy az adott pontból mekkora területet jelöltünk ki. Majd a felengedéskor újra számol és rá zoomol a területre. Az N gomb lenyomásával változtathatjuk az iterációs határt amivel változik a kép részletessége is, ugyanis az N gomb lenyomására az iterációs határ két-szereződik emiatt nagyobb lesz a részletessége a képnél, így amikor jobban rá zoomolunk akkor nem csak olyan mintha egy sima képre nagyítanánk mivel ha az iterációs határ megnő akkor a mandelbrothalmaznak egyre több eleme lesz emiatt változik a kép is. Persze minden számolás után update-eljük az osztá�ban lévő értékeket. Az újra számoláshoz készítünk mindenkor egy új FrakSzal-t, a régi mandelbrot halmazt töröljük és a frissített iterációshatárral számolunk.



5.5. ábra. Mandelbrot nagyító

5.6. Mandelbrot nagyító és utazó Java nyelven



Tutor

Ebben a feladatban tutoráltam Kun-Limberger Anetett és Duszka Ákos Attilát.

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/javacnagyito>

A mandelbrot nagyító javában. Mivel a mandelbrot javás programunkhoz készítünk nagyítót, ezért ez lesz a szülő osztály. A származtatást az extends kifejezéssel érjük el. Létrehozunk két változót private jogosultságokkal. Az elsőben azt a pontot tároljuk el, ahova kattintunk majd az egérrel tehát a kezdő pont koordinátáit. A másik az egér jelenlegi pozícióját fogja tárolni. Ez a terület kijelölésénél lesz majd fontos, hogy mekkora területet jelölünk ki. Majd meghívjuk az osztály konstrukturát public jogosultságokkal és 6 argumentummal, amelyből az első 4 argumentum a tartományt adja meg. Majd a tömbnek a méretét, amelyben a halmaz szerepel. Majd a nagyítási pontosságot, a képen ez adja azt, hogy minél részletesebb legyen a többszörös nagyításnál. Ezt követően a superrel meghívjuk az ōs osztály konstrukturát az argumentumaival. A supert többféle képpen lehet használni. Képesek vagyunk vele azonnal a szülő osztály konstrukturát argumentumokkal vagy nélküle, esetleg a változóit, függvényeit meghívni. A settilevel adjuk meg az ablaknak a nevét. Majd a mouseListenerrel figyeljük az egér vezérlést. Ha kattintunk akkor letároljuk a koordinátáit az egérnek. Felengedéskor a megadott tartományt újra számoljuk. És létrehozunk egy új objektumot a halmaznak, amelyet kirajzolunk. A nagyítandó területet úgy számoljuk, hogy a jelenlegi egér pozícióból, azaz a négyssög jobb alsó sarkából kivonjuk a bal felső sarok koordinátáit, ez lesz az mx és my. Az s lenyomásával pillanatfelvételt készíthetünk a fájl nevében megjelenítsük, hogy milyen tartományi értéknél készítettük a képet. Az n gombbal változtatunk az iterációs értékeken minden növeljük a pontosságot 256 al. Az m-el ugyan úgy növeljük a pontosságot de itt 10*256-al pontosítunk. Majd jön a kép kirajzolása, ha számítást végezünk akkor egy vörös vonallal jelezzük az állapotát ezt a drawline éri el. A számításhoz a kijelölt területet zölddel jelezzük, ezt a drawrect éri el. Végül a mainbe létrehozunk egy mandelbrot halmaz példányt és már futtathatjuk is a programunkat.

A fordítás a következővel hajtjuk végre: **javac MandelbrotHalmazNagyító.java**.

A futtatást pedig, így hajtjuk végre: **java MandelbrotHalmazNagyító**.

```
a=-0.5381235707889077  
b=-0.5381235697982686  
c=0.5349807456063211  
d=0.5349807498247798  
n=2559
```

5.6. ábra. Java Mandelbrot nagyító

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/polar>

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

```
public class PolarGen
{
    boolean nincsTarolt = true;
    double tarolt;

    public PolarGen ()
    {
        nincsTarolt = true;
    }

    public double kovetkezo ()
    {
        if (nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do
                w = Math.random();
            while (w == 0);
            u1 = Math.random();
            u2 = Math.random();
            v1 = Math.random();
            v2 = Math.random();
        }
        return w;
    }
}
```

```
        {
            u1 = Math.random();
            u2 = Math.random();
            v1 = 2 * u1 - 1;
            v2 = 2 * u2 - 1;
            w = v1 * v1 + v2 * v2;
        }
        while (w > 1);

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

        tarolt = r * v2;
        nincsTarolt = !nincsTarolt;

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

public static void main (String[] args)
{
    PolarGen pg = new PolarGen();

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

}
```

Elsőként létrehozzuk a polargen osztályt. Amelyben deklarálunk egy konstruktort és egy destruktort. A konstruktorban egy a private részben tárolt nincstarolt nevű bool típusú változót inicializálunk és meghívjuk a randomot. És létrehozunk még egy következő nevű double függvényt.

A private részben két változót deklarálunk egyik a tarolt a másik a nincstarolt. Ezután a PolarGen névterben lévő kovetkezo függvényt írjuk le, ha a nincstarolt=true akkor létrehozunk 5 változót az u1 és az u2 random értéket vesznek fel és ezeket az értékeket felhasználjuk a v1 és v2 értékek kiszámolásánál. Majd a w a v1 és v2 értékek négyzetének az összege. Ez az érték addig változik, amíg nem lesz kisebb az értéke w-nek 1-nél, a do while miatt legalább 1x lefut a ciklus. Majd egy r változóba a gyököt veszi a -2*log(w)-nek, amelyet eloszt w-vel. Majd ezt az értéket felhasználja a tarolt-nál, amely a private-ban lévő változóba teszi az értéket az r*v2-t és a nincstarolat negáljuk tehát az értéke true helyett false lesz. A visszatérési értéke a

fv-nek r^*v_1 lesz.

Ha már van tárolt érték akkor pedig azzal tér vissza. Majd a main-be meghívjuk az osztályunkat pg néven és a forral létrehozunk 10 mintapéldányát az osztályak. Ez az osztály a random számgenerálás osztálya. Tehát végülis 10 véletlen számot fog generálni nekünk a program a Java-ban ez az osztály ugyan így szerepel a forrásfálok közt Random.java néven találjuk a mappában. Az objektum orientáltságnak 3 lényeges pontja van az egyik, hogy az összetartozó adatok képekes legyenek egy zárt adategységet alkotni. Az adatrejtés segítségével kezeljük a jogosultságokat vagyis, hogy az előtt említett egységből, ki milyen adatokat képes elérni(private,protected,public). Emellett a harmadik fontos tulajdonság az öröklődés. Öröklődés során az "utód" örökli a szülője összes tulajdonságát, de lehetőségünk van még plusz tulajdonságokat adni neki.

```

Tevékenységek Terminál k 9.54
Megnyitás Random.java
Random.java
* independent values at the cost of only one call to {@code StrictMath.log}
* and one call to {@code StrictMath.sqrt}.
*
* @return the next pseudorandom, Gaussian ("normally") distributed
*         {@code double} value with mean {@code 0.0} and
*         standard deviation {@code 1.0} from this random number
*         generator's sequence
*/
synchronized public double nextGaussian() {
    // See Knuth, ACP, Section 3.4.1 Algorithm C.
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1; // between -1 and 1
            v2 = 2 * nextDouble() - 1; // between -1 and 1
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}

/**
 * Serializable fields for Random.
 *
 * @serialField seed long
 *           seed for random computations
 * @serialField nextNextGaussian double
 *           next Gaussian to be returned
 * @serialField haveNextNextGaussian boolean
 *           nextNextGaussian is valid
*/

```

6.1. ábra. Polargen random

6.2. LZW

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

Megoldás video:

Megoldás forrása: <https://github.com/Savitar97/Prog1/blob/master/binfac/bifa.c>

Ez a bináris fa, a bináris fák egy speciális típusa ugyanis LZW algoritmust használ, ami egy tömörítő algoritmus.

```

#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 kiir2 (BINFA_PTR elem);
extern void kiirl (BINFA_PTR elem);
extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);

int
main (int argc, char *argv[])
{
    char b;
    if (argv[1][0]!='-')
        perror("használat ./binfa -kapcsolo");
        return -1;
}
    if(argc!=2)
    {
        perror("nincs kapcsolo ./binfa -kapcsolo");
        return -2;
    }

char kapcsolo=argv[1][1];
BINFA_PTR gyoker = uj_elem ();
gyoker->ertek = '/';
BINFA_PTR fa = gyoker;

while (read (0, (void *) &b, 1))
{
    write (1, &b, 1);
    if (b == '0')
    {
        if (fa->bal_nulla == NULL)
```

```
{  
    fa->bal_nulla = uj_elem ();  
    fa->bal_nulla->ertek = 0;  
    fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;  
    fa = gyoker;  
}  
else  
{  
    fa = fa->bal_nulla;  
}  
}  
else  
{  
    if (fa->jobb_egy == NULL)  
    {  
        fa->jobb_egy = uj_elem ();  
        fa->jobb_egy->ertek = 1;  
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;  
        fa = gyoker;  
    }  
    else  
{  
        fa = fa->jobb_egy;  
    }  
}  
}  
switch(kapcsolo)  
{  
    case 'i':printf ("Inorder\n");  
    kiir (gyoker);  
    break;  
    case 'p':printf ("Preorder\n");  
    kiirl (gyoker);  
    break;  
    case 'o':printf ("Postorder\n");  
    kiir2 (gyoker);  
    break;  
    default: printf("%s\n", "Hibás a kapcsolo.");  
    break;  
}  
  
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
kiir1 (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↔
                ,
                melyseg);
        kiir (elem->jobb_egy);

        kiir (elem->bal nulla);
        --melyseg;
    }
}
void
kiir2 (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;
}
}

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

Kezdésképp létrehozunk egy struktúrát binfa néven, amely tartalmaz egy értéket és 2 struktúra típusú mutatót. A BINFA és *BINFA_PTR a typedefnek a kulcsszavai mindenkor binfa típusú lesz.

Ezután meghívunk egy BINFA_PTR típusú függvényt, amely visszaadja majd értékül a példányosított p eredményét, amely egy sikeres dinamikus memória foglalás az új elemnek.

A következő két eljárásnál az externnel jelezzük a fordítónak, hogy majd a program végén fogjuk deklarálni őket. A kiirba ha van elemünk a fában akkor növeljük a mélységet ha a mélység nagyobb mint a maximum mélység akkor ezt egyenlővé tesszük majd kiiratjuk elsőként a jobbos elemet majd a balost és visszacsökkentjük a mélységet. A szabaditnál a fölöslegesen lefoglalt tárhelyet szabadítjuk fel.

Létrehozunk egy karater típusú változót, amely majd azt mutatja milyen elem megy be a fába. Ezután deklarálunk és inicializáljuk a fának a gyökerét majd a fa mutatót ráállítjuk a gyökérre ezután majd a while ciklusban pakolgatjuk az elemeket jobbra vagy balra attól függően hogy milyen elem megy be a fába, ha nincs több érték az adott ágon akkor visszaállítjuk a mutatót a gyökérre.

6.3. Fabejárás

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

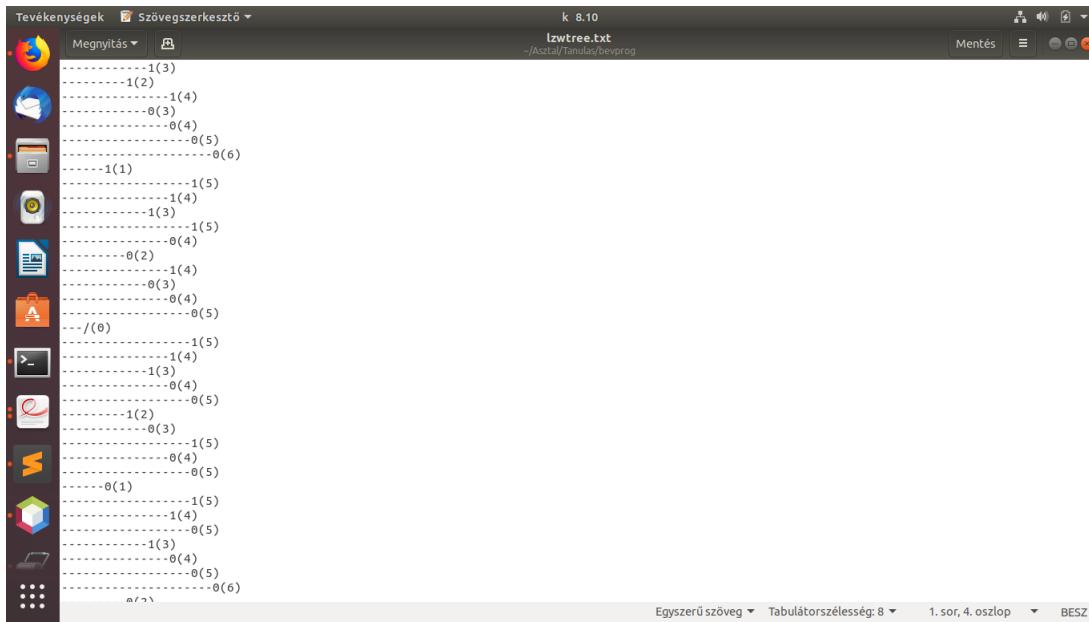
Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/blob/master/binfac/binfa.c>

A fabejárás 3 különböző féle képpen történhet. Lehet inorder amikor a baloldalt veszi először majd a gyökeret irassa ki és csak aztán tér át a jobb oldalra. A második lehetőség a preorder itt a gyökeret irassa ki először, majd a bal oldalt és végül a jobb oldalt. A legutolsó variáció a postorder, amikor elsőként irassuk ki a bal oldalt majd a jobb oldalt és csak végül a gyökeret. Inorderrel a csomópontok mindenkor középre kerülnek.

A lényeg hogy mikor irassuk ki az egyes és nullás gyermeket, ha a forciklus előtt akkor postorder, ha a forciklus után akkor preorder, ha pedig az egyest a forciklus előtt és a nullást a forciklus után akkor inorder bejárással irassuk ki a bináris fánkat.

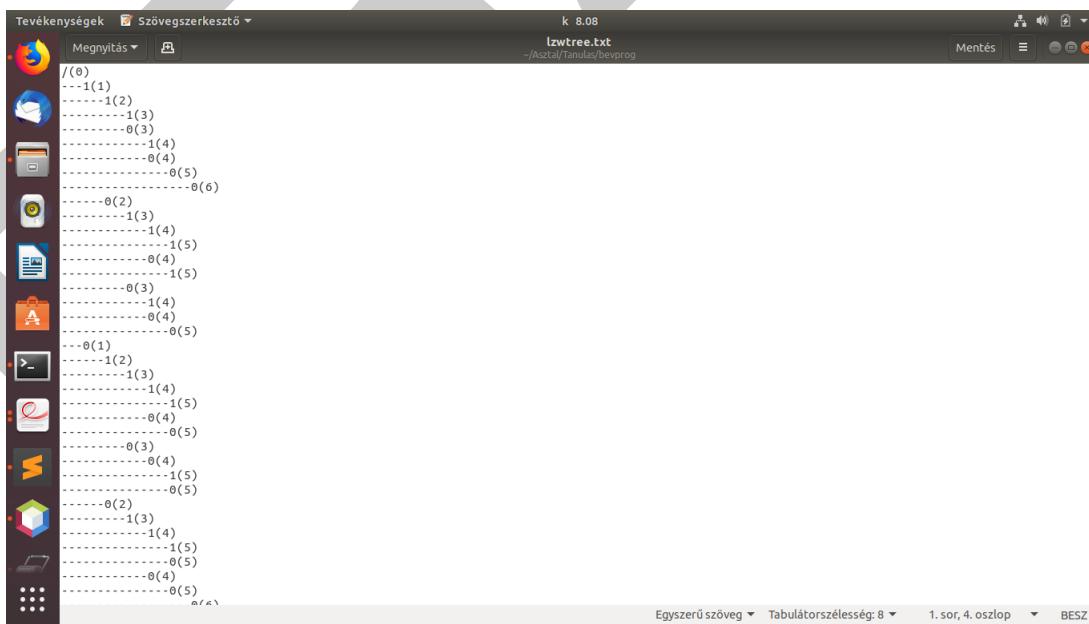
Inorder bejárás:



The screenshot shows a text editor window titled "Szövegszerkesztő" (Text Editor) with file "lzwtree.txt" open. The content of the file is a binary tree structure represented by nodes and their children. The traversal follows the inorder sequence: left child, root node, right child. The tree has 12 nodes labeled 0(1) through 0(6) and 1(1) through 1(6). The editor interface includes a toolbar, menu bar, status bar at the bottom, and a scroll bar on the right.

6.2. ábra. Inorder bejárás

Preorder bejárás:



The screenshot shows a text editor window titled "Szövegszerkesztő" (Text Editor) with file "lzwtree.txt" open. The content of the file is the same binary tree structure as in the previous screenshot, but the traversal follows the preorder sequence: root node, left child, right child. The tree has 12 nodes labeled 0(1) through 0(6) and 1(1) through 1(6). The editor interface includes a toolbar, menu bar, status bar at the bottom, and a scroll bar on the right.

6.3. ábra. Preorder bejárás

Postorder bejárás:

```
Tree traversal output:
0(5)
--0(2)
---1(1)
----0(5)
-----0(4)
-----0(5)
-----0(4)
-----0(3)
-----1(2)
-----1(5)
-----0(5)
-----1(4)
-----0(6)
-----0(5)
-----0(4)
-----1(3)
-----1(4)
-----0(7)
-----1(5)
-----0(7)
-----0(6)
-----0(5)
-----0(4)
-----0(3)
-----0(2)
---0(1)
---0(0)
depth = 7
mean = 5.05882
var = 1.02899
nulas gyermek: 26
egyes gyermek: 20
értékű
```

6.4. ábra. Postorder bejárás

6.4. Tag a gyökér



Tutor

Ebben a feladatban tutoráltam Ádám Petrárt.

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

Megoldás video:

Megoldás forrása: <https://github.com/Savitar97/Bevprog/tree/master/vedes>

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

class LZWBinFa
```

```
{  
public:  
  
    LZWBInFa ()  
    {  
        gyoker = new Csomopont();  
        fa=gyoker;  
    }  
    ~LZWBInFa ()  
    {  
        szabadit (gyoker->egyesGyermek());  
        szabadit (gyoker->>nullasGyermek());  
        delete gyoker;  
    }  
  
    void operator<< (char b)  
    {  
  
        if (b == '0')  
        {  
  
            if (!fa->nullasGyermek()) // ha nincs, hát akkor csinálunk  
            {  
  
                Csomopont *uj = new Csomopont ('0');  
  
                fa->ujNullasGyermek(uj);  
  
                fa = gyoker;  
            }  
            else  
            {  
  
                fa = fa->nullasGyermek();  
            }  
        }  
  
        else  
        {  
            if (!fa->egyesGyermek())  
            {  
                Csomopont *uj = new Csomopont ('1');  
                fa->ujEgyesGyermek(uj);  
                fa = gyoker;  
            }  
            else  
            {  
                fa = fa->egyesGyermek();  
            }  
        }  
    }  
}
```

```
}

void kiir (void)
{
    melyseg = 0;

    kiir (gyoker, std::cout);
}

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

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

private:
    class Csomopont
    {
public:

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

    Csomopont *nullasGyermekek () const
    {
        return balNulla;
    }

    Csomopont *egyesGyermekek () const
    {
        return jobbEgy;
    }

    void ujNullasGyermekek (Csonopont * gy)
    {
```

```
        balNulla = gy;
    }

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

char getBetu () const
{
    return betu;
}

private:

char betu;

Csmopont *balNulla;
Csmopont *jobbEgy;

Csmopont (const Csmopont &);
Csmopont & operator= (const Csmopont &);

};

Csmopont *fa;

int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;

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

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

        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::left
            endl;
        kiir (elem->>nullasGyermek (), os);
        --melyseg;
    }
}
void szabadit (Csmopont * elem)
```

```

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

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

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

};

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

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

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

    rszoras (gyoker);

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

```

```
else
    szoras = std::sqrt (szorasosszeg);

    return szoras;
}

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

        rmelyseg (elem->>nullasGyermek ());
        --melyseg;
    }
}

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

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

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

void
usage (void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;//f a ←
    fájlba írásért c a consoleba írásért
}

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

        if (argc != 5)
        {

            usage ();
            throw std::invalid_argument ("arg");
            return -1;
        }

        char *inFile = argv[1];

        if (argv[2][1] != 'o')
        {
            usage ();
            throw std::ios::failure ("Hibás bemenet");
            return -2;
        }

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

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

        std::fstream kiFile (argv[3], std::ios_base::out);
```

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

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

bool kommentben = false;

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

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

    if (kommentben)
        continue;

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

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

if (argv[4][0]=='f') {
    kiFile << binFa;

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

```
kiFile << "var = " << binFa.getSzoras () << std::endl;
}
else if(argv[4][0]=='c')
{std::cout<< binFa;

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

return 0;
}
catch (std::invalid_argument& e) {
    std::cout << "Hiba történt: ";
    std::cout << e.what() << std::endl;
}
catch (std::ios::failure& e) {
    std::cout << "Hiba történt: ";
    std::cout << e.what() << std::endl;
}
}
```

A C-s változattól abban különbözik, hogy használhatunk osztályokat. Létre is hozzuk az LZWBInFa osztályunkat majd deklarálunk egy konstruktort és egy destruktort. Majd túlterheljük az operátort a void operator-ban, amely paraméterül a char b-t kapja. Ezzel vizsgáljuk milyen elem megy be épp. Ha ez az elem 0 és a fának nincs 0 ás eleme akkor létrehozunk egyet neki. Ha van akkor ráállítjuk a fa mutatót. Ha ez az elem 1-es akkor hasonlóképpen működik. Majd jön a kiir eljárás, amely rekurzívan hívja meg magát. Argumentumként megkapja a gyökeret és azt, hogy mit kell kiirni ez az egyes gyermek és nullás gyermek lesz. Ezután az LZWBInFa osztály private részében létrehozunk egy Csomópont osztályt. A csomópont konstruktora argumentumként kapja meg inicializálva a gyökér karaktert és typedefeljük a betűt a balNullát és a jobbEgyet. Ezután jön a destruktora az osztálynak. Ezután jönnek a csomópontok gyermekinek vizsgálata, van -e nekik ha nincs akkor nullal tér vissza. Majd a két eljárás, amelynek mutatója átadja a címét, hogy hol legyen az egyes vagy nullás gyermek a megadott csomópontnak. A char get betűben pedig az értéket vizsgáljuk, hogy éppen 0 vagy 1 es jön. Majd a private részben deklaráljuk a mutatókat és a változókat és letiltjuk a másoló konstruktort. Ezután a csomópont osztályon kívül létrehozzuk a csomópont fa mutatót, amely minden az aktuális csomópont elemre mutat. Majd deklaráljuk a számításhoz szükséges függvények változóit és letiltjuk a binfának is a másolását. Létrehozzuk a kiir eljárást, a kiiratás csak akkor tud megtörténni ha van elem a fában, itt inorder kiiratás történik. Ezután a fölösleges nem használt részeket felszabadítjuk a szabadíttal. Majd van egy protected rész ahol kiemeljük, hogy a fának van egy kitüntetett tag csomópontja a /. Ezután az osztáyból kilépve a sima globális térbe létrehozunk egy usage eljárást, amelyel ha hibásan futtatnánk a programot segítséget nyújtunk a felhasználónak. A mainben a try catch hibakezelő eljárást alkalmazzuk. Ha nincs elég argumentum megadva akkor hibaüzenetet dobunk, ezután inicializálunk egy mutatót, amely a fájl nevére mutat. Majd vizsgáljuk, hogy a fájl név után a -o kapcsoló

jön -e, ha nem hibaüzenetet dobunk. Majd az fstreammel beolvassuk a fájlt, amelynek megadtuk a bemenő fájl nevének címét. Majd létrehozzuk a kifile-t, amely a fájlba írásért lesz felelős. Deklarálunk egy karakter változót és meghívjuk a LZWBInFa osztályt binfa néven. Majd indítunk egy while ciklust, amelyben felsoroljuk a kivételeket, hogy mit hagyjon figyelmen kívül a beolvasás. A következő forciklusban végig megyünk a 8 biten és ha egyes van akkor egyes kerül a tárba ha 0-ás akkor 0. A kifile-nak átadjuk a binfát, majd a mélység, átlag, szórást és végül bezárjuk a filestreamet.

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: [https://github.com/Savitar97/Prog1/blob/master/mozgato\(pointer2.cpp](https://github.com/Savitar97/Prog1/blob/master/mozgato(pointer2.cpp)

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

class LZWBInFa
{
public:
    LZWBInFa()
    {
        gyoker = new Csomopont();
        fa=gyoker;
    }
    ~LZWBInFa()
    {
        szabadit (gyoker->egyesGyermek ());
        szabadit (gyoker->nullasGyermek ());
        delete gyoker;
    }

    LZWBInFa ( LZWBInFa && regi )
    {
        gyoker = nullptr;
        *this = std::move(regi);
    }
}
```

```
LZWBinFa & operator= (LZWBinFa && regi){  
  
    std::swap(gyoker, regi.gyoker);  
  
    return *this;  
}  
  
void operator<< (char b)  
{  
  
    if (b == '0')  
    {  
  
        if (!fa->nullasGyermek ())  
        {  
            Csomopont *uj = new Csomopont ('0');  
            fa->ujNullasGyermek (uj);  
            fa = gyoker;  
        }  
        else  
        {  
  
            fa = fa->nullasGyermek ();  
        }  
    }  
    else  
    {  
  
        if (!fa->egyesGyermek ())  
        {  
            Csomopont *uj = new Csomopont ('1');  
            fa->ujEgyesGyermek (uj);  
            fa = gyoker;  
        }  
        else  
        {  
            fa = fa->egyesGyermek ();  
        }  
    }  
}  
  
void kiir (void)  
{  
  
    melyseg = 0;  
    kiir (gyoker, std::cout);  
}  
  
int getMelyseg (void);  
double getAtlag (void);  
double getSzoras (void);
```

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

private:
    class Csomopont
    {
public:

    Csomopont (char b = '/') : betu (b), balNulla (0), jobbEgy (0)
    {
    };
    ~Csomopont ()
    {
    };
    Csomopont (const Csomopont& elem) {

        betu = elem.getBetu();
        balNulla = new Csomopont;
        jobbEgy = new Csomopont;
        *balNulla= *(elem.nullasGyermek());
        *jobbEgy= *(elem.egyesGyermek());
    }

    Csomopont & operator= (const Csomopont& elem) {

        betu = elem.getBetu();
        Csomopont* ujBal = new Csomopont();
        *ujBal = *(elem.nullasGyermek());
        delete balNulla;
        balNulla = ujBal;
        Csomopont* ujJobb = new Csomopont();
        *ujJobb = *(elem.egyesGyermek());
        delete jobbEgy;
        jobbEgy = ujJobb;

        return *this;
    }

    Csomopont *nullasGyermek () const
    {
        return balNulla;
```

```
}

Csomopont *egyesGyermek () const
{
    return jobbEgy;
}

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

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

char getBetu () const
{
    return betu;
}

private:

char betu;
Csomopont *balNulla;
Csomopont *jobbEgy;

};

Csomopont *fa;
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
LZWBinFa (const LZWBinFa& binfa);

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

```
{  
    if (elem != NULL)  
    {  
        szabadit (elem->egyesGyermek ());  
        szabadit (elem->>nullasGyermek ());  
        delete elem;  
    }  
}  
  
protected:  
    Csomopont *gyoker;  
    int maxMelyseg;  
    double atlag, szoras;  
  
    void rmelyseg (Csmopont * elem);  
    void ratlag (Csmopont * elem);  
    void rszoras (Csmopont * elem);  
};  
  
int  
LZWBinFa::getMelyseg (void)  
{  
    melyseg = maxMelyseg = 0;  
    rmelyseg (gyoker);  
    return maxMelyseg - 1;  
}  
  
double  
LZWBinFa::getAtlag (void)  
{  
    melyseg = atlagosszeg = atlagdb = 0;  
    ratlag (gyoker);  
    atlag = ((double) atlagosszeg) / atlagdb;  
    return atlag;  
}  
  
double  
LZWBinFa::getSzoras (void)  
{  
    atlag = getAtlag ();  
    szorasosszeg = 0.0;  
    melyseg = atlagdb = 0;  
  
    rszoras (gyoker);  
  
    if (atlagdb - 1 > 0)  
        szoras = std::sqrt (szorasosszeg / (atlagdb - 1));  
    else  
        szoras = std::sqrt (szorasosszeg);  
}
```

```
    return szoras;
}

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

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

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

```
}

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

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

    char *inFile = *++argv;

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

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

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

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

    unsigned char b;
    LZWBinFa binFa,binFa2;

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

    bool kommentben = false;

    while (beFile.read ((char *) &b, sizeof (unsigned char)))
    {
```

```
if (b == 0x3e)
{
    // > karakter
    kommentben = true;
    continue;
}

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

if (kommentben)
continue;

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

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

}

kiFile << binFa;
kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;
kiFile << &binfa
binFa2=std::move(binFa);
kiFile<<"\n Mozgatás után binFa:"<< std::endl;
kiFile << binFa;
kiFile << "depth = " << binFa.getMelyseg () << std::endl;
kiFile << "mean = " << binFa.getAtlag () << std::endl;
kiFile << "var = " << binFa.getSzoras () << std::endl;
kiFile << "\nMozgatás után a binFa2"<< std::endl;
kiFile<<binFa2;
kiFile << "depth = " << binFa2.getMelyseg () << std::endl;
kiFile << "mean = " << binFa2.getAtlag () << std::endl;
kiFile << "var = " << binFa2.getSzoras () << std::endl;

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

```
    return 0;  
}
```

A különbség mostmár az hogy a csomópontból pointer lett. Tehát a konstruktorba be kellett vinni a konstruktur argumentum listájából az eddig átadott fa gyokeret(ugyebár eddig a konstruktor után volt írva a : után felsorolva). Mivel eddig a gyökér tagként szerepelt a csomópontba, de most mutató lett tehát könnyedén átadhatjuk az értékét a fának ami egy mutató, tehát memória cím átadása történik. Miután a gyökérből pointert csináltunk így könnyedén elhagyhatjuk az és jeleket ugyanis alapból a memória címét fogja átadni majd nem kell érték szerinti referenciaként hivatkozni rá. Viszont, így hogy pointer lett a destrukturban őt is felkell szabadítani tehát bele írjuk a delete gyokeret a destruktorkba.

6.6. Mozgató szemantika

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

Megoldás video:

Megoldás forrása: [https://github.com/Savitar97/Prog1/blob/master/mozgato\(pointer2.cpp](https://github.com/Savitar97/Prog1/blob/master/mozgato(pointer2.cpp)

A mozgató szemantikához a mutatós gyökerű binfát fogjuk felhasználni. Amíg nem írtuk meg a mozgató szemantikát addig tiltanunk kellett. Az osztály private részében. Ez a csomópontra és a fára egyaránt vonatkozik. A mozgató konstruktorra dupla és-el hivatkozunk. Az dupla és operátor jelzi, hogy jobboldali referenciairól van szó. Ez azért jó, mert a jobboldali referencia elkerüli a fölösleges másolatot. Ez azért jó mert ha másoljuk akkor megmarad az előző gyakran szükségtelen másolat, vagy úgyis elpusztítjuk egy destruktornal. De a mozgató szemantikával elkerüljük az ideiglenes másolatot mivel konkrét memória címekkel dolgozunk. A konstruktorba elsőként nullára állítjuk a gyökér pointert. Ez azért szükséges, hogy a régit bele tudjuk tenni az újonnan elkészített gyökérbe. Ezt a move segítségével érjük el. Ez lényegében az értéket átpakolja az újba majd a régit null-ra állítja. Majd ezután visszatérünk az új gyökérrel. Következőnek túlterheljük az =jel operátort. És az új gyökérelemekbe beleteszem a régi gyökér elemeket. Majd visszaadom az újnak a this-el. De a régieket ezután töröljük. A mozgatást a mainbe úgy érjük el, hogy a move-al átpakoljuk az egész binFa-t a binFa2-be. Ilyenkor a binFa üres lesz és a binFa2 lesz az új példány.

7. fejezet

Helló, Conway!

7.1. Hangyszimulációk

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



Felhasznált irodalom:

http://www.cs.ubbcluj.ro/~csatol/mestint/pdfs/BME_SpecialisUtkeresoAlgoritmusok.pdf

Az ábra a <https://bhaxor.blog.hu/2018/10/10/myrmecologist> származik.

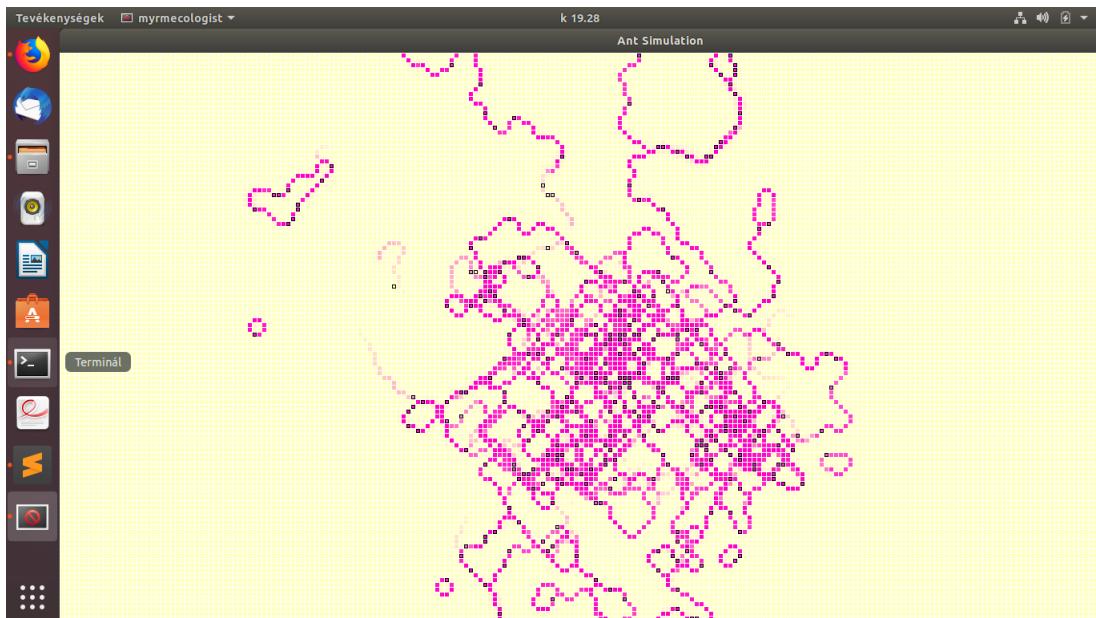
Megoldás videó: <https://bhaxor.blog.hu/2018/10/10/myrmecologist>

Megoldás forrása:

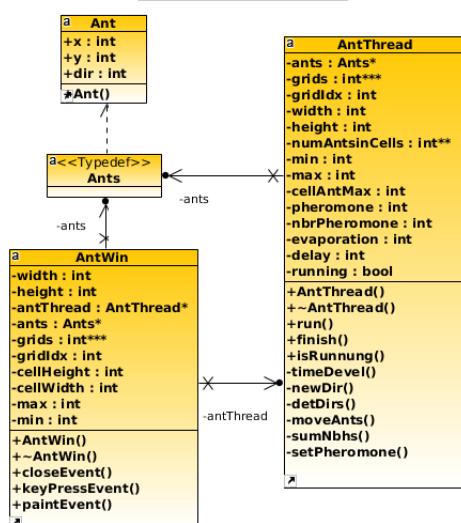
A hangyszimuláció a hangyák mozgását és nyomvonalát szimulálja, minden hangya feromon nyomot hagy, és ha feromon nyomra lép akkor elkezdi követni az előző hangya által hagyott feromont, a feromon kezdetben sötétzöld de egyre halványodik ha nem kezdi el követni egy újabb hangya. Minél erősebb egy feromon nyom annál nagyobb az esélye, hogy elkezdi követni egy közelben járó hangya. Elsőként is létrehozzuk a hangya osztályt, amelyben a hangyák koordinátáit és mozgásukat adjuk meg. A mozgásuk irányát maradékos osztással számoljuk. Majd típusdefiniáljuk a hangya osztályt, ez a multiplicitás miatt, ugyanis több hangyat akarunk majd létrehozni és kezelní. Az AntWinbe adjuk meg az ablak tulajdonságait (A hangyák megjelenítését, amely zöld négyzetekkel történik, a rácsvonalak méreteit és a megfelelő key eventeket ez itt most csak a pause a P vel és a Q-val vagy ESC-el való kilépés. Ha nem adunk meg semmit kapcsolóval akkor az alapértelmezett értékeket veszi fel viszont mi is megadhatjuk neki az ablak tulajdonságait és a hangyák tulajdonságait. Emellett a kirajzolás event is itt történik meg.

A parancssorban itt az érték megadása ást a qcommandlineoptionnal érjük el. Amit módosítani tudunk a w kapcsolóval a szélességet cellákban mérve. Az m-el a magasságot cellákban mérve. Ezen kívül megadhatjuk a hangyák számát (-n), sebességüket (-t két lépés közötti időt nézi), és a párolgási időt (-p), hogy hány mp után pusztitsuk el az objektumot. Majd a nyomvonalukat a hangyáknak ezek a feromonok (-f) és a cellák méretét (-c), hogy hány hangya fér rá egy cellára.

Az antthread.h ban vannak a program eventjei, hogy éppen fut -e a program vagy szünetel. Ha nem adunk meg kapcsolókkal értékeket akkor az antwin.h-ban lévő alap beállításokkal fog futni a program. A program futtatásához a Qt-nak legalább 5.2-es verziója szükséges.



7.1. ábra. Hangyaszimuláció



7.2. ábra. UML osztálydiagramm

Az ábra elején lévő `+` jelzi hogy globálisan hozzáférhető -e az adott program részhez a `-` a helyi hozzáférésű részek.

7.2. Java életjáték

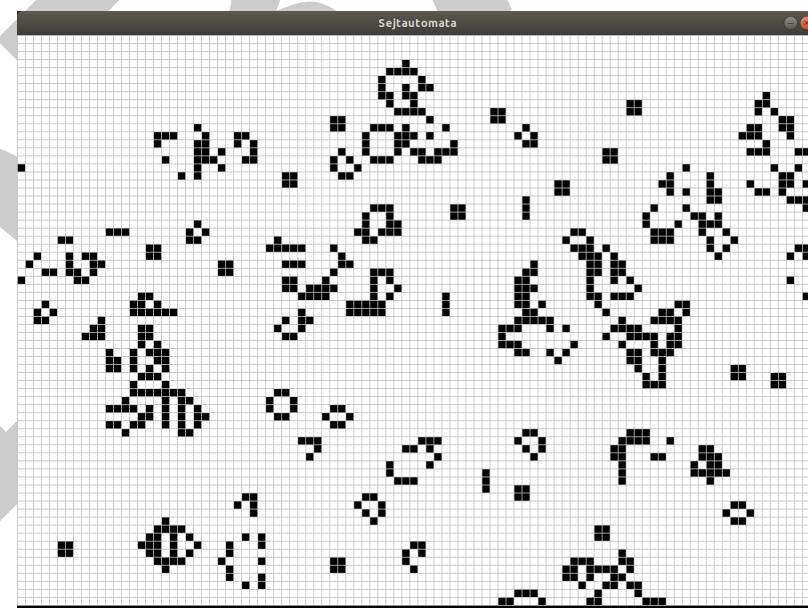
Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás videó:

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

A java életjáték lényege, hogy vannak élő és halott sejtek. Tehát a sejtnek ez a két állapota van. Egy élő sejt addig él tovább, amíg 2 vagy 3 szomszédja élő. Ha ez nem teljesül akkor elpusztul. Egyik véglet a túlnépesedés amikor 3-tól több a másik véglet amikor 2 nél kevesebb akkor pedig túl kevés az életben maradáshoz. Ha halott állapotban van egy sejt, akkor mindenkor halott marad, amíg 3 szomszédja élő. Két ráccsal dolgozunk egy jelenlegi állapottal és a megváltozott állapottal. Ezt az időfejlődés eljárás befolyásolja, itt figyeljük a sejtek állapotát, azt hogy hogyan változnak. A rácsok közötti váltakozást egy indextel figyeljük. Külön definiáljuk a cella méreteit és azt, hogy mekkora a sejttér azaz, hogy hány cella magas és széles. Emellett definiáljuk, hogy mennyi idő múlva váltszon a jelenlegi sejttér a következőre. A programban készíthetünk pillanatfelvételt az S gombbal ezeket számoljuk, hogy hányat készítünk. Magát a felvétel készítését egy bool típusú változóval érjük el. Majd a konstruktörben definiáljuk a sejttérünket, kezdetben minden sejt állapota halott. Majd létrehozzuk az első élőlényeket a siklókilövőket. Magát a sejttérét úgy hozzuk létre, hogy ami kiúszik az egyik oldalon az térjen vissza a másikon. A funkció gombok a program futása során a már tárgyalta 'S', ezen kívül van lehetőségünk felezni a cella méreteit a 'K'-val vagy esetleg dupláznia az 'N'-el. Emellett a 'G'-vel gyorsíthatjuk a rácsok közötti váltást vagy pedig az 'L'-el lassíthatjuk. Az egérmutatóval változtathatjuk a sejtek állapotát. Tehát ha húzzuk az egeret a cellákon akkor a halott cellákban élőket tudunk csinálni. A kezdeti cellaméret 10x10 es. A sejttér kirajzolásáért a paint eljárás a felelős. Ha egy sejt élő akkor feketével rajzoljuk ki ellenkező esetben fehérre színezzük a cellát. A rácsokat szürkével rajzoljuk ki. A létrehozott sikkok a sejttérben automatikusan másolják magukat és megadott irányba haladnak. És végülis a legjobban az egérrel tudunk beavatkozni az egész sejttér fejlődésbe. A mai inben már csak példányosítjuk magát a sejtautomatát a konstruktora segítségével. A következő feladatban majd láthatjuk ennek a feladatnak a c++ és qt segítségével elkészített megoldását.

Élet a sejttérben:



7.3. ábra. Sejtautomata

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kozepes/Qt/sejtautomata/>

A programban a sejt ablakban van definiálva, hogy mekkora legyen az ablak lényegében, hogy hány cella van. A cellák méretét külön adjuk meg itt, 6x6 os kocka. A sejtnek két állapota van élő vagy halott, élő marad mindaddig amíg kettő vagy három szomszédja van ellenkező esetben halott lesz. A sejt ablakban az élőlények a siklók, amelyek másolják magukat és adott irányba haladnak. A siklókat sikló ágyúkból lőjük ki. A sikló ágyúknak fix pozíciója van. A painteventben rajzoljuk ki magát a táblát és a skiló kilövőket. minden egyes kockának 8 szomszédja van az az egy sejt 9 kockát befolyásol. A lényeg, hogy teremthetünk egy világot ahol eldönthetjük a létrehozott sejtek számát az az megadhatjuk mikor lehet élő a sejt vagy mikor halott. Így végülis a populációt tudjuk befolyásolni. És ez egy olyan játék ahol a személy csak megfigyelje az eseményeknek nem pedig részese ugyanis miután beállítottuk a szabályait a világnak onnantól magától fut és teremtődnek az élőlények.

The screenshot shows a Linux desktop environment with a terminal window open in Sublime Text. The terminal output is as follows:

```
~/Asztal/Tanulas/Progi/sejtautomata$ g++ -fPIC -o sejtautomata main.o sejtblak.o sejtszal.o -lQt5Core -lQt5Widgets -lQt5Gui -lpthread
nemesis@nemesis-Aspire-A515-51G:~/Asztal/Tanulas/Progi/sejtautomata$ ./sejtautomata
```

The terminal also shows the execution of the program, which displays a Conway's Game of Life simulation on a 20x20 grid.

7.4. ábra. Életjáték

Káoszba fordult világ a halálozási arány csökkentésével.

7.4. BrainB Benchmark



Tutor

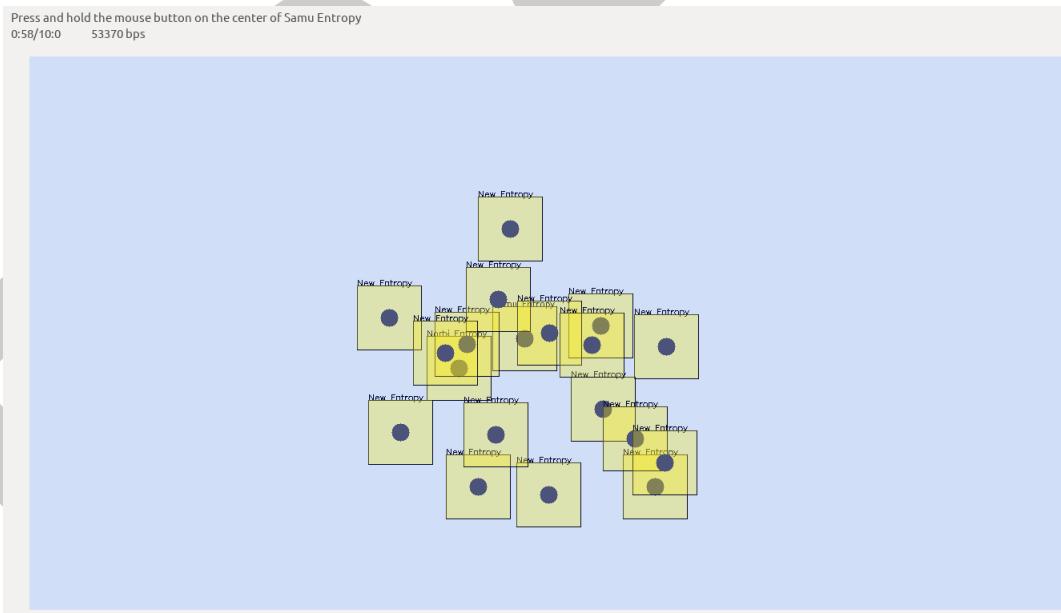
Ebben a feladatban tutoráltam Egyed Annát.

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/BrainBbench>

A BrainB benchmark egy felmérés. Amely főleg az mmorpgvel játszó játékosokat képes mérni. Mégpedig azt, hogy mennyire képesek követni a karakterüket a tömegbe. A lényeg, hogy az egérmutatót a karakterünkön saman tartson a játék pedig egyre több hőst generál a karakterünk köré és 10 percen keresztül nem szabad elveszteni a karakterünk nyomát. Majd a benchmark ad egy eredményt, amely számosítja a teljesítményét a játékosnak. Ez alapján készíthetünk esetleg egy táblázatot, hogy meghatározzuk sávokban is a pontszámokat, így megtudjuk ki a jobb képességű. A moba területen is hasznos lehet a játék ugyanis a teamfightokban a sok effekt között könnyen eveszíthetjük a karakterünket vagy épp a focusolandó enemy játékosét. Ezzel a programmal lehetséges, mérni az egyén teljesítőképességét és kiválogatni azokat akik sokkal jobbak az átlagtól és akár a csapatok meghatározhatnának egy minimális pontszámot, amit el kell érni a jelentkezéshez. Viszont a 10 perc szerintem elég sok idő. Már mint jó a koncentráció képességet próbára tesszük. De akkor is elég unalmas végig ülni, amíg a teszt lefut.

És most beszéljünk kicsit a programról a BrainBThread.cpp-ben hozzuk létre a hős osztályunkat és itt hozzuk létre(deklaráljuk) a hősünket Samut. A hős osztály konstruktorában. A mozgását az ablakban randommal számoljuk. A Qthread-ban határozzuk meg az eventeket. Ilyen a pause. És magát, hogy a hőst hogyan jelnítse meg, hogy írja ki a nevét stb. A BrainBWin-ben vannak meghatározva a presz eventek vagyis, hogy az egérgomb levan -e nyomva vagy nincs és hogy az S el mentse el az eredményt a P-vel pause-oljon az ESC-el vagy Q-val pedig lépj ki. Ha az egérgombot lenyomjuk akkor kezdődik a mérés. Az ifben vizsgáljuk, hogy az egér a hősünkön van -e ha igen akkor létrehozunk egy new entropy-t ez a incCompan van a BrainBThread.h-ban és növeljük a hősünk agilityjét 2 vel ha nincs rajta akkor kiszedünk a vektorból egy entropyt és csökkentjük a hősünk agilityjét.



7.5. ábra. BrainB benchmark

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> (/tensorflow-0.9.0/tensorflow/exa
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

<https://github.com/Savitar97/Prog1/blob/master/tensorflow/twicetwo.py>

Felhasznált irodalom:http://biointelligence.hu/pdf/tf_bkp.pdf

A tensorflowot a google készítette és fejleszti a gépi tanulást segíti, a tervezésben a fejlesztésben és a tanulmányozásban használják főként mivel készít adatáramlási gráfot, amelyben a node-ok a matematikai műveletek és az élek az áramló adatok. A tensorflow-ot import tensorflow ként hívjuk meg. A readimg függvény beolvassa a kép file-t majd dekódolja erre a későbbiekben lesz szükség. A program lényege, hogy a megadott képen szereplő számot felismerje. Ehhez meg kell tanítanunk a programunkat. Tehát elsőnek készítünk egy modelt. Majd ezen gyakoroltassuk a programunkat. Majd futtatunk egy teszt kört ahol a program kiirja a becsült pontosságát. Ezután a 42 es tesztkép felismerése következik. Majd végül a beolvastott képünkön teszteljük a program működését. A tesztnél a program súlyokat használ ami a W változó, ezzel dönt el a súlyokat, amely alapján dönt hogy benne van -e a kép amit megadtunk a jó halmaiban. Az x változó jelenti a bemenő értéket míg az y a számított kimenő érték. Ez hasonló mint a már átvett perceptron és neurális and or xor kapu. A súlyokkal minden szorzunk a b mint bias minden egy konstans érték. A tanulási folyamat is a neurális and or xor kapuhoz hasonló, ugyan úgy vannak hidden rétegek és nekik vannak node-jai. X a példa és Y a várt eredmény. A tanulánál ugyan úgy iterációs határt számolunk. Az Y értékét úgy számoljuk mintha egy egyenes egyenletét írnánk fel az $Y = x * a$ súlyjal ami a W és hozzá adjuk a b-t ami a kostants.

```
y = tf.matmul(x, W) + b
```

Ez az egyenes az ami elválasztja a jó megoldásokat a rosszaktól. Vagyis amelyik teljesíti a feltételt és megközelítőleg helyes értéket ad. Itt a feladatunkban az elfogadási arányt (iterációs határ, gradient) 50% nál húztuk meg tehát a program hibázhat. A pontosságot minél több hidden réteggel és noda-al tudjuk növelni.

Tehát az egész egy valószínűségi értéket figyel ha megüti a meghatározott küszöböt ez az érték és a hiba mértéke kevés akkor a program elfogadja mint megoldást. A programmal 28*28 pixeles képekről döntjük el, hogy milyen szám szerepel a képen. A y_{a} loss-t definiáljuk, amely azt számolja, hogy mennyire térünk el a a várt eredményünktől. A GradientDescentOptimizer deriváltakat számít a hibahatárok figyelembe vételével. Ez a minimize.

```
Fájl Szerkesztés Nézet Keresés Terminál Súgó
File "/home/nemesis/.local/lib/python2.7/site-packages/tensorflow/python/ops/nn_ops.py", line 2315, in _ensure_xent_args
    "named arguments (labels=..., logits=..., ...)" % name)
ValueError: Only call `softmax_cross_entropy_with_logits` with named arguments (labels=..., logits=..., ...)

nemesis@nemesis-Aspire-AS15-51G:~/Asztal/Tanulas/Prog1/tensor$ clear
nemesis@nemesis-Aspire-AS15-51G:~/Asztal/Tanulas/Prog1/tensor$ python twicetwo.py
Extracting ./tmp/tensorflow/mnist/input_data/train-images-idx3-ubyte.gz
Extracting ./tmp/tensorflow/mnist/input_data/train-labels-idx1-ubyte.gz
Extracting ./tmp/tensorflow/mnist/input_data/t10k-images-idx3-ubyte.gz
Extracting ./tmp/tensorflow/mnist/input_data/t10k-labels-idx1-ubyte.gz
2019-05-07 19:04:00.212620: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
2019-05-07 19:04:00.236574: I tensorflow/core/platform/profile_utils/cpu_utils.cc:94] CPU Frequency: 2712000000 Hz
2019-05-07 19:04:00.236888: I tensorflow/compiler/xla/service/service.cc:150] XLA service 0x55e8dd2d0460 executing computations on platform Host. Devices:
2019-05-07 19:04:00.236926: I tensorflow/compiler/xla/service/service.cc:158] StreamExecutor device (0): <undefined>, <undefined>
-- A halozat tanítása
0.0 %
10.0 %
20.0 %
30.0 %
40.0 %
50.0 %
60.0 %
70.0 %
80.0 %
90.0 %
-----
-- A halozat tesztelése
-- Pontosság: 0.9203
-----
-- A MNIST 42. tesztkepenek felismérése, mutatom a számot, a továbblepeshez csukd be az ablakat
-- Ezt a halozat ennek ismeri fel: 4
-----
-- A MNIST 11. tesztkepenek felismérése, mutatom a számot, a továbblepeshez csukd be az ablakat
-- Ezt a halozat ennek ismeri fel: 8
-----
nemesis@nemesis-Aspire-AS15-51G:~/Asztal/Tanulas/Prog1/tensor$
```

8.1. ábra. Softmax mnist

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

A deep mnist az előző példára épül annyi különbséggel, hogy itt sokkal több a hidden réteg és azokon a node-ok száma. Emellett két súlyjal dolgozunk. Az az 2 layerünk van a 2. layer az első layer által számolt adatokból dolgozik így már pontosabb eredményeket kapunk, azaz ezzel növelhetjük a pontosságát a programnak. Ugyebár a neuronoknak két állapot van vagy aktiválva vannak vagy nem tehát ez olyan mint a boolean vagy igen vagy nem. Mindig csak egy neuronunk lehet aktiválva, ha több van megkell találnunk melyik az amelyik a többi közül közelebb áll az eredményhez. Vagyis megkell találnunk azt a súly és bias párt amelyjel a legjobb pontosságot kapunk. Ezt a hibavisszaterjesztéssel ellenőrizhetjük. Tehát akkor a leg pontosabb az eredmény ha a pontosságunk a legnagyobb és a loss a legkisebb. Itt 32x32 képet vizsgálunk és már nem csak számokat képes felismerni hanem bármit amit megadunk az adatbázisba. Itt az adatbázis is egy sokkal nagyobb 100000 kép körüli a, futási idő is nagyon megnövekedett több órára. A reshape a bemeneti 2 dimenziós kép pixeleit átrendezi egy soros listába. Ez azért kell mert a program csak így tudja vizsgálni a pixeleket. A végeredmény ugyan úgy felismeri a képet az adatbázisból.

8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndi8>

Megoldás forrása:<https://github.com/Microsoft/malmo>

Megoldás forrása:https://microsoft.github.io/malmo/0.17.0/Python_Examples/Tutorial.pdf

Valószínűleg sokan ismerik a MineCraft játékot a kockákból álló világ melynek a főszereplője Steve. A program lényege, hogy Steve egy megadott időintervallumon belül ne akadjon el semmiben. Tehát képes legyen kikerülni a mozgását blokkoló akadályokat. Ezt úgy értük el hogy a steve körül lévő 26 kockát vizsgáljuk mivel ő áll a közepén. Ha nincs szabad út előtt megpróbálja kikerülni ha nem lehetséges akkor ugrik. Azt, hogy épp milyen objektum van Steve előtt a program jelzi. A forgásszámláló counter 8-ban lett meghatározva. Steve egyenesen előre halad mindaddig, amíg valamilyen akadályba nem ütközik. Akadályok lehetnek a víz, levegő, növényzet. A program futása során számoljuk az akadályokat és kiirjuk, hogy Steve előtt éppen milyen objektum van. Az akadályok elkerülésének lehetőségei az ugrás, a kitérés vagy az akadály elpusztítása. Az elpusztítás akkor lehet jó lehetőség ha 2 blokk magas fal veszi körül Stevet mert azt már nem tudja átugrani. A programba a cselekvési utasításokat a while world state is mission running ba kell beleírnunk. Az agent alapvető utasításai attack, move, turn, jump, stafe, use ezeknek argumentumként a sebességet adjuk meg 1 a max sebesség tehát 0 és 1 között adhatunk meg argumentumként neki értéket.

Sajnos a gyenge gépem miatt a futtatás eléggé haldoklott. De a telepítése az egésznek eléggé egyszerű letölthető a Malmo-0.37.0-Linux-Ubuntu-18.04-64bit_withBoost_Python3.6 ezt [innen](#) tudjuk megtenni. Ezután kicsomagoljuk a főkönyvtárunkba.

Szükségünk van még a következőkre: **sudo apt-get install libboost-all-dev libpython3.5 openjdk-8-jdk ffmpeg python-tk python-imaging-tk**

Majd bemegyünk a malmö mappájába ott megkeressük a Minecraft mappát és ott futtatjuk a **./launchClient.sh -t**

Valószínűleg szükséges lesz kiadni a terminálba egy **export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/** és egy **export MALMO_XSD_PATH=~/MalmoPlatform/Schemas** parancsot a futtatáshoz.

Ez megfog nyitni egy MineCraft játék ablakot. Ezután nyitunk egy újabb terminált és megkeressük a python exmaples-t. És itt tudjuk futtatni a mintákat és ha csinálunk saját küldetést szintén ide kell betennünk. Ha esetleg problémák lennének a futtatással érdemes megnézni, hogy a 10000-res portot nem -e használja más alkalmazás mivel alapértelmezetten ezt használja a malmö. Ha esetleg ütközés lenne a run_mission.py ba megkeressük a client infot is itt tudjuk átirni.

Ha fut a küldetésünk az f3-al tudunk bővebb információt kapni például memória használat milyen blokk van előttünk milyen irányba nézünk stb.

Minta egy küldetéses xml-re ahol az időt, a játékmódot, küldetés kezdetén a chatre kiirt szöveget stb-t tudjuk beállítani.

```
missionXML='''<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<Mission xmlns="http://ProjectMalmo.microsoft.com" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<About>
```

```
<Summary>Misszió leírás</Summary>
</About>

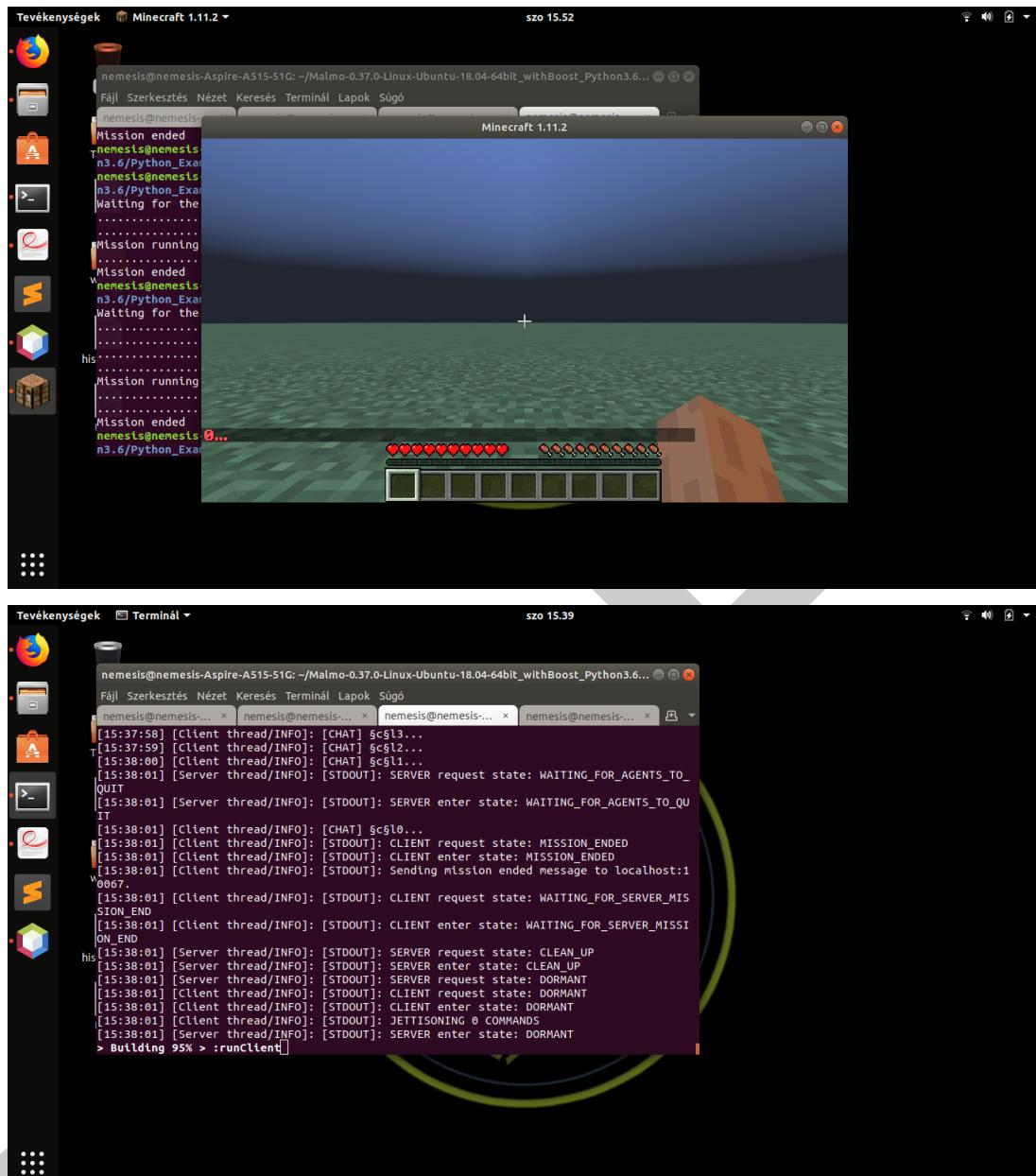
<ServerSection>
  <ServerHandlers>
    <FlatWorldGenerator generatorString="3;7,220*1,5*3,2;3;, ↵
      biome_1"/>
    <ServerQuitFromTimeUp timeLimitMs="50000"/> //itt ↵
      állíthatjuk be az időt
    <ServerQuitWhenAnyAgentFinishes/>
  </ServerHandlers>
</ServerSection>

<AgentSection mode="Survival">
  <Name>QuitBot</Name>
  <AgentStart/>
  <AgentHandlers>
    <ObservationFromFullStats/>
    <ContinuousMovementCommands turnSpeedDegs="180"/>
    <ChatCommands />
    <MissionQuitCommands quitDescription="give_up"/>
    <RewardForMissionEnd>
      <Reward description="give_up" reward="-1000"/>
    </RewardForMissionEnd>''' + malmoutils.get_video_xml( ↵
      agent_host) + '''
  </AgentHandlers>
</AgentSection>
</Mission>'''
```

Egy újabb példa az agent-nek kiadott parancsokhoz

```
while world_state.is_mission_running:
    print(".", end="")
    time.sleep(0.5)
    count += 1
    world_state = agent_host.getWorldState()
    for error in world_state.errors:
        print("Error:", error.text)
    for reward in world_state.rewards:
        print("Reward:", reward.getValue())
    agent_host.sendCommand("attack 1")
    agent_host.sendCommand("move 1")
    agent_host.sendCommand("jump 1")
print()
print("Mission ended")
```

Az agent host gondoskodik a cselekvések elvégzéséről.



9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

A lisp eltér az eddig használt programozási nyelveinktől. Mégpedig mivel a lisp fordított lengyel jelölést használ. Tehát kicsit másabb gondolkodás módot igényel. Itt az operátorok mindenkor elő kerülnek és nem közé tehát nem $a+b$ hanem $(+ a b)$. A függvények definíálását a define-al érjük el. Egy szám faktoriálisát kiszámolhatjuk úgy, hogy $n-1$ faktoriális szorozva n el. Ebből következik, hogy rekurzívan így néz ki a faktoriális:

```
rekurzív:  
  (define (fakt n) (if(< n 1) 1 (* n (fakt(- n 1)))))  
iteratív:  
  (define (factorial n) (define (iter product counter) (if (> ←  
    counter n) product (iter (* counter product) (+ counter 1)))) (←  
    iter 1 1))  
forrás:https://www.programminglogic.com/ ←  
  factorial-algorithm-in-lispScheme/
```

Iteratívan pedig egyszerűen csinálunk egy for ciklust, aminek a countere addig megy amíg kisebb mint a bemenő szám. És eddig összeszorozza az elemeket, amelyek a productban fognak tárolódni.

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 video: https://youtu.be/OKdAkI_c7Sc

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

A gimp króm effekt készítéséhez úgy kezdünk bele, hogy fekete háttérre fehér szöveget írunk. Majd ezt a két layert összefésüljük. Kezdetben létrehozzuk a programban a lokális változókat. Az imageba a gimp-image-new segítségével elkészítjük az új képet, ennek 3 bemenő argumentuma van szélesség ,magasság és rgb. A gimp-context-set-foreground el állítjuk be a színeket ennek a bemenő paramétere egy rgb színkód fontos, hogy a ' el jelezzük, hogy itt az első elem nem fv-név. A gimp-drawable-fill-el töltjük ki az adott színűre a groundot. Az gimp-text-layer-new létrehozunk egy új text layert ez argumentumként megkapja a képet, a szöveget, a betűméretet és a pixeleket. A gimp-image-insert-layer adja hozzá a lajert a képhez a megadott pozícióba. A szöveg eltolását a képen a gimp-layer-set-offsets segíti, ezzel igazítjuk középre. A két layer konbinációját a gimp-image-merge-down éri el. A második lépésben használunk egy erős gaus elmosást. Ezt a plug-in-gauss-iir-el tudjuk használni itt a függvény megkapja a képünket az elmosni kívánt layer, az elmosás értékét és az irányokat. A 3. lépésben állítjuk be a színszinteket, hogy egy nagyon világos vakító fehér színű szöveget kapjunk. Itt a gimp-drawable-levels-t fogjuk használni, amely megkapja a layer, azt hogy melyik csatornát akarjuk módosítani,az intenzitási értékeket, és a gammát. A step 4-el ugyan azt hívjuk. Majd az 5. lépésben kijelöljük a hátteret ami a fekete rész és ezt kell invertálni.A 6. lépésben létrehozunk egy új layert és ugyebár az előző kijelölés miatt megmarad a szöveg körvonala mivel a fekete részt szelektáltuk ki és ezt adjuk hozzá a képhez.Majd a 7. lépésben állítunk be egy szürkés skalár színátmennetet(gradienst) a szövegnek.Majd a 8. lépésben készítünk egy bump mapet a szövegnek. Majd a 9. lépésben állítjuk be a színgörbékét a gimp-curves-spline függvény segítségével és készen is van a scriptünk.



9.1. ábra. Nemesis króm effektel



9.2. ábra. Nemesisborder króm effektel

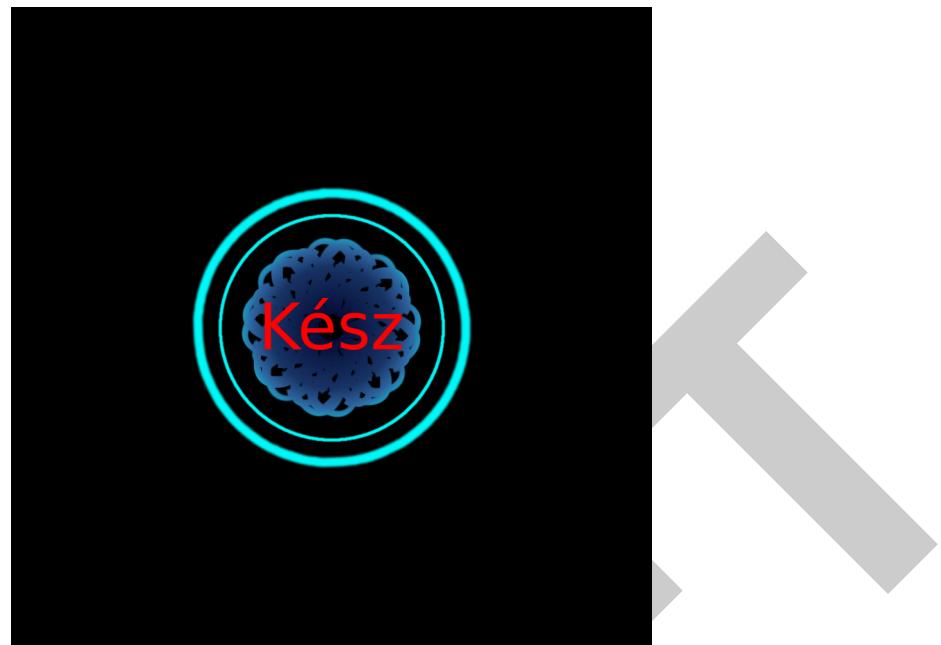
9.3. Gimp Scheme Script-fu: név mandala

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

Ebben a gimp scriptben ugyan úgy mint az előzőnél az elején definiáljuk az új képünket és layereinket. A gimp-image-new függvény létrehoz egy listát, mivel minden gimpfüggvény egy listát ad vissza. A lista elemeihez a car-al férünk hozzá, ami kiveszi a lista első elemét. A legfontosabb rész itt a forgatások mivel a mandala úgy készül, hogy a szöveget körbe forgatjuk egy pontban és így a betűk keresztezik majd egymást. A forgatást a gimp-item-transform-rotate-simple függvénytel érjük el. A változók értékét a set-el tudjuk beállítani. Mivel a betűknek a mérete változó szövegtípusonként ezért függvényt használunk ezeknek az értékeknek a kezelésére ez a text-wh függvény. Ez konkrétan lekéri a fonttípusnak a méreteit és azt tároljuk a szélességben és a magasságban. A forgatások után a plug-in-autocrop-layer törli az üres szegélyeket. Majd megadjuk a szélesség és magasságnak a kirajzolható méreteket a drawable-el ez a függvény a kirajzolható pixelekkal tér vissza. Majd ez alapján újraméretezzük a layert a resize- al. Az ecset méreteit a gimp-context-set-brush-size-al tudjuk módosítani ez pixelben adja meg a méreteket. Ezzel adjuk meg majd a mandala keretének a vastagságát, amelyet az gimp-image-select-ellipse-el hozunk létre. Ebből 2 van az egyik a külső körív a vastagabb 22 pixel míg a véknyebb 8 pixel vastag. Párbeszéd ablakokat a gimp-message-el tudunk készíteni. Az elkészített képet a gimp-display-new-al jelenítsük meg új ablakban. Majd ki cleareljük a képet gimp-image-clean-all. Futtatáskor a kép méretét tudjuk beállítani a betűtípushoz és a betűméretet. Ezen kívül a színt és színskálát.



9.3. ábra. Név mandala

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

A megírt programokat "forrásszövegnek" nevezzük. A nyelvtani szabályai a forrásszövegeknek a szintaktikai szabályok, míg a tartalmi szabályokat a szemantika adja meg. A forrásszöveget a fordítóprogram alakítja gépi kódára, amelyet a processzor feltud dolgozni. A fordító program végzi tehát a kód szemantikai, szintaktikai, lexikális vizsgálatát és a kód generálását a szemantikai hibát nem minden veszi észre mivel lehet, hogy olyan hibát vétettünk, amely formailag helyes csak nem a várt eredményt kapjuk. A gépi kódból a szerkesztő állít elő futtatható programot. A magas szintű nyelvek közül a C-ben előfordító segítségevel generálunk forrásszövegből forrásszöveget. Az interpreterek is megvan a saját elemzője viszont itt soronként veszi az utasítást és egyből el is végzi. A programnyelvek szabályai a hivatkozási nyelvek. Amikor a programozó programkódöt ír akkor algoritmusokat fogalmaz meg, amivel vezérli a processzort. A lefgőbb eszköz a változó, amelyben értékeket tud letárolni, amelyeket az algoritmusok változtatnak.

Imperatív nyelvek: Eljárásorientált nyelvek, Objektum orientált nyelvek.

Dekleratív nyelvek : Funkcionális nyelvek, logikai nyelvek.

Az adattípus egy absztrakt programozási eszköz. Az adattípusnak neve van , amely egy azonosító. Egy adattípushoz három dolog határoz meg: tartomány, műveletek, reprezentáció. A tartomány megmondja milyen értékeket lehet fel a változó. minden típusos nyelvnek vannak standard beépített típusai. Némelyik programozási nyelv megengedi, hogy definiálunk típusokat. Vannak olyan típusok, amelyet úgy kapunk, hogy egy másik típus tartományát szűkítjük le ők az altípusok. Adattípusok csoportja lehet skalár vagy struktúrált. Egyszerű típusok: egész, valós, karakteres, logikai. Összetett típusok: tömb (értékei csak egyfélé típusú lehet kivéve olyan programozási nyelvekben ahol megvan engedve, hogy a tömb összetett adattípusú legyen. A tömb indexei általában egész típusúak. A tömb nevével a tömb összes elemére képesek vagyunk hivatkozni.). Mutató típus: elemei memóriacímek, legfontosabb művelet a memóriacímen lévő érték elérése.

Nevesített konstans: minden deklarálni kell, van neve, típusa és értéke. Mindig a nevével hivatkozunk rá és a hozzá rendelt értékre hivatkozik.

Utasítások

Utasításokkal adjuk meg az algoritmus lépésein. Kétféle van deklarációs és végrehajtó utasítások. A deklarációs utasítások a fordítóprogramnak szólnak, olyan információt szolgáltat amelyet a fordítóprogram használ fel majd a tárgykód elkészítéséhez. A végrehajtó utasítások csoportosítása a következő szerint zajlik:

- 1. Értékkadó utasítás:módosítja vagy beállítja a változó értékeit
- 2. Üres utasítás:főleg az eljárás orientált nyelvekben van rájuk szükség ilyenkor a processzor egy üres utasítást hajt végre
- 3. Ugró utasítás:goto utasítás a program a futását máshonnan folytatja(ahová az ugró utasítás mutat)
- 4. Elágaztató utasítások;if else szerkezet vagy a többirányú switch szerkezet.Itt tudjuk irányítani, hogy a program futása merre haladjon tovább. Ifnél ha egy utasítás van a zárójel blokkot elhagyhatjuk.Switchnél van default-ág amely akkor hajtódiik végre ha egyik lehetőség sem hajtódiik végre.
- 5. Ciklusszervező utasítások:bizonyos utasítások ismétlése. Előírt lépésszámú ciklus for.Elől tesztelős ciklus while, hátul tesztelős do while.Ha egyszersem fut le üres ciklusnak hívjuk a do while mindenképp lefut 1x.Emellett a ciklusok lehetnek végtelenek és összetettek mikor egymásba ágyazzuk őket.
- 6. Hívó utasítás:
- 7. Vezérlésátadó utasítások:continue, break,return. A continuevel kitudunk hagyni például ciklusból lépésekkel,break-el megtudjuk szakítani a ciklust vagy az utasítást. A returnnal adunk vissza értékeket főképp függvényeknél használjuk őket.
- 8. I/O utasítások
- 9. Egyéb utasítások

Programok szerkezete:

Az eljárásorientált nyelvekben :alprogram,blokk,csomag,taszk létezik.

Az alprogramok az újrafelhasználás eszközei másnéven eljárások vagy függvények.A meghívásukkal aktivizálódnak.Meghívni a deklarált nevükkel tudjuk.Az alprogramoknak van neve, paraméter listája,törzse amiben az utasítások és vezérlések szerepelnek és környezete, amelyben megtudjuk hívni. A függvényeknek minden van visszatérési értékük,tehát értéket számolnak ez az érték bármilyen típusú lehet.Az eljárás ezzel szemben valamilyen tevékenységet hajt végre és ahol meghívjuk ennek a tevékenységnek az eredményét akarjuk felhasználni.

Függvényt meghívni csak kifejezésben lehet. A függvény akkor fejeződik be szabályosan ha van visszatérési értéke. Nem szabályosan legtöbbször megszakítás vagy goto utasítással való megszakítással. minden programozási nyelvben van egy fő program egység a main. minden alprogram ennek adja át a vezérlést.

A hívási lánc: amikor egy programegység meghív egy másik programegységet. Rekurzió lehet közvetlen:Amikor a program önmagát hívja meg rekurzívan vagy lehet közvetett amikor egy már előzőleg meghívottat és lefutott alprogramot újra meghív.Ezek minden átírhatók iteratív algoritmusokká ami kevesebb memóriát használ.Néhány programozási nyelvben meglehet határozni másodlagos belépési pontokat vagyis, hogy ne a fejtől fusson le a függvény vagy az eljárás.

Paraméterkiértékelés:formális paraméterlistából csak egy darab van viszont az aktuális paraméterlisták száma végtelen lehet. Paraméterkiértékelés aspektusok:sorrendi kötés vagy név szerinti kötés.

A blokk: olyan programegység amely más programegység belsejében helyezkedik el.A blokk aktivizációja úgy történik hogy vagy rákerül a vezérlés vagy a goto utasítással a kezdetére ugrunk.

Az I/O:

Az I/O az eszközökkel kapcsolatos kommunikációért felelős. Feladata a perifériák és program közötti adatmozgatás. Az I/O-nál az állományok a fontosak. Ezek lehetnek logikai vagy fizikai állományok, amelyeket funkcióik szerint is megkülönböztetjük van az input ami a program bemenete tehát már létező fájl. Az output a program által létrehozott fájl és van az input-output ez az eset, amikor egy fájlt beolvassunk majd módosítjuk a tartalmát, de nem új fájlt hozunk létre mint az outputnál.

Az adatátvitelt is két részre bontjuk van a folyamatos és a bináris átvitel. A bináris átvitelnél a bitsorozatnak meg kell egyeznie a tárban és az adattárolón is. A file streameket minden deklarálni kell. Figyelni kell, hogy milyen adatokkal dolgozunk és aszerint választani adattípust. A filestream deklarálásával és a filenév megadásával megnyitjuk az adott file-t és ekkor dolgozhatunk vele módosíthatjuk, felhasználhassuk a benne lévő adatokat. Ezeket a filestreameket a használat után minden lekell zárni. Kiemelt fontossággal a streamwritereket.

A C nyelvnek az input/output alapból nem része ezt külön könyvtár meghívásával implementálhatjuk a nyelvbe.

Kivétel kezelés:

A kivételkezelés egy meghatározott program rész, amely akkor fut le ha valamilyen esemény bekövetkezik. A kivételeknek van egy neve és egy kódja.

A beépített kivételek például nullával való osztás vagy egy tömb indexén való túl hivatkozás. A programozó is definiálhat kivételeket, ezt főleg a try catch szerkezzel képes elérni vagy az if-el. Kivétel keletkezésekor is folytatódhat a program például a goto utasítás használatával vagy egyszerűen olyan kivételt adunk meg, amely nem szakítja meg a program futását például a do while-ban ha azt vizsgáljuk, hogy a megadott formában adtuk -e meg a bemenetet, ha nem akkor csak annyit csinálunk, hogy újra kérjük, hogy adja meg a felhasználó.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

3. Fejezet Vezérlési szerkezetek A C-nyelvben az utasításokat pontos vesszővel zártuk. Az utasítás blokkokat { }-el jelöljük.

3.2 If-else Az if szerkezet döntést hozó utasítás. Ha(feltétel) utasítás else utasítás2 , az elsre nem minden van szükség lehet olyan is, hogy ha történik valami akkor csináljon valamit a program, ha nem akkor ugorja át. Az else ág minden a hozzá legközelebb lévő else nélküli ifhez fog tartozni. Ha nem így szeretnénk akkor az if hatáskörét { }-jelek közé kell tenni. Az ifnek van egy másik fajtája az else if itt több feltétel egymásba ágyazása történik. Itt a legutolsó else akkor fut le ha egyik feltétel sem teljesül. Amint egy teljesül a feltételek közül a program végrehajtja és kilép az else-if ágról.

3.3 Switch A switchet többirányú programelágazások esetén használjuk itt valamilyen állandó értékhez rendeli az utasítást. A switchben case-eket hozunk létre, amelyek akkor futnak le ha teljesül az állandó, ezen kívül minden case-t break-el kell zárni. Létezik egy default ág, amely akkor fut le ha egyik case feltétele se teljesül.

A break el nem csak a switchből tudunk kilépni hanem bármely ciklust képesek vagyunk vele megszakítani.

A for előírt lépésszámú ciklus, amelynek van egy kezdő értéke egy végértéke és egy lépésszáma.

A while addig fut amíg a ciklusfejben megadott feltétel hamis nem lesz.

A do while utasítás a while-al szemben az, hogy mindenkor lefut egyszer a ciklusban található utasítás.

A continue utasítás segítségével lehetséges, hogy egy ciklusból kihagyunk lépéseket vagyis,hogy átugorjunk egy egy lefutást. A goto utasítással a megadott címkére ugorhatunk, goto utasítással általában több egymásba ágyazott ciklusból tudunk kilépni mivel a break nem működik ezeknél.

10.3. Programozás

[BMECPP]

A BME-tankönyv második fejezete a C++ újításait veszi szemügyre a C-vel szemben ezek nagyrészt, csak az olvashatóbb kódot szolgálják.

Az első változás a függvények paraméterénél van.C++ ban ha egy függvénynek nem adunk paramétert akkor az egy void paraméterrel lesz egyenlő. C-ben ugyanez tetszőleges számú paramétert jelentett,de C++-ban ez a lehetőség már a ...-lett.

Ha nem adunk meg visszatérési típust akkor C-nél ez az int lesz viszont C++-nál már hibát ír ki a fordító azaz nincs alapból ilyen definiálva.

C++-ban kétféle main függvény van a sima int main() és létezik az int main(int argc,char** argv). Az argc a bemenő paraméterek számát, míg az argv egy 2 dimenziós tömböt ad vissza magukról a bemenő argumentumokról. És C++-ban már nem kötelező a return 0; ami a sikeres futást jelzi.

Ezeken kívül bevezetésre került a bool típus ez megkönnyíti az olvasást két értéke van true és false.

Emellett alapértelmezett típus lett a wchar_t amivel több bájtos karaktereket lehet letárolni pl unicode karakterek.

Képesek vagyunk C függvényeket meghívni C++-ból ez az extern "C"-vel lehetséges. Ez a fordítottját is lehetővé teszi,ha a C++-ban definíálunk egy ilyen függvényt akkor C-ből képesek vagyunk C++ függvényt hívni.

Emellett adhatunk meg alapértelmezett argumentumokat ezeket arra az esetre hozhatunk létre ha létrejöhet olyan hiba, hogy a felhasználó kevesebb argumentumot ad meg ekkor van rá egy alapértelmezett alternatíva.

A változók deklarálása bárhol történhet ahol utasítás lehetséges a C++-ban érdemes mindenkor előtt deklarálni egy változót mielőtt felhasználunk ezzel átláthatóbb a kód.

Függvénynevek túlterhelése: A C-ben a függvényeket a neve azonosította, így nem lehetett két ugyanazzal a névvel rendelkező fv.-t létrehozni. De a C++-ban egy fügvényt már a neve és az argumentumai együttesen azonosítja.

Paraméter szerinti átadás: A függvényben pointert hívunk meg int* valami néven,míg a változó előre egy és jelet teszünk, így a változó memória címét adjuk át a függvénynek tehát ha valamelyen módosítás történik a változóval a függvényben az kihat a mainben deklarált változóra is. C++-ban bevezették a referencia típust.Így elég ha simán átadjuk a változó értékét majd a függvényben adunk és jelet az argumentumnak aztán mint egy sima változó úgy tudunk vele dolgozni.

Az és jel ezen kívül még egy egyoperandusú operátor ami a változó címét adja vissza C-ben még nem szerepelhetett deklarációjánál,így a C++-ezt felhasználhatta a referencia típushoz.

A cím szerinti paraméter átadás főként a nagyméretű adatszerkeztnél hasznos ugyanis nem kell egy másolatot készíteni róluk, hanem közvetlenül használhatjuk az adatszerkezetet és módosítgathatunk benne.

A programok egységbe zárás alapelveit nevezik osztálynak. Az osztályoknak lehetnek példányai ezeket objektumoknak nevezzük. Az objektumnak azt a tulajdonságát, hogy a többi program ne férjen hozzá a tulajdonságaihoz adatrejtésnek nevezzük. Öröklődés amikor az egyik osztály öröklí a másik osztály bizonyos tulajdonságait. Behelyettesíthetőség a fentebb lévő osztályba minden behelyettesíthessük az elvontabb osztályokat. Típusmogatás az osztályok támogathatnak operátorokat és típuskonverziót. A struktúráknak nem csak tagváltozói, hanem tagfüggvényei is lehetnek. A tagváltozók megnevezése attribútumok, míg a függvények metódusok. Ahányszor létrehozunk egy példányát a stuktúrára a tagváltozók annyiszor foglalnak helyet a memoriában. A tagváltozókra az arrow vagy a . operátorokkal hivatkozhatunk. A hatókör operátor vagy scope azt segíti elő ha több osztálynak van ugyan olyan nevű függvénye akkor képes megkülönbözteni őket. A tagváltozókkal ellentétben a tagfüggvényeknek nem történik többszörös helyfoglalás ezek egy példányban jönnek létre. Mivel a függvények képesek változtatni a tagváltozókat ezért pointereket használunk a tagfüggvényeknél és láthatatlan első paramétereket alkalmazunk. Ezek a láthatatlan első paraméterek a példányosított osztály mutatója. Ha van egy ugyanolyan nevű tagváltozónk és függvény argumentumunk akkor az argumentumok és lokális változók az erősebbek, ilyenkor általában a this-el hivatkozunk a tagváltozóra. Az adatrejtésnél a public részben lévő tagváltozókat mindenki eléri mint egy globális változót viszont a private részben csak a belső tagfüggvények férnek hozzá, ilyenkor lekérdező függvényeket kell írnunk. Ha nem írunk láthatóságot szabályzó kulcsszavakat automatikusan publicot használ viszont az osztály private-t. Az osztály az egy típus. Egy osztály több osztály is felhasználhat ezért a .h-fájlban a #ifndef és #define segítségevel érjük el hogy az osztálydefiníció többször is be legyen includeolva egy programba. Mivel a csak deklarált változók véletlen értékeit hordoznak ezért szükségünk van a konstruktorkra amelyek inicializálják a tagváltozókat. Ez egy olyan speciális tagfüggvény, amelynek ugyan az a neve mint az osztálynak és minden egyes példányosításkor lefut. A függvénynév túlterhelése miatt egy függvénynek lehet különböző paraméterszámú konstruktora. A destruktorkor a fölösleges memória használat felszabadítását végzik általában ~-el kezdődnek és ezt az osztály neve követi, nem lehet argumentuma. Dinamikus memóriaterület kezelés: a malloc és free függvényekkel lehetséges. A C++-ban a dinamikus memória kezelést a new végzi és a felszabadítást a delete. A new használatával már nem kell számolatni a tömböknél a lefoglalt hely értékét, mivel magától képes kiszámolni. A tömböknél mindenkor a szögletes zárójelt használjuk tehát a delete-hez is hozzá kell írni a szögletes zárójelet ha tömböt akarunk felszabadítani. Ha Fifo adatszerkezetet akarunk használni akkor ha új elemet akarunk hozzáadni akkor egyetlen nagyobb területet kell foglalnunk majd a végére beszűrni az értéket viszont ha ki akarunk értéket szedni akkor az elsőnek betett értéket tudjuk kivenni majd miután kiszedtük az értéket belőle a megmaradt elemeket visszamásolni és a destruktur a fölösleges helyet elpucolja. Másoló konstruktur ez esetében az inicializálás a már meglévő osztály változói alapján történik mivel egy másolatot akarunk létrehozni. A fordító a másoló konstruktort hívja meg ha megvan írva ha nem írunk másoló konstruktort akkor alapértelmezetten bitenként másol. A bitenként másolás neve a sekély másolás. Ha a dinamikus adattagokat is másoljuk azt mély másolásnak nevezzük. Érték szerinti paraméter átadásnál referencia szerint kell átadni a másolókonstruktur paraméterét.

A friend függvénytel egy osztály feljegosít más osztályokat vagy globális függvényeket, hogy a private részéhez hozzáférjenek. Tagváltozókat a :-al tudjuk inicializálni a konstruktur zárójele után írva. A referencia tagváltozókat kötelező az inicializálási listában inicializálni. Statikus változókat a static kulcsszóval deklarálunk. Ez hasonlít a globális változókhöz, de annyi különbséggel, hogy itt meg kell adni melyik osztályból származik a ::-al. Kezdőértéket nem kötelező adni nekik, mivel ekkor 0 lesz a kezdőértékük. A statikus változóknak a program indításakor foglalódik hely és csak a program bezárásával szabadul fel. A statikus függvények törzsében nem használhatunk this mutatót mivel nem lenne értelme. Az első futtatott kód a program indulásakor nem a main függvény első sora hanem a statikus és globális változók

konstruktori. Beágyazott függvények esetén meg kell adni a teljes elérési utat, ha nem az osztálydefinícióban definiált. A beágyazott osztályoknál nem kap speciális jogokat sem a beágyazott sem a tartalmazó osztály. A különböző absztrakt adattípusok miatt megjelent az operátor túlterhelés. A C++-ban az operátorok az argumentumai kon végeznek műveletet, az operátorok különböző számú argumentumot igényelhetnek. A C++-ban az operator kulcsszó. Itt nem az operátorok működésének megváltoztatása a cél, hanem az, hogy a saját magunk által létrehozott típusra is használhassuk. A túlterhelt operátorokat általában tagfüggvényként érdemes definiálni.

Típuskonverzió:

C++-ban az enum típusnál a típuskonverziót muszáj kiírni, ugyan ez a helyzet a void*-nál ugyan ez a helyzet áll fent. Referenciaként való átadásnál nem használható típuskényszerítés. Ugyanis a memóriareprezentációk eltérőek minden típusnál. Függvénynek nem tudunk átadni ideiglenesen létrehozott értéket, csak ha a függvény konstanst kér paraméterül. Mivel az ideiglenesen létrehozott értékek konstansok.

Ha viszont nem akarjuk változtatni az értéket akkor a fordító engedi felhasználni a kényszerített típuscserét csak, ilyenkor a const kulcsszót kell használnunk. Tehát elkerüljük a fölösleges másolatásokat és dolgozhatunk az argumentumként átadott értékkel.

A C++-ban a megírt osztályokat könnyen használhassuk típusként mint a beépített típusokat akkor azt az operátor túlterheléssel és konverziós konstruktur jelenti a megoldás kulcsát. A stringeket a \0 karakterrel zárjuk ez egy nul terminator karakter, az az egy záró karakter, ezt nem számoljuk a string hosszába. Tehát egy char tömböt a legkönnyebben úgy alakítunk stringgé, hogy túlterheljük az összeadás operátort, amely így összefogja fűzni a szöveget és a char tömb elemszámát 1 el megnövelve a végére tesszük a \0 záró karaktert. A típuskonverzió a beépített típusokra úgy működik, hogy a kívánt típust zárójelekbe zárva el tesszük az átalakítani kívánt változó neve elő. Ha egy argumentumú a konstruktur akkor itt a konvezíó magától értetődik. Ha kiakarjuk kapcsolni az automatikus típuskonverziót akkor azt az osztályunk konstruktora elő írt explicit kulcsszóval érhetjük el. A visszaalakítást az operator kulcsszóval érhetjük el ilyenkor a konstruktur visszatérési értékéhez nem kell típust megadni mert azt már a bemenetkor megkapott változó megmutatja. Az ilyen operátorok csak tagfüggvényként hívhatók meg, mivel globálisan nem alkalmazhatók. Viszont nincsenek kizárvá az öröklődés alól és lehetnek virtuálisak is. Ha több megoldás is létezik akkor a fordítónknak meg kell adnunk, hogy pontosan, hogyan szeretnénk létrehozni a típus átalakítását. Ha nem írnunk másoló konstruktort akkor is létezik, mivel a fordító biztosít egy bitenkénti vagy másnéven sekély másolást. A leszármaztatás és az öröklődés nagyon veszélyes ugyanis felléphet olyan probléma, hogy a mutató a program által használt memóriaterüten kívülre próbál hivatkozni, ezért az opréndzszer a biztonság szempontjából leállítja a programot.

A C++ a típuskonverzióra új teret nyit ezek a castok. Ez lehet statikus vagy dinamikus vagy konstans, esetleg újraértelemező konverzió. Használata az előző 4 _cast típus_neve és az átalakítani kívánt változó. Konstans típust csak konstans típuskonverzió alakíthat nem konstans típussá.

Kivételkezelés(hozzá kapcsolódó rész a pici könyvben):

A kivételkezelés lényege, hogy a program futása során felmerülő problémákat kezeljük. Tehát a program ne hibásan folytassa a működését. Ha valamilyen kivétel bekövetkezik a kivételkezelő segítségével érjük el, hogy a program egy új ágon folytatódjon. A try-catch szerkezet az alap hibakezelő parancs. Itt különböző hibaüzeneteket a throw kulcsszóval tudunk megadni. A try blokkban lévő kód lefut ha nem tapasztal hibát és ilyenkor a catch ág nem fut le és a catch után folytatódik azonnal a program. Ha hibát tapasztal akkor kiirja a throw-ban megadott hibaüzenetet. A throw paraméterét kivétel objektumnak is nevezzük. A throw utasítás a returnhoz hasonló. A catch(...) minden kivételt eltud kapni viszont ha megadunk neki valamilyen argumentumot akkor csak akkor dob hibaüzenetet ha illeszkedik valamelyik hiba típusra. A nem elkapott

kivételek esetén a program meghívja az abort függvényt, amely a program bezárásával jár ezek a kezeletlen kivételek. A kivételkezelésnek vannak szintjei, ezt a try-catch blokkok egybeágazásával érhetjük el. A kivételkezelésnél lehetséges a kivétel újra dobása tehát a throw-al elkapott hibát tovább dobhatjuk a throwval egy fentebbi szinten lévő kivétel kezelőnek. A kivétel dobása és elkapása között is futhat le utasítás, ilyenek az osztályok destrukturai amelyek felszabadítják az objektumot. Az uncaught_exception függvény megmondja, hogy a kivételkezelés miatt futott -e le a destruktur vagy sem. A valóságban persze kivételkezelő osztályokat használunk inkább a kivételek kezelésére.

DRAFT

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, Berners Lee!

11.1. C++ és Java

Ebben a fejezetben a Java és C++ nyelv összefüggéseit és különbségeit fogjuk vizsgálni. Már a könyv elején is említik, hogy a Java a jelölésrendszerében nagyon sok minden átvett a C++-ból. Az előzőleges tudásunkból pedig tudjuk, hogy a C++ eljárás és objektum orientált nyelv, míg a Java már szimplán az objektum orientált szemléletmódot követi. Az objektum orientált programozás célja, hogy implementálja a valós-világ egyedeit. Az objektum a valós világ egyedeire utal. Míg az objektum orientált programozás egy paradigma arra, hogy olyan programot írunk, amely osztályokat és objektumokat használ. Az objektumoknak vannak tulajdonságai és van viselkedésük. A tulajdonságokat változókkal írjuk le általában, míg a viselkedést a metódusokkal jellemizzük. Az objektumok lehetnek fizikai vagy logikai dolgok. Emellett az objektum orientáltságánál fontos megemlíteni az öröklődést. Amikor az egyik objektum örökli minden tulajdonságát és viselkedését a szülőobjektumától. A Java-ban ezen kívül nagy figyelmet fordítottak a biztonságra és a megbízhatóságra. Ebből következik az, hogy itt már nincsenek pointerek minden referencia. A Javában interpretált használnak, míg a C++-ban compiler végzi a fordítást. A C++ fordító gépi kódra fordítja a programot. Addig a Java fordítóprogramja egy byte kódot hoz létre, amelyet a JVM futtat. Ezért a Java platform független. Míg azért a C++-nál vannak megkötések. A Java program objektumok és ezek blueprintjeinek összessége. Az osztály változókból és metódusokból épül fel. A JVM hátránya a sebesség.(lassabb mint a compiler)

11.2. Python

12. fejezet

Helló, Arroway!

12.1. 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)

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/tree/master/polar>

A polargenünk két előállított normálisa a v1 és v2 változó. A példánytagunk, amely a nem visszaadottat tárolja az a tarolt nevű változó. A logikai tagunk a nincsTarolt változó ezt igazról indítjuk, mivel az elején még nincs érték a tarolt nevű változónkban. Tehát elsőnek az if águnknak a true ága fog lefutni elsőként. Itt addig fut a do while ciklusunk, amíg a w változó értéke nem lesz 1-nél nagyobb. Majd számolunk egy r-t. A tarolt változónkba letároljuk az r*v2-t. Majd a nincsTarolt-at false-ra állítjuk és kiiratjuk a nem tárolt normálist az r*v1-et. Következő futtatásnál az ifnek a hamis ága fog lefutni. Tehát a letárolt értéket fogja kiiratni a program és a nincsTarolttat újra true-ra állítjuk, hogy új két normálist számoljon a programunk. A konstruktorban inicializáljuk a nincsTarolttat, hogy minden egyes objektum példányosításnál true értéket vegyen fel a nincsTarolt változó.

```

Tevékenységek Terminál ▾
Megnyitás A
Random.java
k 9.54
~/Letöltések/jdk/src/share/classes/java/util
/*
 * independent values at the cost of only one call to {@code StrictMath.log}
 * and one call to {@code StrictMath.sqrt}.
 *
 * @return the next pseudorandom, Gaussian ("normally") distributed
 *         {code double} value with mean {code 0.0} and
 *         standard deviation {code 1.0} from this random number
 *         generator's sequence
 */
synchronized public double nextGaussian() {
    // See Knuth, ACP, Section 3.4.1 Algorithm C.
    if (haveNextNextGaussian) {
        haveNextNextGaussian = false;
        return nextNextGaussian;
    } else {
        double v1, v2, s;
        do {
            v1 = 2 * nextDouble() - 1; // between -1 and 1
            v2 = 2 * nextDouble() - 1; // between -1 and 1
            s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = StrictMath.sqrt(-2 * StrictMath.log(s)/s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
    }
}

/**
 * Serializable fields for Random.
 *
 * @serialField seed long
 *         seed for random computations
 * @serialField nextNextGaussian double
 *         next Gaussian to be returned
 * @serialField haveNextNextGaussian boolean
 *         nextNextGaussian is valid
 */

```

12.1. ábra. Polargen random

A C++-os verzióban a header file-ban írjuk meg a PolarGen osztályunkat. Az osztálynak itt nem csak a konstruktőröt kell megírni, de a destruktőröt is. Javaval ellentétben, ugyanis ott a garbage collector elvégzi a munkát, tehát nekünk nem szükséges definiálnunk. Az osztály következő nevű függvényét mivel kívülről hívtuk meg, ezért a ::-t kell használnunk, amely mutatja melyik osztály hatáskörébe tartozik. A felépítés a Java-s hoz képest nem sokat változott. A különbség annyiban szembetűnő, hogy a java erősen objektum orientált, míg a C++-ban lehetőségünk van ettől eltérni. A példányosításnál is látunk minimális különbséget ugyanis a javaban minden a new kulcsszót kell használnunk, míg C++-ban elég az osztály nevét és az objektum nevét megadni.

12.2. Homokozó

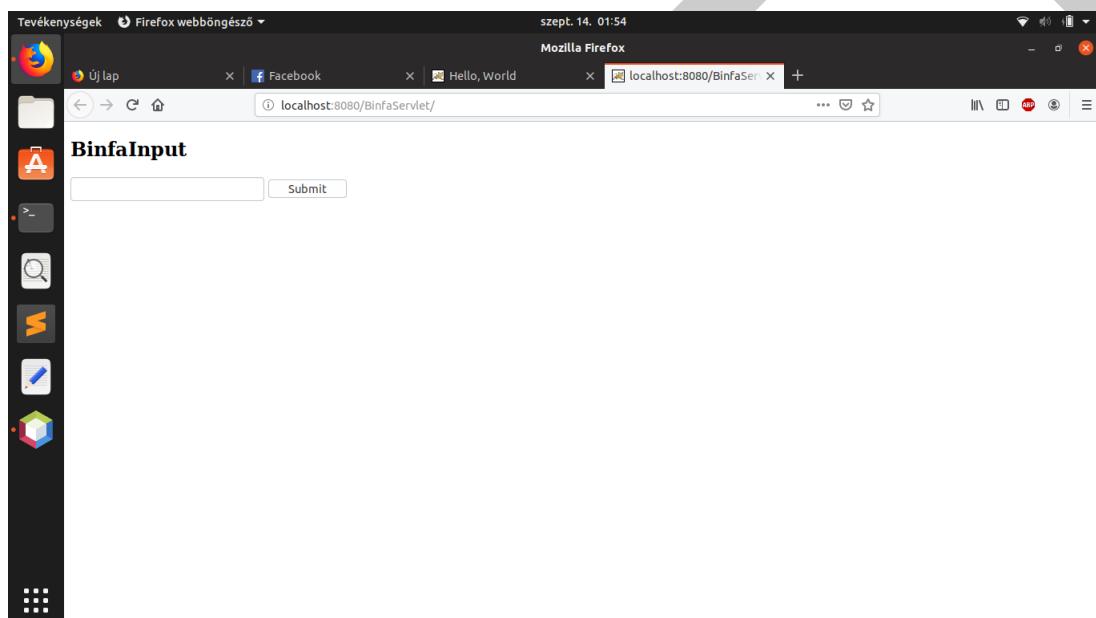
Megoldás videó:

Megoldás forrása:<https://github.com/Savitar97/Prog2/tree/master/Binfa>

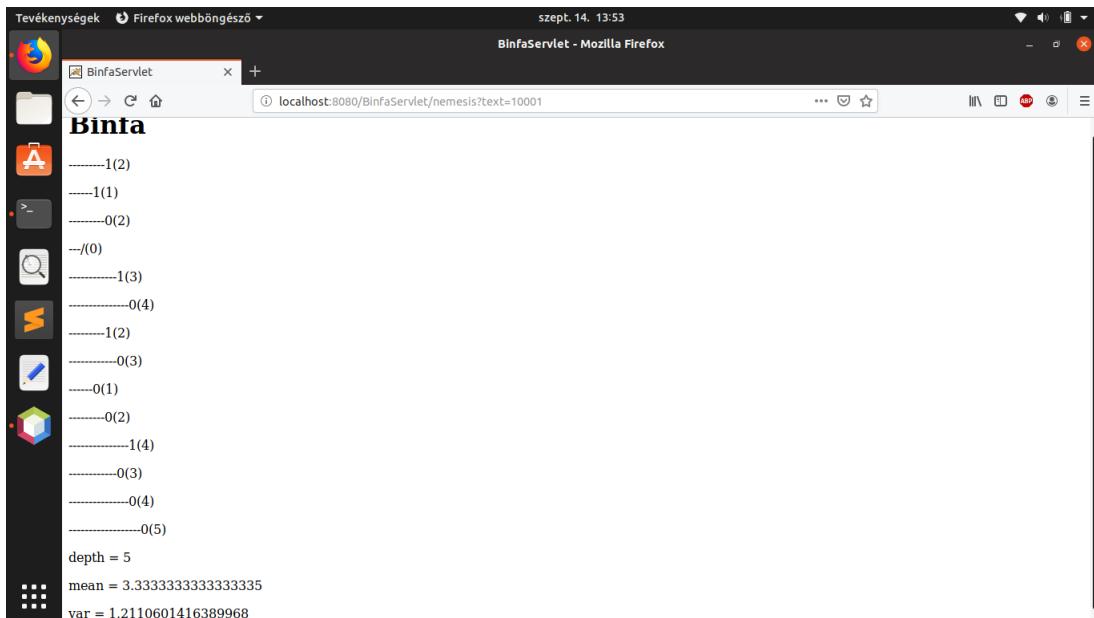
Servlet megoldása:<https://github.com/Savitar97/Prog2/tree/master/WebApplication1>

A binfa java a c++-hoz képest nem sokat változik. A különbség annyi, hogy itt nem választhatunk, hogy tagként, referenciaiként vagy esetleg pointerként adjuk át a csomópontot. Ugyanis a javaban nincsenek pointerek. A létrehozott Node osztály azaz a csomópont osztály tartalmaz egy érték változót és egy jobb és egy bal oldali objektumát a csomópontnak. Ezen kívül tartalmazza a jobb és baloldali csomópontok értékének lekérdezéséhez szükséges get függvényeket és az értékek beállításához szükséges set függvényeket, ezen kívül a csomópont értékének lekérdezéséhez szükséges get függvény. Most nézzük a binfa osztályt. Először is létrehozunk egy final kulcsszóval ellátott Node objektumot a root-ot, ez lesz a fánknak a gyökere. Tehát ennek az értéke nem változik fix marad az egész program futása során. Ezen kívül létrehozzuk a csomópontnak a jelenlegi objektumát amibe, majd az értéket pakoljuk, ami éppen jön a bemenetről. Emellett itt definiáljuk az átlaghoz, mélységhoz, magassághoz és szóráshoz szükséges változókat private kulcsszóval. Javaban látszik, hogy nem private blokk van, hanem minden jelezni kell, hogy az a változó, metódus, függvény

stb éppen milyen láthatósági körben szerepel. Az osztály konstruktoraiban inicializáljuk a csomópontot és a fa magasságát és a fa kezdetét ráállítsuk a csomópontra. Ugyanis innen fogunk indulni. Emellett csinálunk egy olyan Node típusú függvényt, amely mindenkor a root-ot fogja vissza adni. Az épp bemenő értékeket a metódusokban és a Node konstruktoraiban is jelzett Node típusú rhs objektumba adjuk át. A write függvényel írjuk be a bemenő értékeket a fába. Ha a bemenő karakter 0, akkor megnézzük van -e érték a bal oldalon, ha nincs akkor létrehozunk egyet azaz beírjuk a 0-ást a fába. Egyébként ha van akkor rálépünk arra a csomópontra és nézzük a következő karaktert. Ugyan ez fut le akkor is ha 1-es érték megy be a fába csak logikusan ott a jobb oldalt vizsgáljuk. A writeout metódussal fogjuk kiiratni a fánkat. Ezen kívül még vannak a mélység, átlag, szórás számoló függvényeink. Legutolsó sorban nézzük meg a main osztályunkat. Itt láthatjuk, hogy String args-t vizsgálunk. Ugyanis a binfának meg kell adni egy bemenetet és egy kimenetet. Ha tehát az argumentumaink száma kisebb mint 2 akkor hibát íratunk ki. Majd létrehozunk egy filereadert és példányosítjuk a BinfaFánkat bt néven. Majd a while függvényel olvassuk ki a bemenő file-ból az adatokat kihagyva pár karaktert például a spaceket. Végül a 2. megadott argumentumunkba beírjuk a kimenetet.



12.2. ábra. Binfa Servlet indexpage



12.3. ábra. Binfa Servlet after run

A binfa szervlet megvalósításához a netbeans ide-t használtam. A servlethez a binfa main osztályát kellett áthelyezni. Ehhez az osztálynak örökölnie kellett a HttpServlet tulajdonságait és Override-olni a doGet függvényt, amelynek a két paramétere a request és a response, na meg persze importolni a servlet könyvtárakat. Lényegében a böngésző címsorából adom meg a bemenetet a binfának. A request.getParameter el és ennek a zárójelben megadott paramétere hivatkozik az indexben megadott inputra. Tehát a textboxra. És a kimenetnél történt változásnál a setContentType-nál állítjuk be, hogy milyen típusú választ akarunk küldeni a kliensnek itt most jelenleg text/html. Majd a response.getWriterrel küldjük át a kliensnek a választ és a out.println-al építjük fel a html lapot, amit a kliens fog megkapni. A szerverhez Tomcatet használtam.

12.3. "Gagyi"

Az ismert formális 2 tesztkérdéstípusra adj a szokásosnál (miszerint x, t az egyik esetben az objektum által hordozott érték, a másikban meg az objektum referenciája) „mélyebb” választ, írj Java példaprogramot mely egyszer végtelen ciklus, más x, t értékekkel meg nem! A példát építsd a JDK Integer.java forrására 3, hogy a 128-nál inkluzív objektum példányokat poolozza!

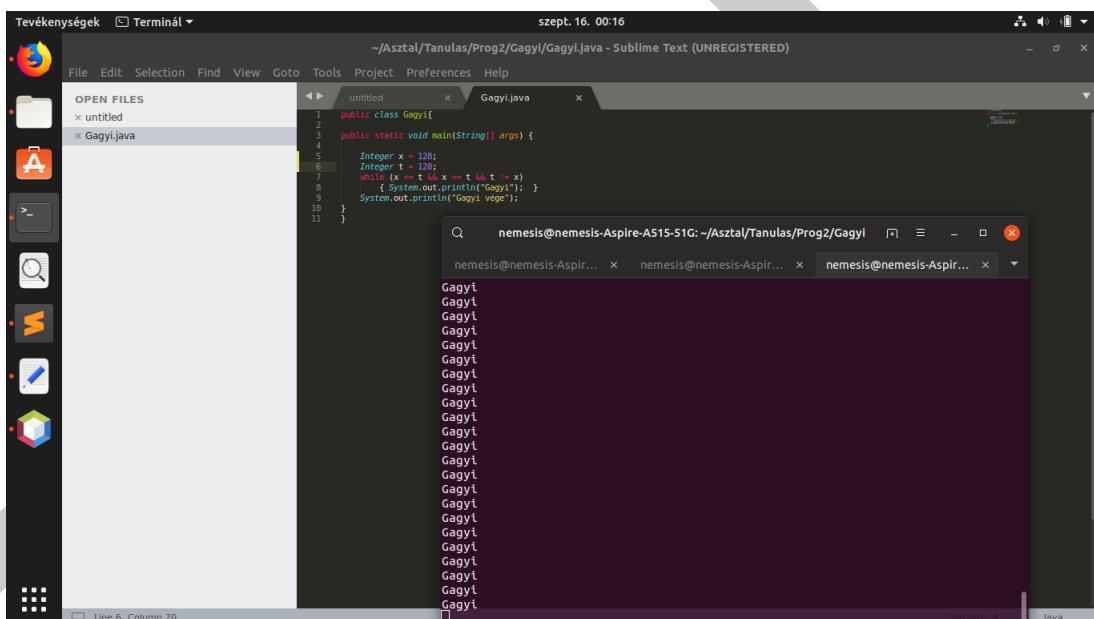
Megoldás videó:

Megoldás forrása:<https://github.com/Savitar97/Prog2/tree/master/Gagyi>

```
90: // This caches some Integer values, and is used by boxing
91: // conversions via valueOf(). We must cache at least -128..127;
92: // these constants control how much we actually cache.
93: private static final int MIN_CACHE = -128;
94: private static final int MAX_CACHE = 127;
95: private static Integer[] intCache = new Integer[MAX_CACHE - ↵
    MIN_CACHE + 1];
```

```
305:     public static Integer valueOf(int val)
306:     {
307:         if (val < MIN_CACHE || val > MAX_CACHE)
308:             return new Integer(val);
309:         synchronized (intCache)
310:         {
311:             if (intCache[val - MIN_CACHE] == null)
312:                 intCache[val - MIN_CACHE] = new Integer(val);
313:             return intCache[val - MIN_CACHE];
314:         }
315:     }
```

Mivel a cache-nek a mérete -128 és 127 közé esik, így ha az objektumunk ebből a tartományból vesz fel értéket akkor az Integer osztály valueof(int val) függvénye fog lefutni. Tehát az előlteszítős ciklusunknak a feltétele egyből teljesül és kilép. Viszont, ha ezen a tartományon kívül veszünk fel értéket akkor false eredményt kapunk és végtelen ciklusunk lesz. Ez egy memória megtakarító és teljesítmény növelő funkció, amit autoboxingnak neveznek.



The screenshot shows a Sublime Text interface. On the left is the file tree with 'untitled' and 'Gagyi.java'. The main editor window contains the following Java code:

```
1  public class Gagyi{
2
3     public static void main(String[] args) {
4
5         Integer x = 128;
6         Integer t = 128;
7         while (x == t && x >= t && t != x)
8             { System.out.println("Gagyi"); }
9         System.out.println("Gagyi vége");
10    }
11 }
```

To the right of the editor is a terminal window titled 'nemesis@nemesis-Aspir...'. It shows the output of the Java program, which is printing the word 'Gagyi' repeatedly until it reaches a value where x is no longer equal to t. The terminal window has three tabs, all labeled 'nemesis@nemesis-Aspir...'. The status bar at the bottom indicates 'Line 6, Column 20' and 'Tab Stop: 4'.

12.4. ábra. Végtelen ciklus 128-nál

The screenshot shows a Linux desktop environment with a terminal window and a code editor. The terminal window, titled 'nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/Prog2/Gagyi\$', displays the command 'javac Gagyi.java' followed by the output 'nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/Prog2/Gagyi\$ java Gagyi'. The code editor, titled 'Gagyi.java', contains the following Java code:

```
1 public class Gagyi{  
2     public static void main(String[] args) {  
3         Integer x = 127;  
4         Integer t = 127;  
5         while (x <= t && x >= t || t != x) {  
6             System.out.println("Gagyi");  
7             System.out.println("Gagyi vége");  
8         }  
9     }  
10 }  
11 }
```

12.5. ábra. Leállás 127-nél

12.4. Yoda

Megoldás videó:

Megoldás forrása:

A Yoda condition a nevét a Starwars karakteréről kapta, aki elég egyedi angolnyelvi szintaxisával is kitűnt a sorozatból ez az egyedi szintaxis az volt, hogy az alany->ige->tárgyat felcserélte tárgy->alany->igére. A Yoda condition lényege az, hogy egy ekvivalenciánál a konstanst nem a megszokott jobb oldalára írjuk az ekvivalenciának, hanem a bal oldalra. Ezt általában stringeknél használjuk. Tehát nem az objektumunkat hasonlítsuk össze a stringgel, hanem a stringet az objektumunkkal. Az előnye a feltétel betartásának az, hogy elkerüljük a NullPointerException-t. Viszont a hátránya az, hogy a kód olvasását nehezebbé teszi. Ugyanis ilyenkor jobbról balra érdemesebb olvasnunk a feltételeket. Viszont ezen kívül még, megkímélhetjük magunkat az == helyett = operátor hibától.

12.5. Kódolás from scratch

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13. fejezet

Helló, Liskov!

13.1. 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/)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.2. Szülő-gyerek

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.3. Anti OO

A BBP algoritmussal 4 a Pi hexadecimális kifejtésének a 0. pozíciótól számított 10 6, 107, 108 darab jegyét határozzuk meg C, C++, Java és C# nyelveken és vessük össze a futási időket! <https://www.tankonyvtar.hu/hu/tartanitok-javat/apas03.html#id561066>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.4. Hello Android!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

13.5. Ciklomatikus komplexitás

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14. fejezet

Helló, Mandelbrot!

14.1. Reverse engineering UML osztálydiagram

UML osztálydiagram rajzolása az első védési C++ programhoz. Az osztálydiagramot a forrásokból generáljuk (pl. Argo UML, Umbrello, Eclipse UML) Mutassunk rá a kompozíció és aggregáció kapcsolatára a forráskódban és a diagramon, lásd még: https://youtu.be/Td_nlERIEOs. <https://arato.inf.unideb.hu/batfai.norbert/UD.html> (28-32 fólia)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.2. Forward engineering UML osztálydiagram

UML-ben tervezünk osztályokat és generálunk belőle forrást!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.3. Egy esettan

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.4. BPMN

Rajzoljunk le egy tevékenységet BPMN-ben! <https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog34-47.pdf> (34-47 fólia)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

14.5. TeX UML

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15. fejezet

Helló, Chomsky!

15.1. Encoding

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.2. OOCWC Lexer

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.3. Paszigráfia Rapszódia OpenGL full screen vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, textúrázás, a szintek jobb elkülönítése, kézreállóbb irányítás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.4. Paszigráfia Rapszódia LuaLaTeX vizualizáció

Lásd vis_prel_para.pdf! Apró módosításokat eszközölj benne, pl. színvilág, még erősebb 3D-s hatás.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

15.5. Perceptron osztály

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16. fejezet

Helló, Stroustrup!

16.1. JDK osztályok

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.2. Másoló-mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.3. 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.

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.4. Változó argumentumszámú ctor

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

16.5. Összefoglaló

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17. fejezet

Helló, Gödel!

17.1. Gengszterek

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.2. C++11 Custom Allocator

<https://prezi.com/jvvbytkwgsxj/high-level-programming-languages-2-c11-allocators/> a CustomAlloc-os példa, lásd C forrást az UDPORG repóban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.3. STL map érték szerinti rendezése

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.4. Alternatív tabella rendezése

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.5. Prolog családfa

Ágyazd be a Prolog családfa programot C++ vagy Java programba! Lásd para_prog_guide.pdf!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

17.6. GIMP Scheme hack

Ha az előző félévben nem dolgoztad fel a témat (például a mandalás vagy a króm szöveges dobozosat) akkor itt az alkalom!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18. fejezet

Helló, !

18.1. 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>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.2. OOCWC Boost ASIO hálózatkezelése

Mutassunk rá a scanf szerepére és használatára! <https://github.com/nbatfai/robocar-emulator/blob/master/-justine/rcemu/src/carlexer.ll>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.3. SamuCam

Mutassunk rá a webcam (pl. Androidos mobilod) kezelésére ebben a projektben: <https://github.com/nbatfai/Samu>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.4. BrainB

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

18.5. OSM térképre rajzolása 6

Debrecen térképre dobunk rá cuccokat, ennek mintájára, ahol én az országba helyeztem el a DEAC hekkereket: <https://www.twitch.tv/videos/182262537> (de az OOCWC Java Swinges megjelenítőjéből: <https://github.com/emulator/tree/master/justine/rcwin> is kiindulhatsz, mondjuk az komplexebb, mert ott időfejlődés is van...)

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19. fejezet

Helló, Schwarzenegger!

19.1. Port scan

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.2. AOP

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

19.3. Junit teszt

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20. fejezet

Helló, Calvin!

20.1. MNIST

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.2. Deep MNIST

Mint az előző, de a mély változattal. Segítő ábra, vesd össze a forráskóddal a https://arato.inf.unideb.hu/batfai.norb_8. fóliáját!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.3. Android telefonra a TF objektum detektálója

Telepítsük fel, próbáljuk ki!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.4. SMNIST for Machines

Készíts saját modellt, vagy használj meglévőt, lásd: <https://arxiv.org/abs/1906.12213>

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

20.5. Minecraft MALMO-s példa

A <https://github.com/Microsoft/malmo> felhasználásával egy ágens példa, lásd pl.: <https://youtu.be/bAPSu3Rndl8>, https://bhaxor.blog.hu/2018/11/29/eddig_csaltunk_de_innentol_mi, https://bhaxor.blog.hu/2018/10/28/minecraft_

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

21. fejezet

Helló, Berners Lee!

21.1. C++ és Java

Ebben a fejezetben a Java és C++ nyelv összefüggéseit és különbségeit fogjuk vizsgálni. Már a könyv elején is említik, hogy a Java a jelölésrendszerében nagyon sok minden átvett a C++-ból. Az előzőleges tudásunkból pedig tudjuk, hogy a C++ eljárás és objektum orientált nyelv, míg a Java már szimplán az objektum orientált szemléletmódot követi. Az objektum orientált programozás célja, hogy implementálja a valós-világ egyedeit. Az objektum a valós világ egyedeire utal. Míg az objektum orientált programozás egy paradigma arra, hogy olyan programot írunk, amely osztályokat és objektumokat használ. Az objektumoknak vannak tulajdonságai és van viselkedésük. A tulajdonságokat változókkal írjuk le általában, míg a viselkedést a metódusokkal jellemizzük. Az objektumok lehetnek fizikai vagy logikai dolgok. Emellett az objektum orientáltságánál fontos megemlíteni az öröklődést. Amikor az egyik objektum örökli minden tulajdonságát és viselkedését a szülőobjektumától. A Java-ban ezen kívül nagy figyelmet fordítottak a biztonságra és a megbízhatóságra. Ebből következik az, hogy itt már nincsenek pointerek minden referencia. A Javában interpretált használnak, míg a C++-ban compiler végzi a fordítást. A C++ fordító gépi kódra fordítja a programot. Addig a Java fordítóprogramja egy byte kódot hoz létre, amelyet a JVM futtat. Ezért a Java platform független. Míg azért a C++-nál vannak megkötések. A Java program objektumok és ezek blueprintjeinek összessége. Az osztály változókból és metódusokból épül fel. A JVM hátránya a sebesség.(lassabb mint a compiler)

21.2. Python

IV. rész

Irodalomjegyzék

DRAFT

21.3. Általános

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

21.4. C

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

21.5. C++

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

21.6. 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 érdeklődésükért és hasznos észrevételeikért.

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

Mindemellett külön köszönet illeti azokat akik segítették, hogy ez a könyv megvalósulhasson.