

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. május 5, v. 0.0.5

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. május 8.

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.Attila

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	10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	12
2.6. Helló, Google!	14
2.7. 100 éves a Brun téTEL	16
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	32
3.8. Deklaráció	33

4. Helló, Caesar!	37
4.1. double ** háromszögmátrix	37
4.2. C EXOR titkosító	38
4.3. Java EXOR titkosító	39
4.4. C EXOR törő	40
4.5. Neurális OR, AND és EXOR kapu	40
4.6. Hiba-visszaterjesztéses perceptron	42
5. Helló, Mandelbrot!	43
5.1. A Mandelbrot halmaz	43
5.2. A Mandelbrot halmaz a std::complex osztállyal	44
5.3. Biomorfok	45
5.4. A Mandelbrot halmaz CUDA megvalósítása	46
5.5. Mandelbrot nagyító és utazó C++ nyelven	50
5.6. Mandelbrot nagyító és utazó Java nyelven	51
6. Helló, Welch!	53
6.1. Első osztályom	53
6.2. LZW	55
6.3. Fabejárás	59
6.4. Tag a gyökér	61
6.5. Mutató a gyökér	70
6.6. Mozgató szemantika	78
7. Helló, Conway!	79
7.1. Hangyszimulációk	79
7.2. Java életjáték	80
7.3. Qt C++ életjáték	82
7.4. BrainB Benchmark	82
8. Helló, Schwarzenegger!	84
8.1. Szoftmax Py MNIST	84
8.2. Mély MNIST	85
8.3. Minecraft-MALMÖ	86

9. Helló, Chaitin!	87
9.1. Iteratív és rekurzív faktoriális Lisp-ben	87
9.2. Gimp Scheme Script-fu: króm effekt	87
9.3. Gimp Scheme Script-fu: név mandala	89
10. Helló, Gutenberg!	91
10.1. Programozási alapfogalmak	91
10.2. Programozás bevezetés	93
10.3. Programozás	94
III. Második felvonás	98
11. Helló, Arroway!	100
11.1. A BPP algoritmus Java megvalósítása	100
11.2. Java osztályok a Pi-ben	100
IV. Irodalomjegyzék	101
11.3. Általános	102
11.4. C	102
11.5. C++	102
11.6. Lisp	102

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	17
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	35
4.1. Memória példa	37
4.2. Alsó háromszög mátrix	38
4.3. Exortörés	40
4.4. Signum függvény	41
5.1. Mandelbrothalmaz	44
5.2. Mandelbrot halmaz komplex osztállyal	45
5.3. Biomorf	46
5.4. Mandel CUDA	50

5.5. Mandelbrot nagyító	51
5.6. Java Mandelbrot nagyító	52
6.1. Polargen random	55
6.2. Inorder bejárás	60
6.3. Preorder bejárás	60
6.4. Postorder bejárás	61
7.1. Hangyaszimuláció	80
7.2. UML osztálydiagramm	80
7.3. Sejtautomata	81
7.4. Életjáték	82
7.5. BrainB benchmark	83
8.1. Softmax mnist	85
9.1. Nemesis króm effektel	88
9.2. Nemesisborder króm effektel	89
9.3. Név mandala	90

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

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;
}
```

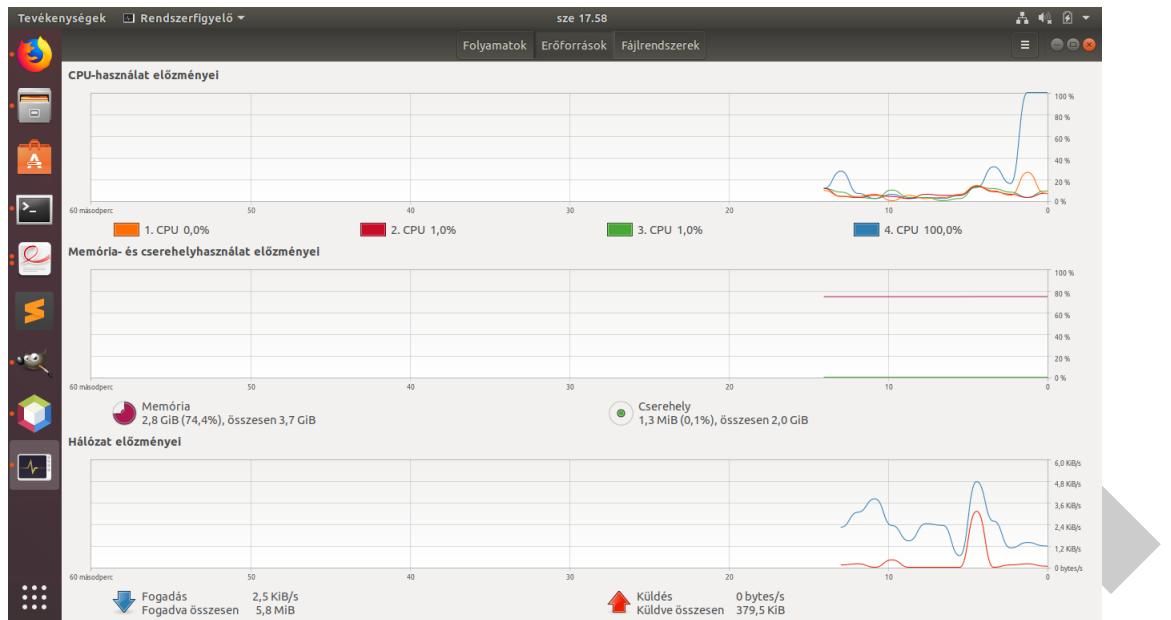
A végtelen ciklus következhet hibából, de van úgy, hogy szándékosan használunk végtelen ciklusokat például a programok menüjénél, de a program futása is egy végtelen ciklus, amelyet az X gombra kattintás szakít meg. Ha simán írunk egy végtelen ciklust az egy szálat fog kihasználni 100%-on, mindaddig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

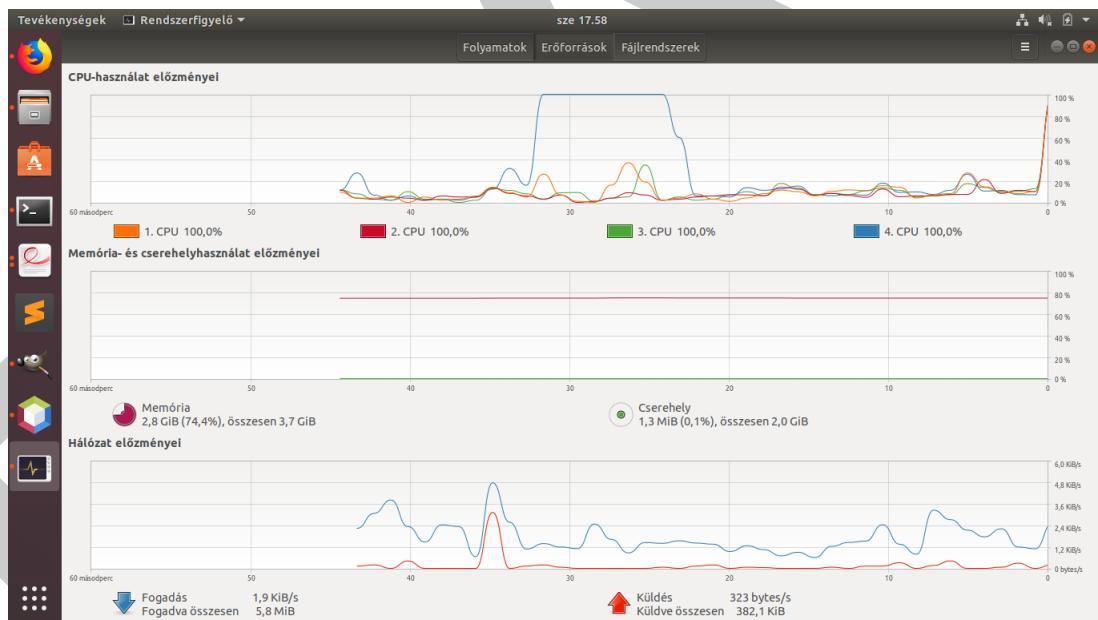
gcc vegtelen3.c -o vegtelen3 -fopenmp

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 végételen időre sleepeltethetünk. A sleep függvényt az:

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

könyvtár tartalmazza. Tehát a használatához meg kell hívnnunk.



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 T100asnak 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

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

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

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

Két változó értékének felcserélése többféle módon is történhet a legalapvetőbb a segédváltozó használatá. 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 also=5,masodik=3,temp;
    temp=also;
    also=masodik;
    masodik=temp;
}
```

De ezen a módszeren kívül, lehetséges összeadás-kivonással, szorzás-osztással, vagy logikai kizárával 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 kizárával vagyval is megcserélhetjük.

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

```
    else=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 kizáró vagyot végrehajtva 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.

```
int main()
{
    int a=5,b=3;
    printf("Elso változó:%d\n Második változó:%d\n",a,b );
    a=a^b;
    b=a^b;
    printf("Elso változó:%d\n Második változó:%d\n",a,b );
    a=a^b;
    b=a^b;
    printf("Elso változó:5\n Második változó:3\n");
    a=a^b;
    b=a^b;
    printf("Elso változó:3\n Második változó:5\n");
}
```

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.

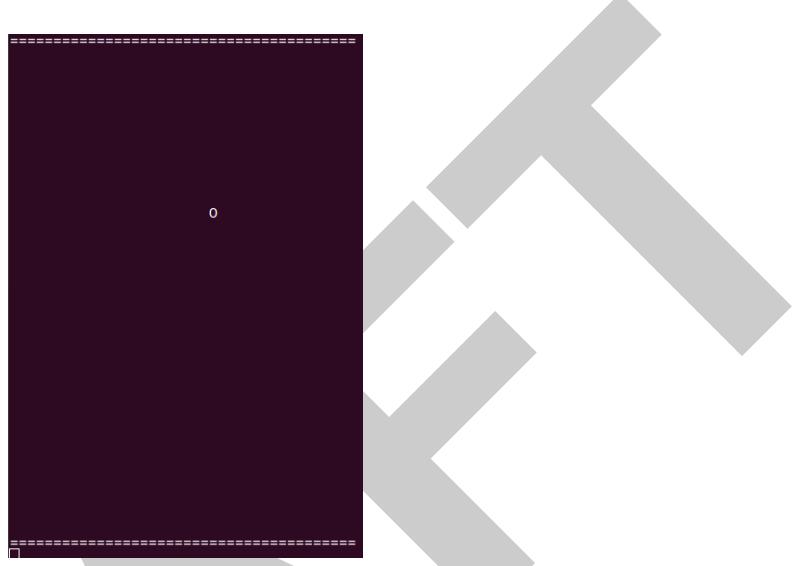
```
x=abs(szelesseg-lepteto%(2*szelesseg));
y=abs(tmagassag-lepteto%(2*tmagassag));
lepteto++;
```

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 koordi-

nátáját.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 függvényel 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.

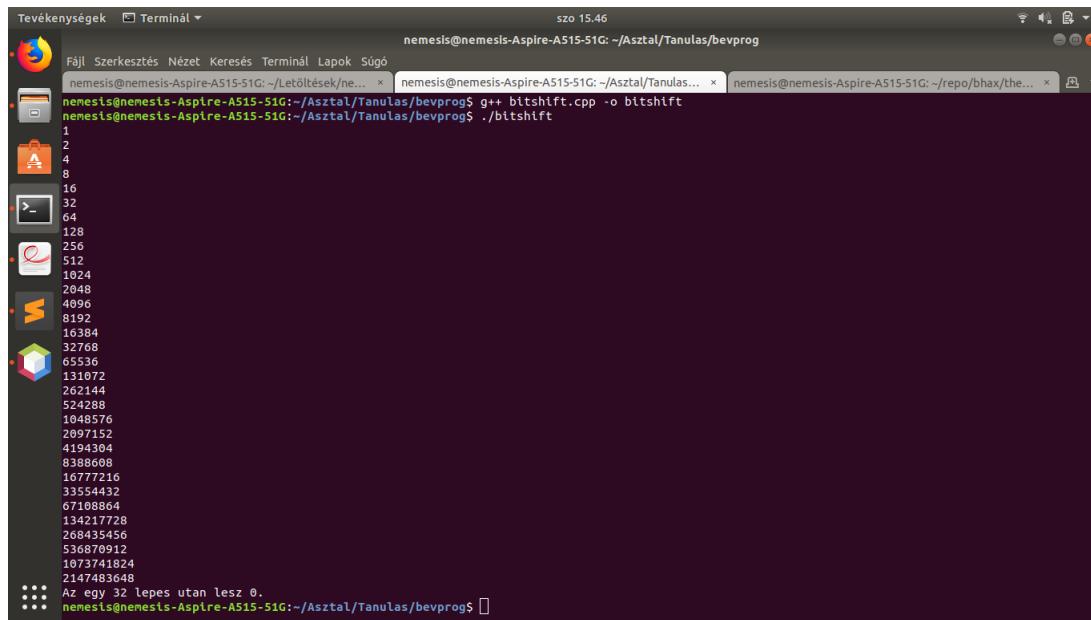
```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
2147483648
Az egy 31 lepes után lesz 0.
```

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);
```



```
Tevékenységek ┌ Terminál ┴
Fájl Szerkesztés Nézet Keresés Terminál Lapok Súgó
nemesis@nemesis-Aspire-A515-51G: ~/Letöltések/n... × nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/bevprog...
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
3355432
67108864
134217728
268435456
536870912
1073741824
2147483648
Az egy 32 lepes utan lesz 0.
nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/bevprog$
```

2.7. ábra. Szóhossz jó megoldás

2.6. Helló, Google!



Tutor

Ebben a feladatban tutoráltam Ádám Petrát és Duszka Ákos Attilát.

Ír 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 azaz a csillapító értéket, most ez 0.00001-ben lett meghatározva.

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

Ezt követően indítunk egy egybeágazott for ciklust, amely a letárolt kapcsolati gráfos tömb(a szétosztott 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 nevezzük azokat a számokat, amelyek csak 1-el és önmagukkal oszthatók. Ikerprímek azok a prímszámok, 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.

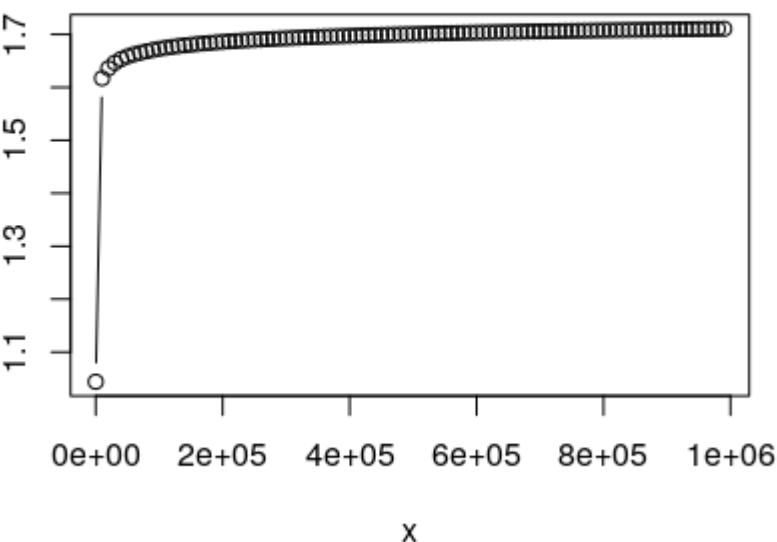
```
primes = primes(x)
diff = primes[2:length(primes)]-primes[1:length(primes)-1]
idx = which(diff==2)
t1primes = primes[idx]
t2primes = primes[idx]+2
```

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 bejárjuk 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.

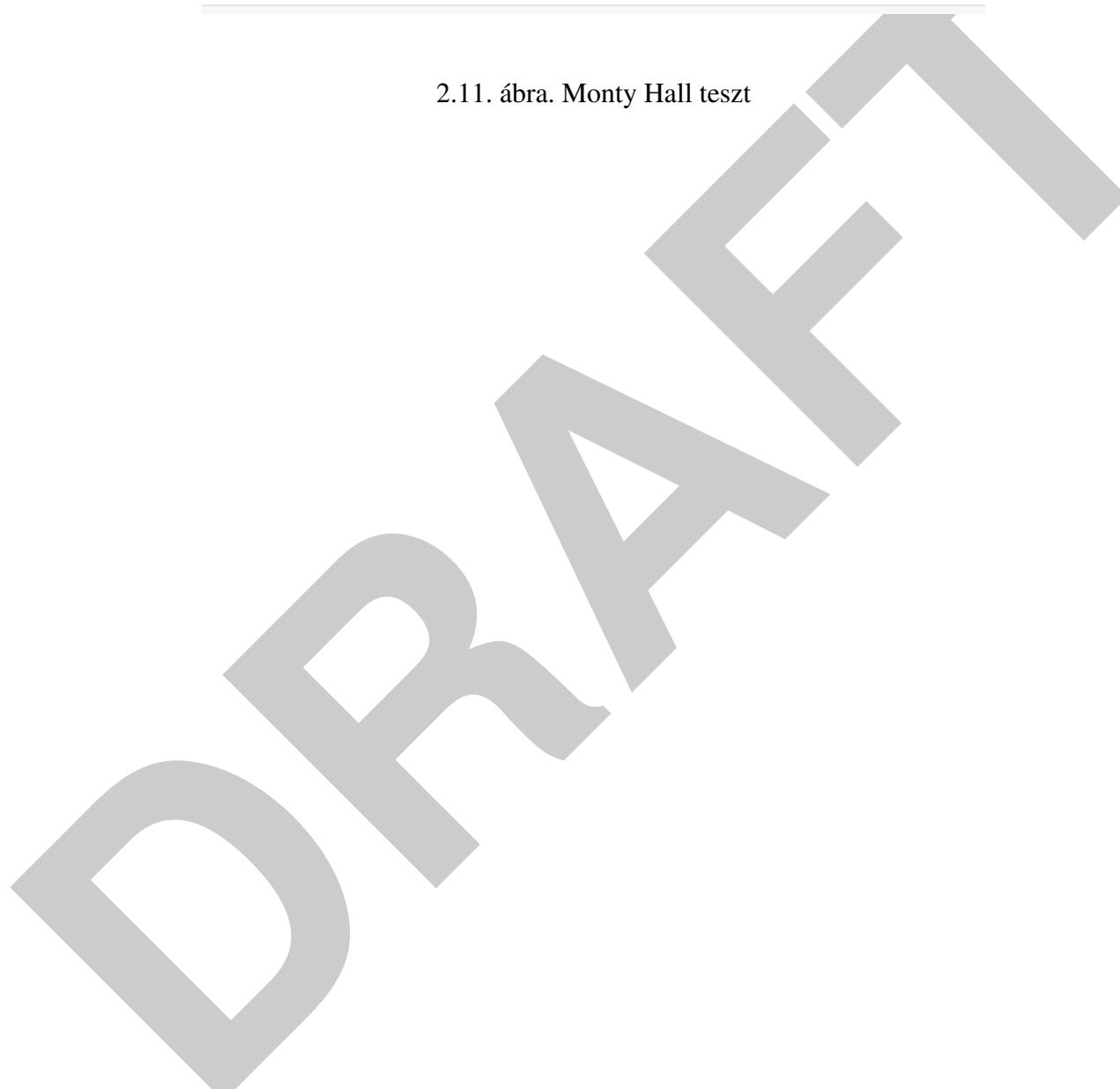
Létrehozunk egy új vektort amiben megváltoztatjuk a választást úgy, hogy kivételként adjuk a műsorvezető által kinyitott ajtót és a játékost által választottat.

Végül lefuttatunk egy ugyan ilyen feltételes vizsgálatot, hogy mi lett volna ha minden változtatunk. És itt is ugyan úgy letároljuk egy tömbbe, hogy mely indexeknél 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("Kiserletek szama: %i", kiserletek_szama)
[1] "Kiserletek szama: 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!

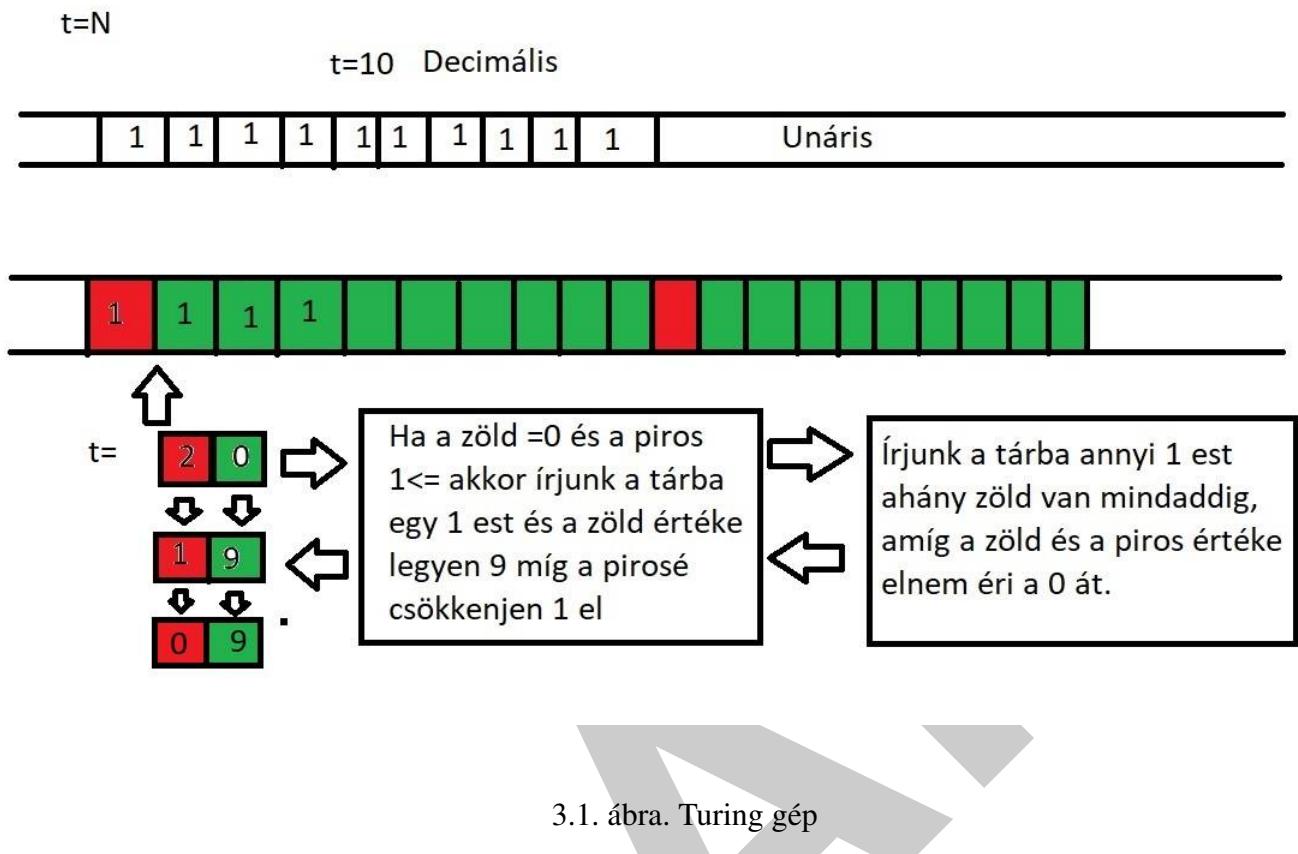
Megoldás videó:

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 grammata olyan szabályok összessége, amely segítségével a nyelvben minden jelsorozatot képesek vagyunk előállítani.

- A grammata 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-ből 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 gramatika 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 gramatika 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 definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

Megoldás videó:

Megoldás forrása: <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= ←
      IwAR1eImHN5PINqTMnhpJItlqA_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 ponterek férhetnek hozzá az adott memória területhez, ezeket a pontereket 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 jöhetsz 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 jönne 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 counter 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:

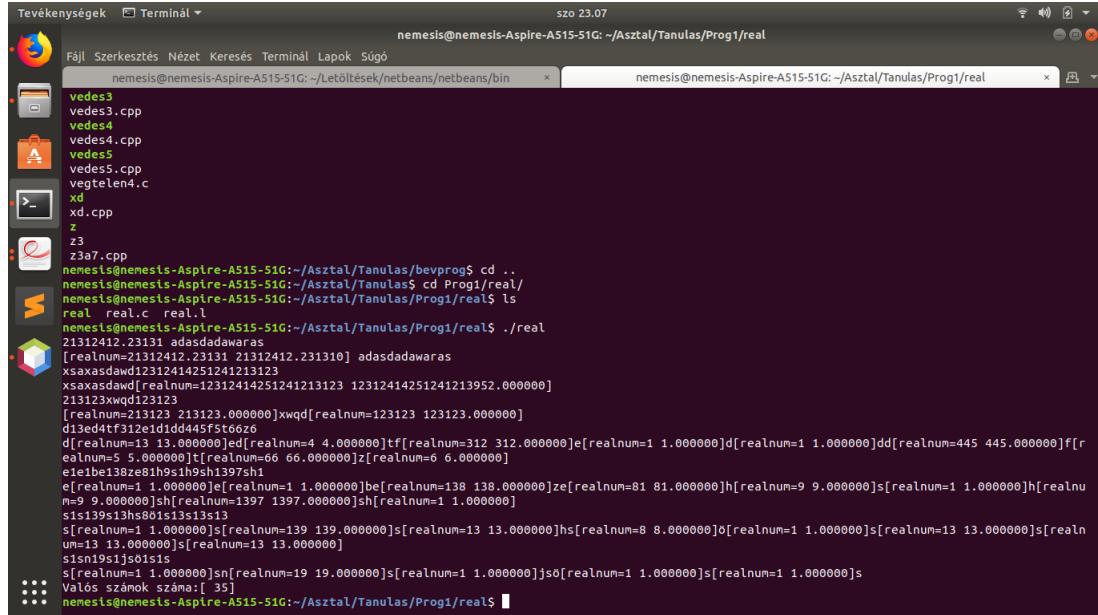
lex -o real.c real.l

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.



The screenshot shows a terminal window titled "Tevékenységek" with the sub-tab "Terminál". The window title bar says "szó 23.07". The terminal content shows a file browser on the left and a command-line session on the right. The command-line session starts with "nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/Prog1/real\$ cd .." followed by several lines of lexical analysis output. The output includes various tokens like "real", "real.c", "real.h", and many numerical values and identifiers such as "realnum=13.000000", "realnum=1.000000", and "realnum=1.000000". The session ends with "Valós számok száma: [35]" and "nemesis@nemesis-Aspire-A515-51G: ~/Asztal/Tanulas/Prog1/real\$".

3.3. ábra. Lexikális elemző

3.5. I33t.I

Lexelj össze egy I33t cipher!

Megoldás video:

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.

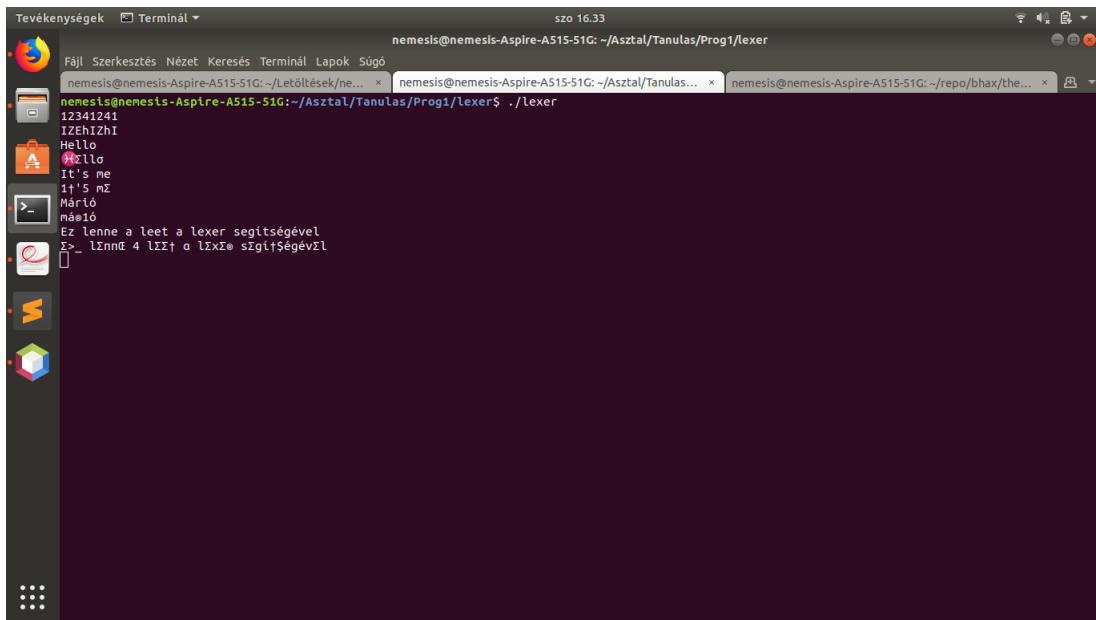
Erre a sorra kerül az első rész a lexernak ahol definiálunk és könyvtárakat hívunk meg.Most a definíciókat kihagyjuk.

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 forrás, 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 opció 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,ezért 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.



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++)
    tomb[i] = 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 video:

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: jelkezelo
    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 jelkezelo
```

A signalis program ignorálja(SIG_IGN) vagy másképp 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 hajtódik 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 hajtódkik 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 függvényvel 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 egészken 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})) $  
$ (\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 $y+2$ 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 ránkvetkező 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  
0x7fff06a097c  
0x7fff06a097c  
7  
7  
0x7fff06a097c  
0x7fff06a097c
```

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 mutató mutatót ad vissza.

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

Egy egészre mutató mutatóra 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 függvénnnyel.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.

DRAFT

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

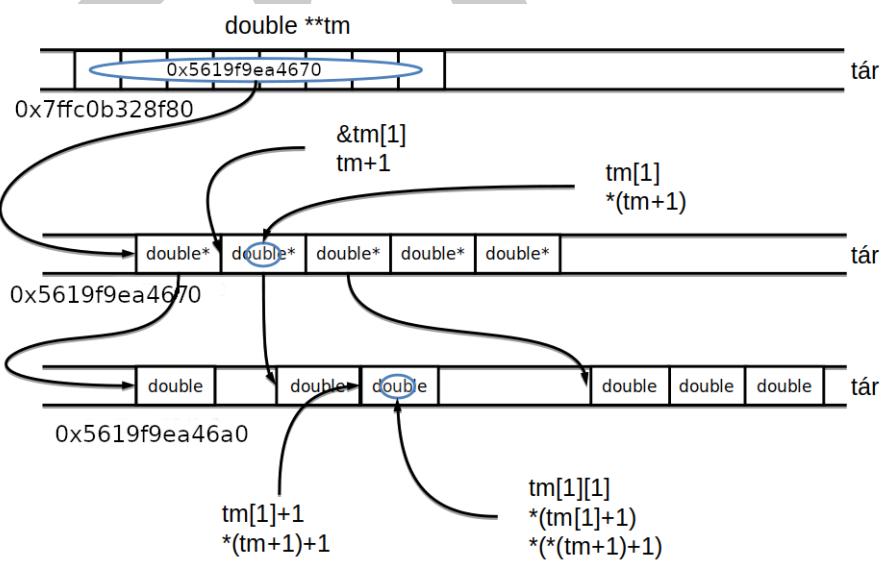
Írj egy olyan malloc és free párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

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 területtel. 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 mallocall 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 feltöljük a mátrixunkat elemekkel ez a forcikus 0-14 ig fogja feltölteni.

A következő értékkedésekkel kicseréljük 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$ cd Caesar/
nemesis@nemesis-Aspire-A515-51G: ~/repo/bhax/thematic_tutorials/bhax_textbook/Caesar$ ls
tm tm.c
nemesis@nemesis-Aspire-A515-51G: ~/repo/bhax/thematic_tutorials/bhax_textbook/Caesar$ ./tm
0x7ffc0b328f80
0x5019f9ea4a670
0x5019f9ea4a6a0
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
42.000000, 43.000000, 44.000000, 45.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
nemesis@nemesis-Aspire-A515-51G: ~/repo/bhax/thematic_tutorials/bhax_textbook/Caesar$
```

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 strcpy-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 beolvasott 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 definiálunk 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 minden 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/exor>

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:

4.3. ábra. Exortörés

4.5. Neurális OR, AND és EXOR kapu

**Tutor**

Ebben a feladatban tutoráltam Molnár Antalt.

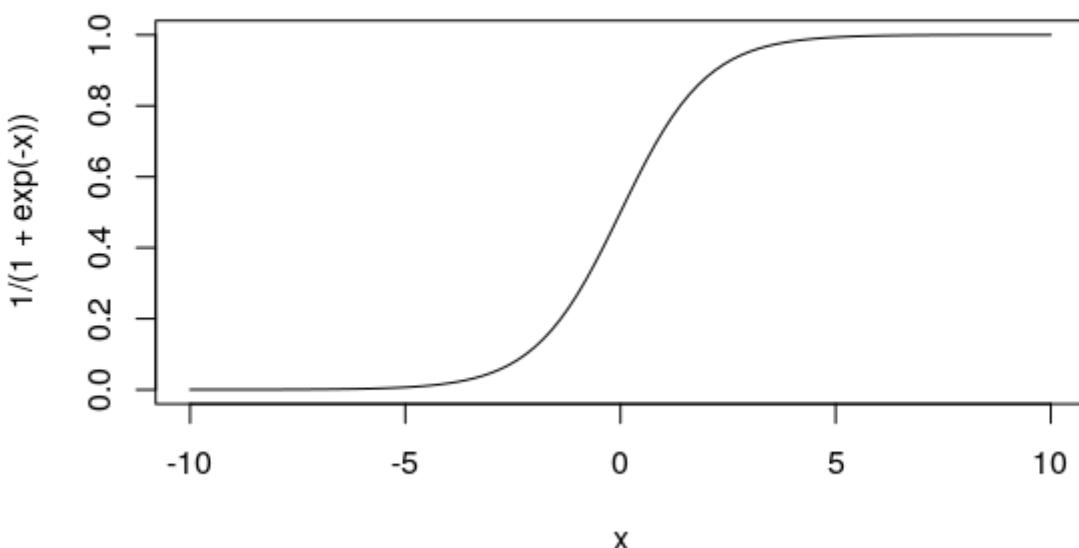
Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

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 függvényénél 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 küszöbfüggvény, 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 videó:

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

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 hibák kérthetők 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 sigmoid 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 határból majd frissítsük a súlyokat és újra megnézzük.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

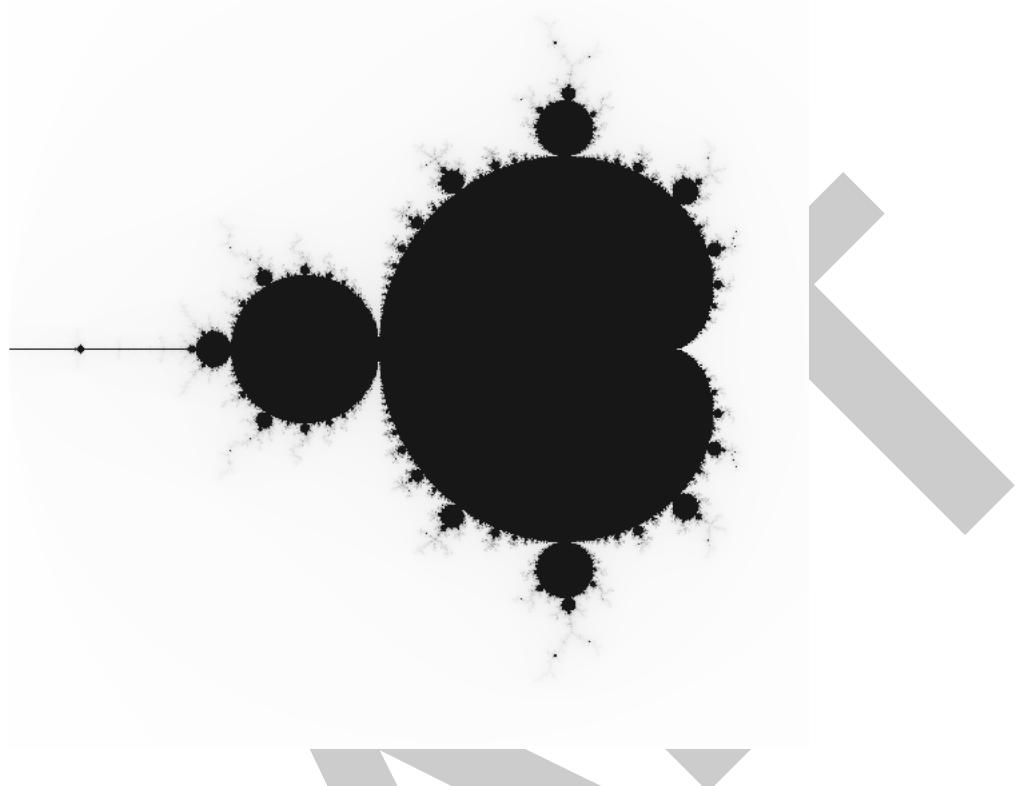
Megoldás videó:

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 feltöljük a tömböt. Majd létrehozunk egy új képet kép néven és pixelenként bejárjuk és ami benne van a halmazba 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 nagyítunk ismétlődésként megfog jelenni ez az ábra.



5.1. ábra. Mandelbrothalmaz

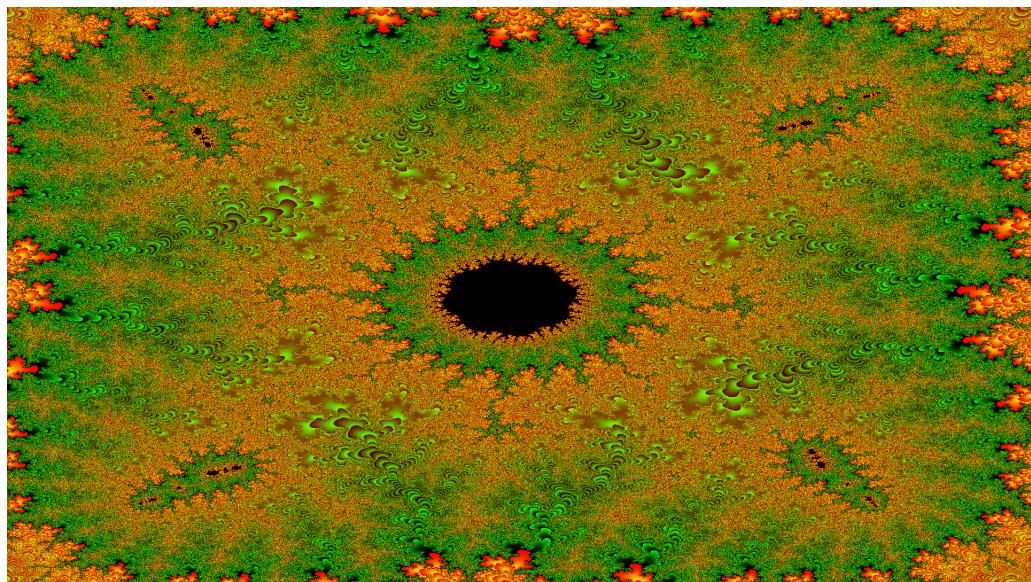
5.2. A Mandelbrot halmaz a `std::complex` osztályval

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 használatára.

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 képnél 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 elérteük az iterációs határt. Majd a while ciklus törzsében kiszámoljuk a z_n értékét és növeljük az iterációs határt. Majd ha kilép a while ciklusból a képnél 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 osztállyal

5.3. Biomorfok

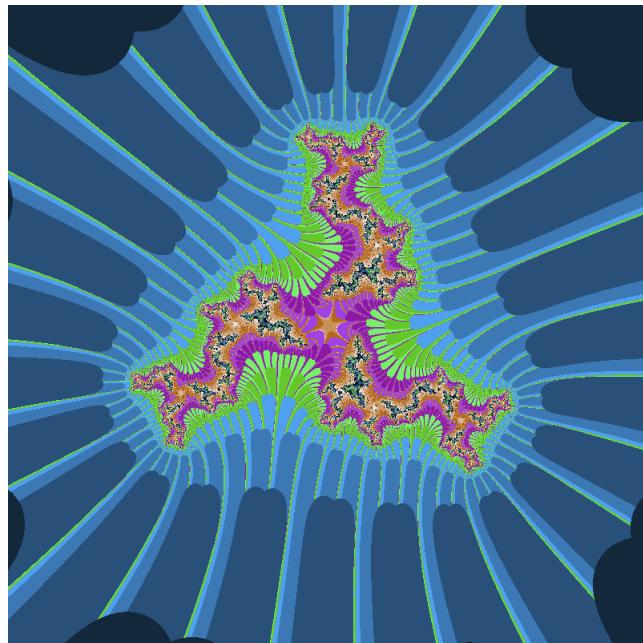
Megoldás videó: <https://youtu.be/IJMbgRzY76E>

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

Tanulságok, tapasztalatok, magyarázat...

A biomorfos 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 még az atof lebegőpontos számmá. Az előzőhöz képest ahol egy while-t futtattam, hogy eldönthetem mely elemeket tartoznak a halmazba és ez a feltétel az volt hogy abszolút értékbe a z komplex szám kisebb mint négy, vagy elérteuk -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 most már 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 biomorfos képek az egysejtűkre 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/types.h>
#include <iostream>

#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:
```

```
// most eppen a j. sor k. oszlopaban vagyunk

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

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

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

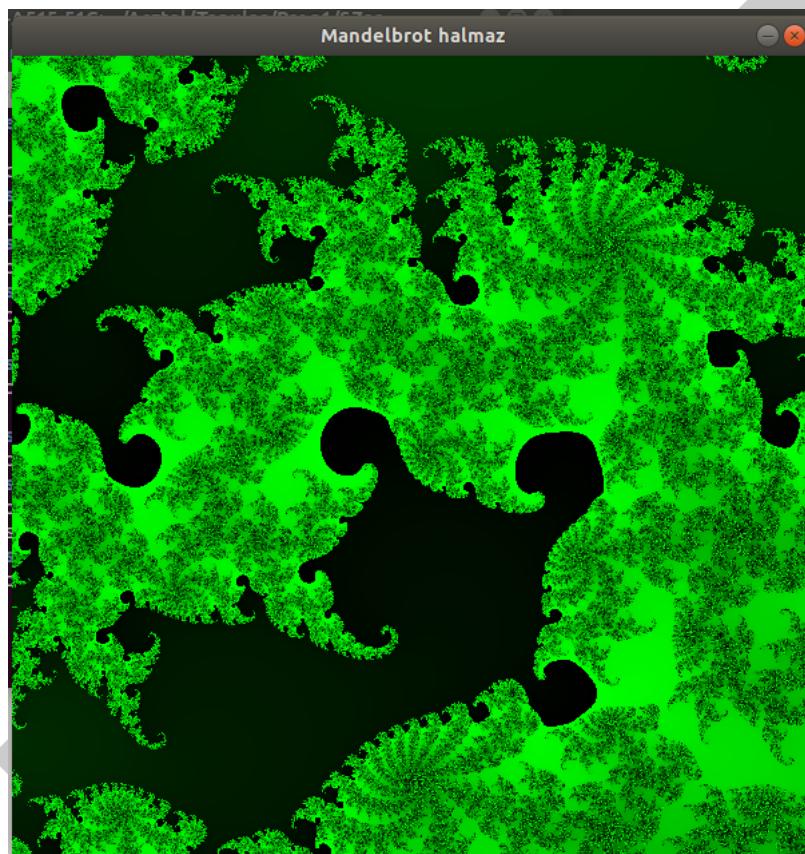
Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteráció bejárta z_n komplex számokat!

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 frakszal.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 tehát, 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 készereződik 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 osztáiban lévő értékeket. Az újra számoláshoz készítünk mindenkor minden számolás után update-eljük az osztáiban lévő értékeket. Az újra számoláshoz készítünk mindenkor minden számolás után update-eljük az osztáiban lévő értékeket.



5.5. ábra. Mandelbrot nagyító

5.6. Mandelbrot nagyító és utazó Java nyelven



Tutor

Ebben a feladatban tutoráltam Kun-Limberger Anettet 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 pozíóját 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 settile-vel 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 négyzet 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. 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=-0.5381235707885977  
b=-0.5381235597982536  
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 természetes neked!

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

    public PolarGen ()
    {
        nincsTarolt = true;
    }

    public double kovetkezo ()
    {
        if (nincsTarolt)
        {

            double u1, u2, v1, v2, w;
            do
            {

```

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

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

    tarolt = r * v2;
    nincsTarolt = !nincsTarolt;

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

}

public static void main (String[] args)
{
    PolarGen 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 gyökét 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 nincstaroltat negáljuk tehát az értéke true helyett false lesz. A visszatérési értéke a fv-nek r*v1 lesz.

Ha már van tárolt érték akkor pedig azzal tér vissza. Majd a main-be meghívjuk az osztályunkat pg néven és a forról létrehozunk 10 mintapéldányát az osztálynak. 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 orientáltságának 3 lényeges pontja van az egyik, hogy az összetartozó adatok képesek 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
Megnyitás  A
Random.java
k 9.54
Random.java
-/Letöltések/jdk/jdk-1.8.0_121/bin/java/util
Mentés

/*
 * Independent values at the cost of only 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
 * @serialField seed for random computations
 * @serialField nextNextGaussian double
 * @serialField next Gaussian to be returned
 * @serialField haveNextNextGaussian boolean
 */

```

6.1. ábra. Polargen random

6.2. LZW

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

Megoldás videó:

Megoldás forrása: <https://github.com/Savitar97/Prog1/blob/master/binfac/binfa.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 ertekek;

```

```
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 kiirl1 (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  
kiirl (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 eredményét, amely egy sikeres dinamikus memória foglalás az új elemnek.

A következő két eljársnál az externnel jelezük 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. Inordernél a csomópontok minden 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:

```
Tevékenységek Szövegszerkesztő ▾ k 8.10
Megnyitás ▾ Mentés
lzwtree.txt
-/Asztal/Tanulas/bevprog

-----1(3)
----1(2)
-----1(4)
----0(3)
-----0(4)
-----0(5)
-----0(6)
-1(1)
-----1(5)
----1(4)
----1(3)
-----1(5)
----0(4)
---0(2)
-----1(4)
----0(3)
-----0(4)
-----0(5)
-/(8)
-----1(5)
----1(4)
----1(3)
-----0(4)
-----0(5)
---1(2)
-----0(3)
-----1(5)
-----0(4)
-----0(5)
---0(1)
-----1(5)
-----1(4)
-----0(5)
-----1(3)
-----0(4)
-----0(5)
-----0(6)
0/21
```

6.2. ábra. Inorder bejárás

Preorder bejárás:

```
lzwtree.txt
k 8.08
-/Azott/Tanulas/bevprog

/(0)
---1(1)
-----1(2)
-----1(3)
-----0(3)
-----1(4)
-----0(4)
-----0(5)
-----0(6)
 0(2)
    -1(3)
    -----1(4)
    -----1(5)
    -----0(4)
    -----1(5)
    -0(3)
    -----1(4)
    -----0(4)
    -----0(5)
--0(1)
  -1(2)
  -----1(3)
  -----1(4)
  -----1(5)
  -----0(4)
  -----0(5)
  -0(3)
  -----0(4)
  -----1(5)
  -----0(5)
 0(2)
    -1(3)
    -----1(4)
    -----1(5)
    -----0(5)
    -0(4)
    -----0(5)
  n/a\
```

6.3. ábra. Preorder bejárás

Postorder bejárás:



6.4. ábra. Postorder bejárás

6.4. Tag a gyökér



Tutor

Ebben a feladatban tutoráltam Ádám Petrát.

Az LZW algoritmust ültess át egy C++ osztályba, legyen egy Tree és egy beágazott 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)
    {
    };
    ~Csomopont ()
    {
    };

    Csomopont *nullasGyermek () const
    {
        return balNulla;
    }

    Csomopont *egyesGyermek () const
    {
        return jobbEgy;
    }

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

    void ujEgyesGyermek (Csmopont * gy)
```

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

```
        szabadit (elem->nullasGyermek ());
        delete elem;
    }
}

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

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

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

```
    }

}

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 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 osztályból 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 (Csmopont * gy)
{
    jobbEgy = gy;
}

char getBetu () const
{
    return betu;
}

private:

char betu;
Csmopont *balNulla;
Csmopont *jobbEgy;

};

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

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

```
        }

}

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

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

};

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

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

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

    rszoras (gyoker);

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

    return szoras;
}

void
LZWBinFa::rmelyseg (Csmopont * elem)
```

```
{  
    if (elem != NULL)  
    {  
        ++melyseg;  
        if (melyseg > maxMelyseg)  
            maxMelyseg = melyseg;  
        rmelyseg (elem->egyesGyermek ());  
        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 konstruktor 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 gyökerből 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 destruktora.

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>

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 forrásaindról 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:

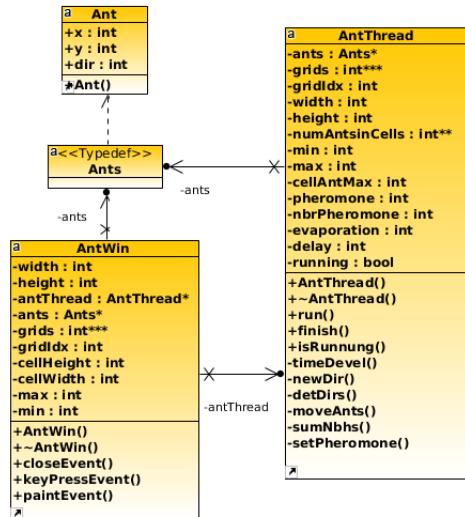
A hangyszimuláció a hangyák mozgását és nyomvonalát szimulálja, minden hangya feromon nyomat hagy, és ha feromon nyomra lép akkor elkezdi követni az előző hangya által hagyott feromont, a feromon kezdetben sötétzöld de egyre halványodik ha nem kezdi el követni egy újabb hangya. Minél erősebb egy feromon 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 kezelni. 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 által 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 pusztítsuk 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. Hangyszimuláció



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

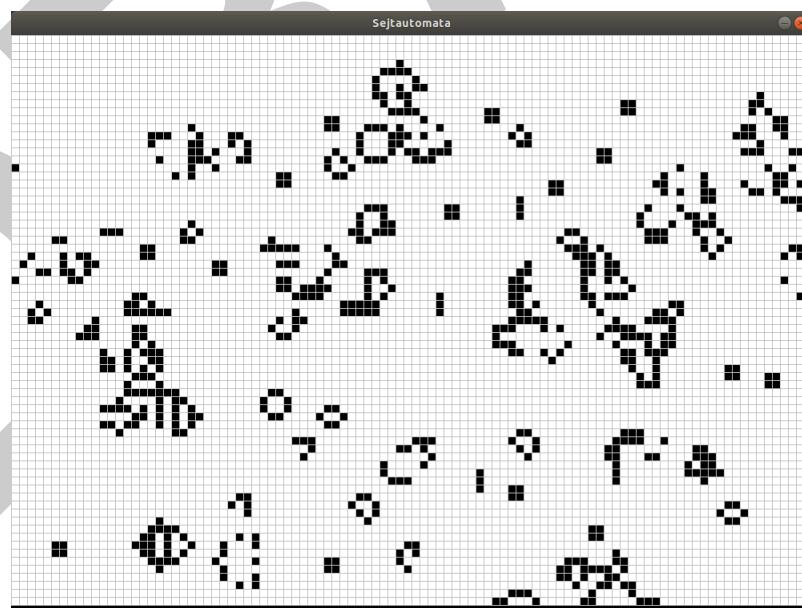
Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

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 váltsan 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 konstruktorban definiáljuk a sejtterünket, 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 cellákból 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 siklók 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

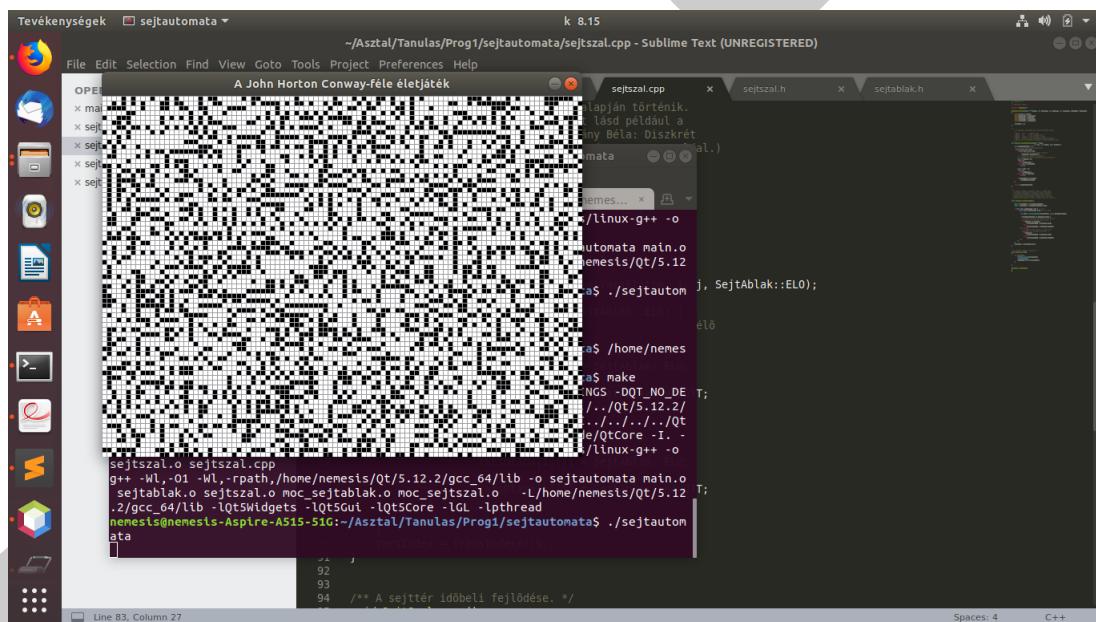
7.3. Qt C++ életjáték

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 skilő 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 véglőleg a populációt tudjuk befolyásolni. És ez egy olyan játék ahol a személy csak megfigyelője az eseményeknek nem pedig részese ugyanis miután beállítottuk a szabályait a világnak onnantól magától fut és teremtődnek az élőlények.



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 tartunk 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 tessük. De akkor is elég unalmas végig ülni, ami 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 konstruktőrben. 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 incCompanban 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 kérhetjük 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 köröt 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 dönti el a súlyokat, amely alapján dönt hogyan 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 teljesít 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 minden 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 eredményüktől. A GradientDescentOptimizer deriváltakat számít a hibahatárok figyelembe vételével. Ez a minimize.

```
nemesis@nemesis-Aspire-AS15-S1G:~/Asztal/Tanulas/Prog1/tensor$ python twicetwo.py
Extracting /tmp/tensorflow/mnist/input_data/train-labels-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-labels-idx3-ubyte.gz
Extracting /tmp/tensorflow/mnist/input_data/t10k-images-idx3-ubyte.gz
2019-05-07 19:04:00.212630: 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. tesztképeinek felismerése, mutatva a számot, a továbblepeshez csukd be az ablakat
- Ezt a halozat ennek ismeri fel: 4
-----
-- A MNIST 11. tesztképeinek felismerése, mutatva a számot, a továbblepeshez csukd be az ablakat
- Ezt a halozat ennek ismeri fel: 8
-----
nemesis@nemesis-Aspire-AS15-S1G:~/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 amelyivel 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/bAPSu3Rndl8>

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

Valószínűleg sokan ismerik a MineCraft játéket 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 kiírjuk, 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.

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó: <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 definiálását 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 videó: 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 kapunk. 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 állíjuk 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

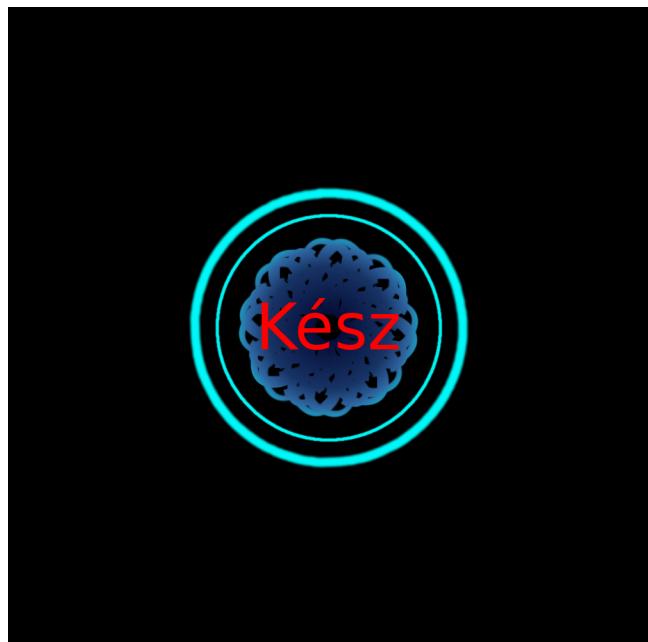
9.3. Gimp Scheme Script-fu: név mandala

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 függvénnnyel 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 szélességen 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 pixelekkel 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 vékonyabb 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 kóddá, amelyet a processzor feltud dolgozni. A fordító program végzi tehát a kód szemantikai, szintaktikai, lexikális vizsgálatát és a kód generálását a szemantikai hibát nem mindig 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 segítségével 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 vezéri 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 kapok, hogy egy másik típus tartományát szűkítem le ők az altípusok. Adattípusok csoportja lehet skalár vagy struktúrált. Egyszerű típusok: egész, valós, karakteres, logikai. Összetett típusok: tömb (értekei csak egyfél 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. Értékadó 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 minden képp 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 mindig 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 zárjuk. Az utasítás blokkat {}-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 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 minden azelőtt deklarálni egy változót mielőtt felhasználunk ezzel átláthatóbb a kód.

Függvénynevek túlterhelése: A C-ben a függvényeket a neve azonosította, így nem lehetett két ugyan azzal 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 elé egy és jelet teszünk, így a változó memória címét adjuk át a függvénynek tehát ha valamelyen módosítás történik a változával 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 adatszerkezetnél 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. Típusátámogatás az osztályok támogathatnak operátorokat és típuskonverziót. A struktúráknak nem csak tagváltozói, hanem tagfüggvényei is lehetnek. A tagváltozók megnevezése attribútumok, míg a függvényeké 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 megkülönböztetni 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 paramétereit 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 osztályt több osztály is felhasználhat ezért a .h-fájlban a #ifndef és #define segítségével 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 delet-hez is hozzá kell írni a szögletes zárójelet ha tömböt akarunk felszabadítani. Ha Fifo adatszerkezetet akarunk használni akkor ha új elemet akarunk hozzáadni akkor egygel nagyobb területet kell foglalnunk majd a végére beszúrni 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 bitenkénti 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 megkell 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 között 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 másolatokat 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 elé tesszük az átalakítani kívánt változó neve előtt. 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 kizártak 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 átalakítást. Ha nem írunk 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 operánszer a biztonság szempontjából leállítja a programot.

A C++ a típuskonverzióra új teret nyit ezek a castok. Ez lehet statikus vagy dinamikus vagy konstans, esetleg újraértelmező 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 kiírja 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, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

DRAFT

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

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

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

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

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