

Univerzális programozás

Így neveld a programozód!

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Ács Bátfai, Margaréta	2019. május 16.	

REVISION HISTORY

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

Ajánlás

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

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

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	4
2. Helló, Turing!	6
2.1. Végtelen ciklus	6
2.2. Lefagyott, nem fagyott, akkor most mi van?	7
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	10
2.6. Helló, Google!	11
2.7. 100 éves a Brun tétel	12
2.8. A Monty Hall probléma	12
3. Helló, Chomsky!	13
3.1. Decimálisból unárisba átváltó Turing gép	13
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	13
3.3. Hivatkozási nyelv	14
3.4. Saját lexikális elemző	15
3.5. l33t.1	16
3.6. A források olvasása	19
3.7. Logikus	20
3.8. Deklaráció	20

4. Helló, Caesar!	23
4.1. double ** háromszögmátrix	23
4.2. C EXOR titkosító	25
4.3. Java EXOR titkosító	26
4.4. C EXOR törő	27
4.5. Neurális OR, AND és EXOR kapu	29
4.6. Hiba-visszaterjesztéses perceptron	30
5. Helló, Mandelbrot!	31
5.1. A Mandelbrot halmaz	31
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	34
5.3. Biomorfok	36
5.4. A Mandelbrot halmaz CUDA megvalósítása	36
5.5. Mandelbrot nagyító és utazó C++ nyelven	40
5.6. Mandelbrot nagyító és utazó Java nyelven	43
6. Helló, Welch!	47
6.1. Első osztályom	47
6.2. LZW	48
6.3. Fabejárás	53
6.4. Tag a gyökér	57
6.5. Mutató a gyökér	61
6.6. Mozgató szemantika	65
7. Helló, Conway!	67
7.1. Hangyaszimulációk	67
7.2. Java életjáték	67
7.3. Qt C++ életjáték	67
7.4. BrainB Benchmark	74
8. Helló, Schwarzenegger!	77
8.1. Szoftmax Py MNIST	77
8.2. Mély MNIST	80
8.3. Minecraft-MALMÖ	81

9. Helló, Chaitin!	82
9.1. Iteratív és rekurzív faktoriális Lisp-ben	82
9.2. Gimp Scheme Script-fu: króm effekt	82
9.3. Gimp Scheme Script-fu: név mandala	82
10. Helló, Gutenberg!	83
10.1. Programozási alapfogalmak	83
10.2. Programozás bevezetés	83
10.3. Programozás	83
III. Második felvonás	85
11. Helló, Arroway!	87
11.1. A BPP algoritmus Java megvalósítása	87
11.2. Java osztályok a Pi-ben	87
IV. Irodalomjegyzék	88
11.3. Általános	89
11.4. C	89
11.5. C++	89
11.6. Lisp	89

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogyan lássuk mást 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.xsl
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 találsz az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk?

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: <http://esr.fsf.hu/hacker-howto.html>!
- Olvasgasd aztán 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
- C kapcsán a [KERNIGHANRITCHIE] könyv adott részei.
- C++ kapcsán a [BMECPP] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a [The GNU C Reference Manual](https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf), mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: <https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf>
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [BMECPP] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, <https://www.imdb.com/title/tt0478087/>, benne a **Monty Hall probléma** bemutatása.
- Kódjátzsma, <https://www.imdb.com/title/tt2084970>, benne a **kódtörő feladat** élménye.

- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

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:

```
int main() {  
while(1) {  
  
sleep();  
};  
  
}
```

```
int main() {  
while(1) {  
  
};  
  
}
```

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

```
int main() {
```

```
#pragma omp parallel
```

```
while(1) {
```

```
};  
  
}
```

-vegtelen_0.c Egy szálat futtatunk 0%-on, ezt úgy érjük el, hogy a szálat a sleep paranccsal "elaltatjuk", azaz leállítjuk. -vegtelen_100.c Egy szálat futtatunk 100%-on, a while a zárójelben lévő egyes miatt mindig igaz lesz, ezért folyamatosan futni fog. -vegtelen_a.c Az open MP segítségével az összes szála futtatni párhuzamosan, ugyan azt az elvet, amit az egy szál 100%-on történő futtatásakor használtunk.

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 építő 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...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

```
#include <stdio.h>

int a= 13;
int b= 9;
```

```
int main(){

printf("%d\n", a);
printf("%d\n", b);


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


printf("%d\n", a);
printf("%d\n", b);

}
```

A-nak és b-nek megadunk 2 egész értékű változót, majd kiiratjuk az a és b változókat eredeti alakjukban. Ezek után a két változót összeadjuk, és az összegből kivonjuk az eredeti változókat, majd megint kiiratjuk a változókat, de mostmár felcserélt állapotban.

2.4. Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írd egy olyan programot, ami egy labdát pattogtat a karakteres konzolon! (Hogy mit értek pattogtatás alatt, alább láthatod a videón.)

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

Megoldás forrása:

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

int
main ( void )
{
    WINDOW *ablak;
    ablak = initscr ();

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;
```

```
int mx;
int my;

for ( ;; ) {

    getmaxyx ( ablak, my , mx );

    mvprintw ( y, x, "O" );

    refresh ();
    usleep ( 100000 ); //was 100000

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

    if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
        xnov = xnov * -1;
    }
    if ( x<=0 ) { // elerte-e a bal oldalt?
        xnov = xnov * -1;
    }
    if ( y<=0 ) { // elerte-e a tetejet?
        ynov = ynov * -1;
    }
    if ( y>=my-1 ) { // elerte-e a aljat?
        ynov = ynov * -1;
    }

}

return 0;
}
```

A curses headerrel kezeljük az ablakot melyben pattog a labdánk. Az x és y alap értékének 0-t állítunk be, majd ezeket az x- és ynov paranccsal növeljük az értékét. A végtelen ciklusban a getmaxyx meghatározza a az ablak maximum x és y értékeit, míg az mvprintw kiírja a labdát. A refresh parancs frissíti az ablakot, a usleep késlelteti, a labda mozgását. Az ifek ellentétes irányba fordítják a labda útvonalát, ha eléri a labda a max értéket.

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:

Tanulságok, tapasztalatok, magyarázat...

2.6. Helló, Google!

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

```
#include <stdio.h>
#include <math.h>
void
kiir (double tomb[], int db)
{
    int i;
    for (i = 0; i < db; ++i)
        printf ("%f\n", tomb[i]);
}
double
tavolsag (double PR[], double PRv[], int n)
{
    double osszeg = 0.0;
    int i;
    for (i = 0; i < n; ++i)
        osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);
    return sqrt(osszeg);
}
int
main (void)
{
    double L[4][4] = {
        {0.0, 0.0, 1.0 / 3.0, 0.0},
        {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
        {0.0, 1.0 / 2.0, 0.0, 0.0},
        {0.0, 0.0, 1.0 / 3.0, 0.0}
    };
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
    double PRv[4] = { 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0 };
    int i, j;
    for (;;)
    {
        for (i = 0; i < 4; ++i)
        {
            PR[i] = 0.0;
            for (j = 0; j < 4; ++j)
                PR[i] += (L[i][j] * PRv[j]);
        }
        if (tavolsag (PR, PRv, 4) < 0.00000001)
            break;
        for (i = 0; i < 4; ++i)
            PRv[i] = PR[i];
    }
    kiir (PR, 4);
    return 0;
}
```

Megoldás videó:

A PageRank nevű algoritmust arra használjuk, hogy weboldalak fonrosságát tudjuk kiszámolni. A lényege, hogy véletlenszerű böngészések során mekkora az esélye annak, hogy a keresett oldalra találunk. Egy oldalon, minden feltüntetett link, egy "pont" az adott oldalnak. Ezeket alapul véve mondjuk meg az oldal relatív fontosságát, hogy hány oldalra mutató hivatkozás van a többi weboldalon, illetve, hogy hány oldalra megy hivatkozás az oldalunkról. Az algoritmus során, egy népszerűbb oldal által adott "pont" többet számít, mint egy kevésbé népszerű oldalé. Azt kijelenthetjük, hogy nem fogja mindenki végignézni az összes linket az adott oldalon, ezért egy enyhítés is van a képletben.

```
if (tavolsag(PR, PRv , 4) < 0.00000001)
break;
```

A fenti végtelen ciklust megtalálhatjuk a programunkban, ez azt jelenti, hogyha a megkapott távolságunk kisebb, mint 0.00000001 akkor a program kilép a ciklusból.

A mutatók rendszerezésének érdekében mátrixokat használunk. Azt, hogy mi mutat mire, 4x4-es mátrixok oszlopai és sorai szemléltetik számunkra. Például, ha az első sor első oszlopa egyest tartalmaz, akkor a z első sor eleme mutat az első oszlop elemére.

A program futása után 4 értéket kapunk majd, amik a várt Pagerank értékek lesznek.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat...

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

A turinggép olvas egy számot, ami most 10. Kivon egyet a decimális számunk utolsó számjegyéből majd eltávolítja az egyest. Mivel az utolsó helyen nem 0 áll ezért átváltja 0 helyett 9-re. Ezek után az első számjegyből is kivon egy egyest és azt is tárolja. Addig ismétli ezt a feladatsort, míg minden helyi értékben 0-át nem látunk. A tárolt egyesek sorozata lesz a jelen esetben unáris 10-esünk. Tutor: Rémiás Dávidot

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

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

Megoldás videó:

Megoldás forrása:

1. A, B, C „változók” a, b, c „konstansok”
 $A \rightarrow aAB, A \rightarrow aC, CB \leftrightarrow \rightarrow bCc, cB \rightarrow Bc, C \rightarrow bc$
 S -ből indulunk ki \leftarrow
 $\text{-----} A (A \rightarrow aAB) aAB (A \rightarrow aC) aaCB (\leftarrow$
 $CB \rightarrow bCc) aabCc (C \rightarrow bc) aabbcc$
Révész könyv, 13. o. (Bev. a \leftarrow
form. nyelvek elméletébe, Akadémiai Kiadó, 1979)
 $A (A \rightarrow aAB) \leftarrow$
 $aAB (A \rightarrow aAB) aaABB (A \rightarrow aAB) aaaABBB (A \rightarrow aC) aaaaCBBBB (\leftarrow$
 $CB \rightarrow bCc) aaaabCcBB (cB \rightarrow Bc) aaaabCBcB (cB \rightarrow Bc) aaaabCBBc (\leftarrow$
 $CB \rightarrow bCc) aaaabbCcBc (cB \rightarrow Bc) aaaabbCBcc (CB \rightarrow bCc) \leftarrow$
 $aaaabbbCccc (C \rightarrow bc) aaaabbbbcccc$
2. S, X, Y „változók” a, b, c „konstansok”
 $S \rightarrow abc, S \rightarrow aXbc, \leftarrow$
 $Xb \rightarrow bX, Xc \rightarrow Ybcc, bY \rightarrow Yb, aY \rightarrow aaX, aY \rightarrow aaS$ -ből \leftarrow
indulunk ki $\text{-----} S (S \rightarrow aXbc) aXbc (Xb \leftarrow$
 $\rightarrow bX) abXc (Xc \rightarrow Ybcc) abYbcc (bY \rightarrow Yb) aYbbcc (aY \rightarrow aa) \leftarrow$
 $aabbcc$
Révész könyv, 12. o. (Bev. a form. nyelvek elméletébe, \leftarrow

Akadémiai Kiadó, 1979S ($S \rightarrow aXbc$) $aXbc$ ($Xb \rightarrow bX$) $abXc$ ($Xc \leftrightarrow \rightarrow Ybcc$) $abYbcc$ ($bY \rightarrow Yb$) $aYbbcc$ ($aY \rightarrow aaX$) $aaXbbcc$ ($Xb \rightarrow bX$) \leftrightarrow $aabXbcc$ ($Xb \rightarrow bX$) $aabbXcc$ ($Xc \rightarrow Ybcc$) $aabbYbcc$ ($bY \rightarrow Yb$) \leftrightarrow $aabYbbccc$ ($bY \rightarrow Yb$) $aaYbbbccc$ ($aY \rightarrow aa$) $aaabbbccc$

$P_1XP_2 \rightarrow P_1QP_2$, P_1, P_2 eleme $(VN \cup VT)^*$, $X \in VN$ belüli, $Q \in (VN \cup VT) \leftrightarrow$ $+ \text{belüli}$, kivéve $S \rightarrow \text{üres}$, de akkor S nem lehet jobb oldali \leftrightarrow egyetlen szabályban sem

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:

```
#include <stdio.h>

int main ()
{
    // Print string to screen.
    printf ("Hello World\n");
return 0;
}
// C89-C99 es hiba / pipa
```

Meghatározhatjuk adott műveletek végrehajtási sorrendjét, a nyelv vezérlésátadó utasításainak segítségével.

Az utasításfajtáinkban több típusa is van a C programnyelvben.

Például ezek:

```
*kifejezés utasítás
*összetett utasítás
*iterációs utasítás
*vezérlésátadó utasítás
*kiválasztó utasítás
*cimkézett utasítás
```

Kifejezés :

```
x = 0, i++ vagy printf(...)
x=0;
i++;
printf(...)
```

Ha pontosvesszőt írunk utána utasítássá válik, ez zárja le az utasításokat. A kapcsos zárójelek használatával, az adott deklarációk és utasítások csoportját foglaljuk össze egy összetett blokkba. Ez szintaktikailag egyenértékű egyetlen utasítással.

If-Else: döntések kifejezésére használjuk.

```
if \ \(kifejezés)
  \ 1.utasítás
else
  \ 2.utasítás
```

Az else rész mindig opcionális, nem kötelező. Igaz kiértékelés esetén az első megadott utasítást hajtjuk végre, hamis esetén az else fog lefutni. Az else, mindig az első, hozzá legközelebb álló else nélküli ifhez csatlakozik. Ha nem ezt szeretnénk, a kívánt összerendelés kapcsos zárójelek használatával érhető el.

```
if (n > 0) {
    if (a > b)
        z = a;
}
else
    z = b;
```

Az Else-If utasítás:

```
if \ \(kifejezés)
  \ utasítás
else if (kifejezés)
  \ utasítás
else if (kifejezés)
  \ utasítás
else if \ \(kifejezés)
  \ utasítás
.
.
else
  \ utasítás
```

Ezzel adhatunk meg többszörös döntéseket. A gép sorra kiértékeli a kifejezéseket és ha bármelyik ezek közül igaz, a programban folyamatosan kiértékelődésre kerülnek a kifejezések és ha bármelyik igaz ezek közül, a megfelelő utasítás hajtódik végre, majd befejezi a keresési láncot.

A Switch utasítás:

A switch utasítással kezeltjük a többirányú programágakat. Úgy működik az utasítás, hogy összehasonlítja egy kifejezés értékét több egész értékű állandó kifejezés értékével, hogy az egyik kifejezés értékér összehasonlítja több egész értékű konstans kifejezés értékével, és ennek megfelelően hajtja végre az utasítást.

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 használjunk, azaz óriások

vállán álljunk és ne kispályázzunk!

Megoldás videó:

Megoldás forrása:

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

Deklarálunk egy `real_numbers` változót, ez megszabja, hogy a számjegyeink 0 és 9 között vehetnek fel értéket. Ezek után a lexerünk eldönti (`{digit}*({digit}+)?`)-t alkalmazva, hogy valós-e az adott szám. Ha valós a szám a `real_numbers` értéke nő 1-gyel. A mainben meghívjuk a lexert és kiíratjuk a valós számok számát.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: `/* Fordítás: $ lex -o l337d1c7.c l337d1c7.l Futtatas: $ gcc l337d1c7.c -o l337d1c7 -lfl` (kilépés az input vége, azaz Ctrl+D)

Copyright (C) 2019

Norbert Bátfai, batfai.norbert@inf.unideb.hu

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program. If not, see <<https://www.gnu.org/licenses/>>.

```

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

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

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

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

    {'0', {"D", "0", "D", "0"}},
    {'1', {"I", "I", "L", "L"}},
    {'2', {"Z", "Z", "Z", "e"}},
    {'3', {"E", "E", "E", "E"}},
    {'4', {"h", "h", "A", "A"}},
    {'5', {"S", "S", "S", "S"}},
    {'6', {"b", "b", "G", "G"}},
    {'7', {"T", "T", "j", "j"}},

```

```
{'8', {"X", "X", "X", "X"}},
{'9', {"g", "g", "j", "j"}}

// https://simple.wikipedia.org/wiki/Leet
};

%}
%%
. {

    int found = 0;
    for(int i=0; i<L337SIZE; ++i)
    {

        if(l337d1c7[i].c == tolower(*yytext))
        {

            int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

            if(r<91)
                printf("%s", l337d1c7[i].leet[0]);
            else if(r<95)
                printf("%s", l337d1c7[i].leet[1]);
            else if(r<98)
                printf("%s", l337d1c7[i].leet[2]);
            else
                printf("%s", l337d1c7[i].leet[3]);

            found = 1;
            break;
        }

    }

    if(!found)
        printf("%c", *yytext);

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

Egy leet táblázat alapján kicserélünk karaktereket leetekre. Minden karakterhez 4 leetet fűzünk, ezt tömbökbe osztjuk, azért hogy könnyebben tudjunk hivatkozni ezekre. Az *r* változó eldönti, hogy a karakter 4 lett értéke közül melyik kerüljön kiírásra. Generál egy számot 0 és 100 között véletlenszerűen és ezt az

értéket veszi fel. Az értéktől függően kapjuk a leet[0], leet[1], leet[2] vagy leet[3] értéket. Tutor: Rémiás Dávid

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezeslo)==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 jelkezeslo 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 meggyránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

1. Minden x -re létezik nagyobb y amely prím. 2. Minden x -re létezik nagyobb y amely prím, s a rákövetkezőjének a rákövetkezője is prím. 3. Létezik y minden x -re ahol ha x prím, akkor kisebb, mint y . 4. Létezik y minden x -re ahol ha y kisebb, mint x , akkor x nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája
- egészek tömbje
- egészek tömbjének referenciája (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:

```
int a;
int *b = &a;
int &r =a;
int c[5];
int (&tr) [5] = c;
int *d[5];
int *h ();
int *(*j ());
int (*i(*v (int e))) (int f, int g);
```

</para>

<para>

```
int a; = egész
int *b=&a; = egészre mutató mutató
int &r=a; =egész referenciája
int c[5]; = egészek tömbje
int (&tr)[5] =c; =egészek tömbjének referenciája
int *d[5]; = egészre mutató mutatók tömbje
int *h(); = egészre mutató mutatót visszaadó függvény
int *(*l()); = egészre mutató mutatót visszaadó függvényre ←
mutatót mutató (bár l-es helyett egy betű használata jobb ←
választás volna)
int (*v(int c)) (int a, int b); = egészet visszaadó és két ←
egészet kapó függvényre mutató mutatót visszaadó, egészet ←
kapó függvény
```

```
int ((*z) (int)) (int, int); = függvénymutató egy egészet ↵  
    visszaadó és két egészet kapó függvényre mutató mutatót ↵  
    visszaadó, egészet  
                                kapó függvényre ( az "int"-ek ↵  
                                után hiányzik az elnevezés)
```

Tutor: Rémiás Dávid

DRAFT

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
#include <stdlib.h>
int
main ()
{
    int rs = 5;
    double **tb;

    printf("%p\n", &tb);

    if ((tb = (double **) malloc (rs * sizeof (double *))) == NULL)
    {
        return -1;
    }
    printf("%p\n", tb);

    for (int i = 0; i < rs; ++i)
    {
        if ((tb[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL) ←
        {
            return -1;
        }
    }
    printf("%p\n", tb[0]);

    for (int i = 0; i < rs; ++i)
        for (int j = 0; j < i + 1; ++j)
            tb[i][j] = i * (i + 1) / 2 + j;
```



```
for (int i = 0; i < rs; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tb[i][j]);
    printf ("\n");
}
tb[3][0] = 42.0;
(*(tb + 3))[1] = 43.0;
*(tb[3] + 2) = 44.0;
*(*(tb + 3) + 3) = 45.0;
for (int i = 0; i < rs; ++i)
{
    for (int j = 0; j < i + 1; ++j)
        printf ("%f, ", tb[i][j]);
    printf ("\n");
}
for (int i = 0; i < rs; ++i)
    free (tb[i]);
free (tb);
return 0;
}
```

rs=5

Létrehoztuk a `rs = 5` -el a sorainknak a számát amire szükségünk van.

```
double **tb
```

A `double **tb` deklarálunk és lefoglalunk a memóriában 8 byte-ot jelen esetben. Majd szimplán egy kiíratást végzünk. A `%p` segítségével egy hexadecimális számot fogunk ki írni, mely megadja a pointer számát. Ebben a sorban a vessző után megfigyelhető, hogy egy olyan operátor segítségével melyet címkézésre használunk, 1 bájt címét fogjuk ki íratni.

```
if ((tb = (double **) malloc (rs * sizeof (double *))) == NULL)
```

A `malloc` adunk egy pointert a lefoglalt területre, majd szabad területen memóriát foglalunk. Kap egy paramétert ami az `rs` függvényében zajlik. (5) Következőkben látjuk, hogy a `sizeof (double*)` megmondja hogy hány byte kell a `double*` típusnak, ami a feladatban 40, vagyis `5x8` helyet foglal. Ha bármilyen hiba merülne fel akkor pedig a program kilép. Ezt biztosítjuk a `==NULL` szekcióban. A `NULL` nem mutat sehova. A folytatásban megkreáljuk az 5 sorból álló háromszögünket.

```
tb[3][0] = 42.0;
(*(tb + 3))[1] = 43.0;
*(tb[3] + 2) = 44.0;
*(*(tb + 3) + 3) = 45.0;
```

`tb[3][0] = 42.0` A harmadik sor első elemét egyenlővé tesszük 42.00-val. Az alatta lévő sorban, a második, harmadik, majd negyedik elemre hivatkozunk. A `"*"` al hivatkozok azon a címen tárolt pointerre. Tökéletesen reprezentálja a példa mily módon és hányféleképpen tudunk hivatkozni. A `for` ciklusukhoz

különösképpen nem tudunk kiegészítést írni . A Chomsky fejezetünkben a hivatkozással példánál kiveséztük nagy részét és a lényeg megtalálható benne ami ezen feladat megértéséhez kell.Amit esetleg meglehet említeni a `tb[i][j]` ahol az "i" a sor a "j" pedig az oszlopot jelzik.Fontos hogy vizsgáljuk a kapcsos zárójelek elhelyezkedését,hisz úgy fogjuk meg érteni mikor mi hajtódik végbe.A `%f` egy float típusú változót fog vissza adni.A végén a `free`-ről kell pár sort írunk.Lényegében felszabadítja a memóriában tárolt/lévő pointerok által lefoglalt címeket.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define MAX_KULCS 100
#define BUFFER_MERET 256
int
main (int argc, char **argv)
{
    char kulcs[MAX_KULCS];
    char buffer[BUFFER_MERET];
    int kulcs_index = 0;
    int olvasott_bajtok = 0;
    int kulcs_meret = strlen (argv[1]);
    strncpy (kulcs, argv[1], MAX_KULCS);
    while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
    {
        for (int i = 0; i < olvasott_bajtok; ++i)
        {
            buffer[i] = buffer[i] ^ kulcs[kulcs_index];
            kulcs_index = (kulcs_index + 1) % kulcs_meret;
        }
        write (1, buffer, olvasott_bajtok);
    }
}
```

Definiáljuk , hogy a `MAX_KULCS = 100` A `BUFFER_MERET`-et `256`-al tesszük egyenlővé Megadja, hogy hány karakter tud egyszerre beolvasni. Az `argc` megmondja, hogy a program hány szóval fordítottam le. az `argv` tárolja az argumentumainkat. A `strlen`, amely a `string.h`-ban megtalálható, megmondja, hogy hány karakterből áll a stringünk. A `while` ciklusban a `read` segítségével végigolvassuk a file-t egészen addig, amíg a file vége jellel nem találkozik a program. A `write` kiírja az olvasott byte-okat. A `for` ciklusunkon belül a byte-ok számát nulláról indítjuk, majd mindig növeljük eggyel. A cikluson belül a magban a `buffer` "i"-edik elemét exorozzuk az adott kulcsindex-szel. A kulcs indexhez hozzá adunk egyet majd maradékosztást végzünk rajta.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcsSzöveg,
        java.io.InputStream bejövőCsatorna,
        java.io.OutputStream kimenőCsatorna)
        throws java.io.IOException {

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;
        while((olvasottBájtok =
            bejövőCsatorna.read(buffer)) != -1) {

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

                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

            new ExorTitkosító(args[0], System.in, System.out);

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <string.h>
double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
    int sz = 0;
    for (int i = 0; i < titkos_meret; ++i)
        if (titkos[i] == ' ')
            ++sz;
    return (double) titkos_meret / sz;
}
int
tisztalehet (const char *titkos, int titkos_meret)
{
    // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
    // illetve az atlagos szohossz vizsgálatával csökkentjük a
    // potenciális töréseket
    double szohossz = atlagos_szohossz (titkos, titkos_meret);
    return szohossz > 6.0 && szohossz < 9.0
        && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
        && strcasestr (titkos, "az") && strcasestr (titkos, "ha");
}
void
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{
    int kulcs_index = 0;
    for (int i = 0; i < titkos_meret; ++i)
    {
        titkos[i] = titkos[i] ^ kulcs[kulcs_index];
        kulcs_index = (kulcs_index + 1) % kulcs_meret;
    }
}
int
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
            int titkos_meret)
{
    exor (kulcs, kulcs_meret, titkos, titkos_meret);
    return tisztalehet (titkos, titkos_meret);
}
```

```
}
int
main (void)
{
    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;
    // titkos fajt berantasa
    while ((olvasott_bajtok =
        read (0, (void *) p,
            (p - titkos + OLVASAS_BUFFER <
                MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
        p += olvasott_bajtok;
    // maradék hely nullazása a titkos bufferben
    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\\0';
    // összes kulcs eloallitasa
    for (int ii = '0'; ii <= '9'; ++ii)
        for (int ji = '0'; ji <= '9'; ++ji)
            for (int ki = '0'; ki <= '9'; ++ki)
                for (int li = '0'; li <= '9'; ++li)
                    for (int mi = '0'; mi <= '9'; ++mi)
                        for (int ni = '0'; ni <= '9'; ++ni)
                            for (int oi = '0'; oi <= '9'; ++oi)
                                for (int pi = '0'; pi <= '9'; ++pi)
                                    {
                                        kulcs[0] = ii;
                                        kulcs[1] = ji;
                                        kulcs[2] = ki;
                                        kulcs[3] = li;
                                        kulcs[4] = mi;
                                        kulcs[5] = ni;
                                        kulcs[6] = oi;
                                        kulcs[7] = pi;
                                        if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
                                            printf
                                                ("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
                                                    ii, ji, ki, li, mi, ni, oi, pi, titkos);
                                        // újra EXOR-ozunk, így nem kell egy második buffer
                                        exor (kulcs, KULCS_MERET, titkos, p - titkos);
                                    }
    return 0;
}
```

Az `atlagos_szohossz`-on belül láthatjuk a `const char *titkos`-t ami egy későbbiekben meg nem változtatható mutató. Beállítom hova mutasson a pointer és úgy is marad. A `for` cikluson belül az "i" 0-tól indul egészen míg a `titkos_meret`-et el nem éri. Az `if`-en belül megszámloljuk a karaktereket szavanként. Ha space jelenséget észlel tehát az "i"-edik eleme szóköz akkor növeljük az "sz"-t. Ami a "titkos" fakkba fog menni. A `return`el pedig szimplán átlagot számolunk. Elosztjuk a `titkos_meret`-et

az sz szóközök számával.

A `tiszta_lehet` függvény ahogy a komment szekcióban is írjuk a programon belül , a potenciális töréseket csökkenthetjük. Jelen esetben be is állítjuk, hogy az átlagos szóhossz 5 és 9 közés esik , a leggyakoribb magyar szavaink pedig , hogy - az - nem - ha , melyeket az `strcasestr` a `strstr()`-el ellentétben megvizsgálja ezeknek az előfordulását és rögzítik is azt." Tű a szénakazalban " ahogyan a manuál is írja. A lényege pedig , hogy a `szohossz`-ban tároljuk az `atlagos_szohossz` által kapott értékeket.

4.5. Neurális OR, AND és EXOR kapu

R

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

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

```
# Copyright (C) 2019 Dr. Norbert Bاتفai, nbاتفai@gmail.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program. If not, see <http://www.gnu.org/licenses/>
#
# https://youtu.be/Koyw6IH5ScQ
library(neuralnet)
a1 <- c(0,1,0,1)
a2 <- c(0,0,1,1)
OR <- c(0,1,1,1)
or.data <- data.frame(a1, a2, OR)
nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE, ←
  stepmax = 1e+07, threshold = 0.000001)
plot(nn.or)
compute(nn.or, or.data[,1:2])
a1 <- c(0,1,0,1)
a2 <- c(0,0,1,1)
OR <- c(0,1,1,1)
AND <- c(0,0,0,1)
orand.data <- data.frame(a1, a2, OR, AND)
nn.orand <- neuralnet(OR+AND~a1+a2, orand.data, hidden=0, linear.output= ←
  FALSE, stepmax = 1e+07, threshold = 0.000001)
plot(nn.orand)
compute(nn.orand, orand.data[,1:2])
```

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

Ez az R program egyszerű logikai műveletekre tanítja a gépünket egy neutrális hálót használva. Mindig adunk egy mintát, ez alapján a számítógép egy adott súlyt ad értékként, majd elvégzi a műveleteket elvégzi és ezt adja meg értéknek. Csak közelítő értékeket kaphatunk. (pl.: igaz érték esetén 0.9998 érték, míg hamis érték esetén 0,01) Tutor: Rémiás Dávid

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

```
// mandelpngt.c++
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternosztér/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↵
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
//
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>
```



```
#define MERET 600
#define ITER_HAT 32000
void
mandel (int kepadat[MERET][MERET]) {
    // MÉRÜNK IDŐT (PP 64)
    clock_t delta = clock ();
    // MÉRÜNK IDŐT (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);
    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;
    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiindulási c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;
                ++iteracio;
            }
            kepadat[j][k] = iteracio;
        }
    }
}
```

```

    }
    times (&tmsbuf2);
    std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
               + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;
    delta = clock () - delta;
    std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}
int
main (int argc, char *argv[])
{
    if (argc != 2)
    {
        std::cout << "Hasznalat: ./mandelpng fajlnev";
        return -1;
    }
    int kepadat[MERET][MERET];
    mandel(kepadat);
    png::image < png::rgb_pixel > kep (MERET, MERET);
    for (int j = 0; j < MERET; ++j)
    {
        //sor = j;
        for (int k = 0; k < MERET; ++k)
        {
            kep.set_pixel (k, j,
                           png::rgb_pixel (255 -
                                              (255 * kepadat[j][k]) / ITER_HAT <-
                                              '
                                              255 -
                                              (255 * kepadat[j][k]) / ITER_HAT <-
                                              '
                                              255 -
                                              (255 * kepadat[j][k]) / ITER_HAT <-
                                              ));
        }
    }
    kep.write (argv[1]);
    std::cout << argv[1] << " mentve" << std::endl;
}

```

A kód a mandelbrot halmazt vizualizálja. a Void mandle halmazban definiáljuk a halmazt. (Matematikailag a halmaz egyik z $n+1$ -edik elemét $(x_n^2)+c$ -képpen írhatjuk fel.) Komplex számokkal végezzük el a fenti műveletet. Emiatt van egy koordináta rendszerünk, melyet griddel jelenítünk meg, és csomópontjait komplex számok alkotják. A halmaz elemeihez, a fekete színt rendeljük a kiírandó pontokhoz, a megadott határokon belül. (Az iterációs határ fontos szerepet tölt be, hiszen ha a számok nem esnek bele ebbe az iterációs határba, akkor nem elemei a halmaznak, azaz a végtelenbe is 'ugorhatnak/ugranak'.) Tutor: Rémiás Dávid

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

Megoldás videó:

Megoldás forrása: // Verzio: 3.1.2.cpp

```
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040 ↵
// -0.01947381057309366392260585598705802112818 ↵
// -0.0194738105725413418456426484226540196687 ↵
// 0.7985057569338268601555341774655971676111 ↵
// 0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020 ↵
// 0.4127655418209589255340574709407519549131 ↵
// 0.4127655418245818053080142817634623497725 ↵
// 0.2135387051768746491386963270997512154281 ↵
// 0.2135387051804975289126531379224616102874
// Nyomtatás:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1 --left-footer=" ↵
// BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↵
// color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfaí, batfai.norbert@inf.unideb.hu
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
#include <iostream>
#include "png++/png.hpp"
#include <complex>
int
main ( int argc, char *argv[] )
{
    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double a = -1.9;
```

```
double b = 0.7;
double c = -1.3;
double d = 1.3;
if ( argc == 9 )
{
    szelesseg = atoi ( argv[2] );
    magassag =  atoi ( argv[3] );
    iteraciosHatar =  atoi ( argv[4] );
    a = atof ( argv[5] );
    b = atof ( argv[6] );
    c = atof ( argv[7] );
    d = atof ( argv[8] );
}
else
{
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↵
        " << std::endl;
    return -1;
}
png::image < png::rgb_pixel > kep ( szelesseg, magassag );
double dx = ( b - a ) / szelesseg;
double dy = ( d - c ) / magassag;
double reC, imC, reZ, imZ;
int iteracio = 0;
std::cout << "Szamitas\n";
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon
    for ( int k = 0; k < szelesseg; ++k )
    {
        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam
        reC = a + k * dx;
        imC = d - j * dy;
        std::complex<double> c ( reC, imC );
        std::complex<double> z_n ( 0, 0 );
        iteracio = 0;
        while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
        {
            z_n = z_n * z_n + c;
            ++iteracio;
        }
        kep.set_pixel ( k, j,
            png::rgb_pixel ( iteracio%255, (iteracio*iteracio ↵
                )%255, 0 ) );
    }
    int szazalek = ( double ) j / ( double ) magassag * 100.0;
    std::cout << "\r" << szazalek << "%" << std::flush;
}
}
```

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

A Mandelbrot-halmazt Benoit Mandelbrot fedezte fel, és Adrien Douady és John Hamal Hubbard nevezte el róla 1982-ben. A matematikában a Mandelbrot-halmaz azon c komplex számokból áll, melyekre az alábbi x_n rekurzív sorozat: $x_1 := c$, $x_{n+1} := (x_n)^2 + c$. Segítségül hívjuk a headerben az `#include complex` könyvtárat amelyben már megtalálhatóak a korábban leírt függvények. Az elején deklaráljuk a megfelelő egységeinket `int/double` típusú változóinkat felhasználva. Az `argc` utal arra, hogy hány. Szóval hívom meg a programunkat, illetve az `argv` amely a parancssori argumentumok kezeléséhez szükséges. `Argc` amennyiben 9 szóval hívjuk meg a programot az esetben a szélesség lesz a harmadik a magasság (ez a kettő lesz a komplex sík vizsgált tartományára feszített háló) lesz a negyedik az iterációs határ, tehát hogy maximum hány lépésig tudom nagyítani. Az `atoi` egy string típusú változót átkonvertál `int` típusúvá. Az `atof` pedig `double` típusúvá konvertál. A `dx/dy` esetében megadjuk, hogy az $[a,b] \times [c,d]$ tartományon milyen sűrű a megadott szélesség/magasság háló. A `for` ciklusban a j -vel végigmegyünk a sorokon a k -val pedig az oszlopokon. A `while` ciklusban, ha z_n kisebb, mint 4, a feltétel nem teljesül.

5.3. Biomorfok

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

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

A biomorf algoritmus a gerinctelen szervezetekre hasonlító változatos és bonyolult formák létrehozásában használható. A csokor második feladványához hasonlóan indul programunk. Deklaráljuk a megfelelő egységeinket, melyekkel felkosztjuk a programunkat, hogy tudjon mivel dolgozni. Kiegészülve három számmal: "reC" "imC" és "R". A "reC" a C komplex számunk valós része az "imC" pedig a C komplex számunknak az imaginárius része. Az R pedig a valós szám. Ha azt át lépjük akkor a végtelenbe fogunk elszállni. Ezekkel természetesen a parancssori argumentumunkat is ki kell egészítenünk. Ami még újdonság a kódunkban az az `std::pow` ami a complex könyvtárban található. Elvégez helyettünk műveleteket, jelen esetben z_n -t a 3. ra emeljük vele és nem kell leírunk háromszor egymás után szorzás formájában. Az `std::real/std::imag` a valós illetve imaginárius rész. Feltételünkön belül találhatóak meg és rájuk érvényesek a korábban kitett végtelenbe kirepülő kijelentésem. A `break`-re azért van szükségünk, mert ha már minden rácpontot megvizsgáltunk, akkor ki kell léptetni az iterációból. A `kep.set_pixel (x, y, png::rgb_pixel ((iteracio*20)%255, (iteracio*40)%255, (iteracio*60)%255))`; kódban színezhajjuk illetve a formáján is mahinálhajjuk kedvünkre, a lefutás után megkapott ábrát.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

```
// mandelpngc_60x60_100.cu  
// Copyright (C) 2019  
// Norbert Bاتفai, batfai.norbert@inf.unideb.hu  
//
```

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program. If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// Mandelbrot png
// Programozó Páternosztér/PARP
// https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ←
// _01_parhuzamos_prog_linux
//
// https://youtu.be/gvaqijHlRUs
//
#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>
#include <sys/times.h>
#include <iostream>
#define MERET 600
#define ITER_HAT 32000
__device__ int
mandel (int k, int j)
{
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:
    // most éppen a j. sor k. oszlopában 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 elértük, akkor úgy vesszük,
// hogy a kiindulási 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 << "Hasznalat: ./mandelpngc fajlnev";
        return -1;
    }
    int kepadat[MERET][MERET];
    cudamandel (kepadat);
    png::image < png::rgb_pixel > kep (MERET, MERET);
    for (int j = 0; j < MERET; ++j)
    {
        //sor = j;
        for (int k = 0; k < MERET; ++k)
        {
            kep.set_pixel (k, j,
                png::rgb_pixel (255 -
                    (255 * kepadat[j][k]) / ITER_HAT,
                    255 -
                    (255 * kepadat[j][k]) / ITER_HAT,
                    255 -
                    (255 * kepadat[j][k]) / ITER_HAT));
        }
    }
    kep.write (argv[1]);
    std::cout << argv[1] << " mentve" << std::endl;
    times (&tmsbuf2);
    std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
        + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;
    delta = clock () - delta;
    std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
}
```

A CUDA-s megoldás gyorsabb mint a fent prezentált programunk. Az első program a processzorunkat veszi igénybe. míg ez a videókártyánkat ezért a CUDA (GPU) esetében 60x60x100 szál dolgozik. Ezáltal párhuzamosan számolunk ki pontokat. A gyorsaságunk a 'delta clock()' funkció miatt még látványosabb.(időt mérünk vele) Tutor: Rémiás Dávid

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

```
#include "SFML/Graphics.hpp"
const int width = 1280;
const int height = 720;
struct complex_number
{
    long double real;
    long double imaginary;
};
void generate_mandelbrot_set(sf::VertexArray& vertexarray, int pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            long double x = ((long double)j - pixel_shift_x) / zoom;
            long double y = ((long double)i - pixel_shift_y) / zoom;
            complex_number c;
            c.real = x;
            c.imaginary = y;
            complex_number z = c;
            int iterations = 0;
            for (int k = 0; k < precision; k++)
            {
                complex_number z2;
                z2.real = z.real * z.real - z.imaginary * z.imaginary;
                z2.imaginary = 2 * z.real * z.imaginary;
                z2.real += c.real;
                z2.imaginary += c.imaginary;
                z = z2;
                iterations++;
                if (z.real * z.real + z.imaginary * z.imaginary > 4)
                    break;
            }
            if (iterations < precision / 4.0f)
            {
                vertexarray[i*width + j].position = sf::Vector2f(j, i);
                sf::Color color(iterations * 255.0f / (precision / 4.0f), 0, 0);
            }
        }
    }
}
```

```
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision / 2.0f)
    {
        vertexarray[i*width + j].position = sf::Vector2f(j, i);
        sf::Color color(0, iterations * 255.0f / (precision / 2.0f) ←
            , 0);
        vertexarray[i*width + j].color = color;
    }
    else if (iterations < precision)
    {
        vertexarray[i*width + j].position = sf::Vector2f(j, i);
        sf::Color color(0, 0, iterations * 255.0f / precision);
        vertexarray[i*width + j].color = color;
    }
}
}
}
int main()
{
    sf::String title_string = "Mandelbrot Set Plotter";
    sf::RenderWindow window(sf::VideoMode(width, height), title_string); ←
        méretekkel és címmel)
    window.setFramerateLimit(30);
    sf::VertexArray pointmap(sf::Points, width * height);

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

    generate_mandelbrot_set(pointmap, x_shift, y_shift, precision, zoom);

    /**
     *
     *
     *
     *
     *
     * */
    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }
    }
}
```

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
    sf::Vector2i position = sf::Mouse::getPosition(window);
    x_shift -= position.x - x_shift;
    y_shift -= position.y - y_shift;
    zoom *= 2;
    precision += 200;

    #pragma omp parallel for
    for (int i = 0; i < width*height; i++)
    {
        pointmap[i].color = sf::Color::Black;
    }
    generate_mandelbrot_set(pointmap, x_shift, y_shift, precision, ←
        zoom);
}
window.clear();
window.draw(pointmap);
window.display();
}

return 0;
}
```

A program fordításához szükségünk van arra, hogy telepítsünk gépünkre egy két alapvető csomagot. sudo apt-get install libsfml-dev Build: g++ MandelbrotSetPlotter.cpp -lsfml-window -lsfml-system -lsfml-graphics -fopenmp -O2 Használat: g++ -o mbhzoom MandelbrotSetPlotter.cpp -lsfml-window -lsfml-system -lsfml-graphics -fopenmp -O2 Indítás: ./mbhzoom

```
const int width = 1280;
const int height = 720;
```

Első körben beállítjuk az ablak szélességét/magasságát. A void generate_mandelbrot_set után egészen az "int main()-ig a mandelbrot komplex alapján legenerálunk egy mandelbrot halmazt. Megadjuk az ablaknak a címét majd az ablak objektum létrehozza az ablakot a megadott mértékkel és címmel.

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
    sf::Vector2i position = sf::Mouse::getPosition(window);
    x_shift -= position.x - x_shift;
    y_shift -= position.y - y_shift;
    zoom *= 2;
precision += 200;
```

Továbbá ha a bal egérgommbal kattintunk, akkor az egér helyére nagyít az alábbi algoritmus segítségével. Minden nagyítás után újra legenerálja a halmazt.

5.6. Mandelbrot nagyító és utazó Java nyelven

```
/*
 * MandelbrotHalmazNagyító.java
 *
 * DIGIT 2005, Javat tanítók
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt nagyító és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class MandelbrotHalmazNagyító extends MandelbrotHalmaz {
    /** A nagyítandó kijelölt területet bal felső sarka. */
    private int x, y;
    /** A nagyítandó kijelölt terület szélessége és magassága. */
    private int mx, my;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló és nagyítani tudó
     * <code>MandelbrotHalmazNagyító</code> objektumot.
     *
     * @param a a [a,b]x[c,d] tartomány a koordinátája.
     * @param b a [a,b]x[c,d] tartomány b koordinátája.
     * @param c a [a,b]x[c,d] tartomány c koordinátája.
     * @param d a [a,b]x[c,d] tartomány d koordinátája.
     * @param szélesség a halmazt tartalmazó tömb szélessége.
     * @param iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmazNagyító(double a, double b, double c, double d,
        int szélesség, int iterációsHatár) {
        // Az ősz osztály konstruktorának hívása
        super(a, b, c, d, szélesség, iterációsHatár);
        setTitle("A Mandelbrot halmaz nagyításai");
        // Egér kattintó események feldolgozása:
        addMouseListener(new java.awt.event.MouseAdapter() {
            // Egér kattintással jelöljük ki a nagyítandó területet
            // bal felső sarkát:
            public void mousePressed(java.awt.event.MouseEvent m) {
                // A nagyítandó kijelölt területet bal felső sarka:
                x = m.getX();
                y = m.getY();
                mx = 0;
                my = 0;
                repaint();
            }
        })
    }
}
```

```
// Vonszolva kijelölünk egy területet...
// Ha felengedjük, akkor a kijelölt terület
// újraszámítása indul:
public void mouseReleased(java.awt.event.MouseEvent m) {
    double dx = (MandelbrotHalmazNagyító.this.b
        - MandelbrotHalmazNagyító.this.a)
        /MandelbrotHalmazNagyító.this.szélesség;
    double dy = (MandelbrotHalmazNagyító.this.d
        - MandelbrotHalmazNagyító.this.c)
        /MandelbrotHalmazNagyító.this.magasság;
    // Az új Mandelbrot nagyító objektum elkészítése:
    new MandelbrotHalmazNagyító(MandelbrotHalmazNagyító.this.a+ ←
        x*dx,
        MandelbrotHalmazNagyító.this.a+x*dx+mx*dx,
        MandelbrotHalmazNagyító.this.d-y*dy-my*dy,
        MandelbrotHalmazNagyító.this.d-y*dy,
        600,
        MandelbrotHalmazNagyító.this.iterációsHatár);
}
});
// Egér mozgás események feldolgozása:
addMouseListener(new java.awt.event.MouseMotionAdapter() {
    // Vonszolással jelöljük ki a négyzetet:
    public void mouseDragged(java.awt.event.MouseEvent m) {
        // A nagyítandó kijelölt terület szélessége és magassága:
        mx = m.getX() - x;
        my = m.getY() - y;
        repaint();
    }
});
}
/**
 * Pillanatfelvételek készítése.
 */
public void pillanatfelvétel() {
    // Az elmentendő kép elkészítése:
    java.awt.image.BufferedImage mentKép =
        new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKép.getGraphics();
    g.drawImage(kép, 0, 0, this);
    g.setColor(java.awt.Color.BLUE);
    g.drawString("a=" + a, 10, 15);
    g.drawString("b=" + b, 10, 30);
    g.drawString("c=" + c, 10, 45);
    g.drawString("d=" + d, 10, 60);
    g.drawString("n=" + iterációsHatár, 10, 75);
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
}
```

```
    }
    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
    g.dispose();
    // A pillanatfelvétel képfájl nevének képzése:
    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrotHalmazNagyitas_");
    sb.append(++pillanatfelvételSzámláló);
    sb.append("_");
    // A fájl nevébe bele vesszük, hogy melyik tartományban
    // találtuk a halmazt:
    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
    sb.append("_");
    sb.append(d);
    sb.append(".png");
    // png formátumú képet mentünk
    try {
        javax.imageio.ImageIO.write(mentKép, "png",
            new java.io.File(sb.toString()));
    } catch (java.io.IOException e) {
        e.printStackTrace();
    }
}
/**
 * A nagyítandó kijelölt területet jelző négyzet kirajzolása.
 */
public void paint(java.awt.Graphics g) {
    // A Mandelbrot halmaz kirajzolása
    g.drawImage(kép, 0, 0, this);
    // Ha éppen fut a számítás, akkor egy vörös
    // vonallal jelöljük, hogy melyik sorban tart:
    if (számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
    // A jelző négyzet kirajzolása:
    g.setColor(java.awt.Color.GREEN);
    g.drawRect(x, y, mx, my);
}
/**
 * Példányosít egy Mandelbrot halmazt nagyító obektumot.
 */
public static void main(String[] args) {
    // A kiinduló halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35]
    // tartományában keressük egy 600x600-as hálózattal és az
```

```
// aktuális nagyítási pontossággal:  
new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);  
}  
}
```

DRAFT

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 elspájazolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltéve kiszámolt szám.

Megoldás videó:

Megoldás forrása:

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



```
        v2 = 2 * u2 - 1;
        w = v1 * v1 + v2 * v2;
    }
    while (w > 1);
    double r = std::sqrt ((-2 * std::log (w)) / w);
    tarolt = r * v2;
    nincsTarolt = !nincsTarolt;
    return r * v1;
}
else
{
    nincsTarolt = !nincsTarolt;
    return tarolt;
}
}
private:
    bool nincsTarolt;
    double tarolt;
};
int
main (int argc, char **argv)
{
    PolarGen pg;
    for (int i = 0; i < 10; ++i)
        std::cout << pg.kovetkezo () << std::endl;
    return 0;
}
```

A Polargen nevű osztály létrehozásával kezdtük a feladatot. Egy random number generatornak random seedet adunk, ez 100-ig megy. A konstruktorban, pedig megadjuk, hogy még nincs eltárolt szám. A "kovetkezo" keresi, hogy van-e eltárolt számunk, ha nincs generál random két számot. Az egyiket eltárolja, majd a logikai változót hamisra állítja és visszatér a másik számmal. Igaz esetén azt jelenti, hogy az eltárolt változóban eltároltuk a vissza adandó számot.

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
typedef struct node{
    char c;
    struct node* left;
    struct node* right;
```

```
} Node;
Node* fa;
Node gyoker;
#define null NULL
Node* create_empty()
{
    Node* tmp = &gyoker;
    tmp->c= '/';
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
Node* create_node(char val)
{
    Node* tmp = (Node*)malloc(sizeof(Node));
    tmp->c=val;
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
void insert_tree(char val)
{
    {
        if(val=='0')
        {
            if(fa->left == null)
            {
                fa->left = create_node(val);
                fa = &gyoker;
                //printf("Inserted into left.");
            }
            else
            {
                fa = fa->left;
            }
        }
        else
        {
            if(fa->right == null)
            {
                fa->right = create_node(val);
                fa = &gyoker;
                //printf("Inserted into left.");
            }
            else
            {
                fa = fa->right;
            }
        }
    }
}
void inorder(Node* elem,int depth)
```

```
{
if (elem==null)
{
return;
}
inorder(elem->left, depth+1);
if (depth)
{
char *spaces;
spaces = (char*) malloc(sizeof(char)*depth*2+1);
for(int i=0; i<depth; i+=2)
{
spaces[i]='-';
spaces[i+1]='-';
}
spaces[depth]='\0';
printf("%s%c\n", spaces, elem->c);
}
else
{
printf("%c\n", elem->c);
}
inorder(elem->right, depth+1);
}

void preorder(Node* elem, int depth)
{
if (elem==null)
{
return;
}
if (depth)
{
char *spaces;
spaces = (char*) malloc(sizeof(char)*depth*2+1);
for(int i=0; i<depth; i+=2)
{
spaces[i]='-';
spaces[i+1]='-';
}
spaces[depth*2]='\0';
printf("%s%c\n", spaces, elem->c);
}
else
{
printf("%c\n", elem->c);
}
preorder(elem->left, depth+1);
preorder(elem->right, depth+1);
}

void postorder(Node* elem, int depth)
```

```
{
if(elem==null)
{
return;
}
postorder(elem->left,depth+1);
postorder(elem->right,depth+1);
if(depth)
{
char *spaces;
spaces = (char*) malloc(sizeof(char)*depth*2+1);
for(int i=0;i<depth;i+=2)
{
spaces[i]='-';
spaces[i+1]='-';
}
spaces[depth*2]='\0';
printf("%s%c\n",spaces,elem->c);
free(spaces);
}
else
{
printf("%c\n",elem->c);
}
}
void destroy_tree(Node* elem)
{
if(elem==null)
{
return;
}
destroy_tree(elem->left);
destroy_tree(elem->right);
if(elem->c == gyoker.c)
{
}
else
{
free(elem);
}
}
void usage()
{
printf("Használat: ./binfa KAPCSOLÓ\n");
printf("Az KAPCSOLÓ lehet:\n");
printf("--preorder\tA bináris fa preorder bejárása\n");
printf("--inorder\tA bináris fa inorder bejárása\n");
printf("--postorder\tA bináris fa postorder bejárása\n");
}
```

```
int main(int argc, char** argv)
{
    srand(time(NULL));
    fa = create_empty();
    //gyoker = *fa;
    for(int i=0; i<10000; i++)
    {
        int x=rand()%2;
        if(x)
        {
            insert_tree('1');
        }
        else
        {
            insert_tree('0');
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1], "--preorder")==0)
        {
            preorder(&gyoker, 0);
        }
        else if(strcmp(argv[1], "--inorder")==0)
        {
            inorder(&gyoker, 0);
        }
        else if(strcmp(argv[1], "--postorder")==0)
        {
            postorder(&gyoker, 0);
        }
        else
        {
            usage();
        }
    }
    else
    {
        usage();
    }
    destroy_tree(&gyoker);
    return 0;
}
```

Az Lempel-Ziv-Welch algoritmus egy veszteségmentes tömörítési eljárás, ilyen lehet a gif formátuma és ezeket tömörítők is használhatják. A kódoló jelen esetben a szótárbeli paramétert küldi át, ez a kódolás során dinamikusan bővül. A `create_empty` függvény a program használata előtt a működéshez szükséges kezdőértékeket be állítja a bináris fát illetően egy kitüntetett gyökérelemmel, ami a "/" karakterrel van megjelölve a bejárások esetében. A `create_node` amely függvényünk generál egy csomópontot az argumentumként kapott karakterrel (`malloc(sizeof(Node))`), a gyermekekre mutató mutatókat NULL-

ra állítja be. Az `insert_tree` függvény valósítja meg az LZW bináris fa felépítését. Az `inorder` eljárás `inorder` módon azaz önmagát ismételve/rekurzívan bejárja a bináris fát. Itt először a fa baloldalát, majd a gyökérelemet, legutoljára pedig a fa jobb oldalát járjuk be. A `destroy_tree` az az eljárás ami rekurzívan `postorder` módon bejárja a fát és minden ismétlés/rekurzió végén felszabadítja a részfának a gyökérelemét. A `main` függvényben feltöltöm a fát 10000 elemmel, majd az `inorder` módon bejárom a bináris fánkat, majd felszabadítom a fa pointereit.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
typedef struct node{
    char c;
    struct node* left;
    struct node* right;
} Node;
Node* fa;
Node gyoker;
#define null NULL
Node* create_empty()
{
    Node* tmp = &gyoker;
    tmp->c= '/';
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
Node* create_node(char val)
{
    Node* tmp = (Node*)malloc(sizeof(Node));
    tmp->c=val;
    tmp->left = null;
    tmp->right = null;
    return tmp;
}
void insert_tree(char val)
{
    if(val=='0')
    {
        if(fa->left == null)
        {
            fa->left = create_node(val);
        }
    }
}
```

```
        fa = &gyoker;
        //printf("Inserted into left.");
    }
    else
    {
        fa = fa->left;
    }
}
else
{
    if(fa->right == null)
    {
        fa->right = create_node(val);
        fa = &gyoker;
        //printf("Inserted into left.");
    }
    else
    {
        fa = fa->right;
    }
}
}

void inorder(Node* elem,int depth)
{
    if(elem==null)
    {
        return;
    }
    inorder(elem->left,depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth]='\0';
        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
    inorder(elem->right,depth+1);
}

void preorder(Node* elem,int depth)
{
    if(elem==null)
```

```
{
    return;
}
if(depth)
{
    char *spaces;
    spaces = (char*) malloc(sizeof(char)*depth*2+1);
    for(int i=0;i<depth;i+=2)
    {
        spaces[i]='-';
        spaces[i+1]='-';
    }
    spaces[depth*2]='\0';
    printf("%s%c\n", spaces, elem->c);
}
else
{
    printf("%c\n", elem->c);
}
preorder(elem->left, depth+1);
preorder(elem->right, depth+1);
}

void postorder(Node* elem, int depth)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left, depth+1);
    postorder(elem->right, depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';
        printf("%s%c\n", spaces, elem->c);
        free(spaces);
    }
    else
    {
        printf("%c\n", elem->c);
    }
}

void destroy_tree(Node* elem)
{

```



```
if(elem==null)
{
    return;
}
destroy_tree(elem->left);
destroy_tree(elem->right);
if(elem->c == gyoker.c)
{
}
else
{
    free(elem);
}
}
void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}
int main(int argc, char** argv)
{
    srand(time(NULL));
    fa = create_empty();
    //gyoker = *fa;
    for(int i=0;i<10000;i++)
    {
        int x=rand()%2;
        if(x)
        {
            insert_tree('1');
        }
        else
        {
            insert_tree('0');
        }
    }
    if(argc == 2)
    {
        if(strcmp(argv[1], "--preorder")==0)
        {
            preorder(&gyoker, 0);
        }
        else if(strcmp(argv[1], "--inorder")==0)
        {
            inorder(&gyoker, 0);
        }
        else if(strcmp(argv[1], "--postorder")==0)
```

```
{
    postorder(&gyoker, 0);
}
else
{
    usage();
}
}
else
{
    usage();
}
destroy_tree(&gyoker);
return 0;
}
"
```

A preorder eljárás annyiban más mint az inorder, hogy ebben először feldolgozzuk a részfa gyökerét, majd bejárjuk a részfa bal oldalát, ha ez megvan akkor a jobb oldalát is. A postorder eljárás pedig annyiban különbözik mint az alap eset "inorder", hogy ebben először a részfának a bal oldalát járjuk be, után a a jobb oldalát, majd feldolgozzuk a részfának a gyökerét.

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

```
"#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string.h>
#define null NULL
class Binfa
{
private:
    class Node
    {
    public:
        Node(char c='/')
        {
            this->c=c;
            this->left = null;
            this->right = null;
        }
        char c;
```

```
    Node* left;
    Node* right;
};
Node* fa;

public:
    Binfo(): fa(&gyoker)
    {
    }

    void operator<<(char c)
    {
        if(c=='0')
        {
            if(fa->left == null)
            {
                fa->left = new Node('0');
                fa = &gyoker;
            }
            else
            {
                fa = fa->left;
            }
        }
        else
        {
            if(fa->right == null)
            {
                fa->right = new Node('1');
                fa = &gyoker;
            }
            else
            {
                fa = fa->right;
            }
        }
    }

    void preorder(Node* elem, int depth=0)
    {
        if(elem==null)
        {
            return;
        }
        if(depth)
        {
            char *spaces;
            spaces = (char*) malloc(sizeof(char)*depth*2+1);
            for(int i=0; i<depth; i+=2)
            {
```

```
        spaces[i]='-';
        spaces[i+1]='-';
    }
    spaces[depth*2]='\0';
    printf("%s%c\n", spaces, elem->c);
}
else
{
    printf("%c\n", elem->c);
}
preorder(elem->left, depth+1);
preorder(elem->right, depth+1);
}
void inorder(Node* elem, int depth=0)
{
    if(elem==null)
    {
        return;
    }
    inorder(elem->left, depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0; i<depth; i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';
        printf("%s%c\n", spaces, elem->c);
    }
    else
    {
        printf("%c\n", elem->c);
    }
    inorder(elem->right, depth+1);
}
void postorder(Node* elem, int depth=0)
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left, depth+1);
    postorder(elem->right, depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
```

```
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';
        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
}
void destroy_tree(Node* elem)
{
    if(elem==null)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if(elem->c!='/') delete elem;
}
Node gyoker;
};
void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}
int main(int argc, char** argv)
{
    srand(time(0));
    Binfo bfa;
    for(int i=0;i<100;i++)
    {
        int x=rand()%2;
        if(x)
        {
            bfa<<'1';
        }
        else
        {
            bfa<<'0';
        }
    }
    if(argc == 2)
```

```
{
    if (strcmp(argv[1], "--preorder")==0)
    {
        bfa.preorder(&bfa.gyoker);
    }
    else if (strcmp(argv[1], "--inorder")==0)
    {
        bfa.inorder(&bfa.gyoker);
    }
    else if (strcmp(argv[1], "--postorder")==0)
    {
        bfa.postorder(&bfa.gyoker);
    }
    else
    {
        usage();
    }
}
else
{
    usage();
}
bfa.destroy_tree(&bfa.gyoker);
return 0;
}
">Binfa c++
```

A bináris fát kezelő függvényeket és eljárásokat, a Binfa osztályon belülre kell rendeznünk. Létrehozzuk a "public" és a "privát" szekciót illetve a privát részévé tettem a Node struktúrát. A binfa objektum létrehozása után amit a "main"-ben tettem meg, ezen keresztül lehet elérni a binfa adatait. A binfa osztályon belül túlterhelem a "balra bitshift" operátort ami mostmár a bináris fa építését látja el. Ezt ugyanazon a munkameneten elven keresztül teszi meg mint az előző megoldásoknál tette az insert_tree eljárás.

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:

```
"#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string.h>
#define null NULL
class Binfa
{
private:
```

```
class Node
{
public:
    Node(char c='/')
    {
        this->c=c;
        this->left = null;
        this->right = null;
    }
    char c;
    Node* left;
    Node* right;
};
Node* fa;

public:
    Binfo()
    {
        gyoker=fa=new Node();
    }

    void operator<<(char c)
    {
        if(c=='0')
        {
            if(fa->left == null)
            {
                fa->left = new Node('0');
                fa = gyoker;
            }
            else
            {
                fa = fa->left;
            }
        }
        else
        {
            if(fa->right == null)
            {
                fa->right = new Node('1');
                fa = gyoker;
            }
            else
            {
                fa = fa->right;
            }
        }
    }

    void preorder(Node* elem,int depth=0)
```

```
{
    if(elem==null)
    {
        return;
    }
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';
        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
    preorder(elem->left,depth+1);
    preorder(elem->right,depth+1);
}

void inorder(Node* elem,int depth=0)
{
    if(elem==null)
    {
        return;
    }
    inorder(elem->left,depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';
        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
    inorder(elem->right,depth+1);
}

void postorder(Node* elem,int depth=0)
```



```
{
    if(elem==null)
    {
        return;
    }
    postorder(elem->left,depth+1);
    postorder(elem->right,depth+1);
    if(depth)
    {
        char *spaces;
        spaces = (char*) malloc(sizeof(char)*depth*2+1);
        for(int i=0;i<depth;i+=2)
        {
            spaces[i]='-';
            spaces[i+1]='-';
        }
        spaces[depth*2]='\0';
        printf("%s%c\n",spaces,elem->c);
    }
    else
    {
        printf("%c\n",elem->c);
    }
}
void destroy_tree(Node* elem)
{
    if(elem==null)
    {
        return;
    }
    destroy_tree(elem->left);
    destroy_tree(elem->right);
    if(elem->c!='/') delete elem;
}
Node* gyoker;
};
void usage()
{
    printf("Használat: ./binfa KAPCSOLÓ\n");
    printf("Az KAPCSOLÓ lehet:\n");
    printf("--preorder\tA bináris fa preorder bejárása\n");
    printf("--inorder\tA bináris fa inorder bejárása\n");
    printf("--postorder\tA bináris fa postorder bejárása\n");
}
int main(int argc, char** argv)
{
    srand(time(0));
    Binfo bfa;
    for(int i=0;i<100;i++)
    {
```

```
int x=rand()%2;
if(x)
{
    bfa<<'1';
}
else
{
    bfa<<'0';
}
}
if(argc == 2)
{
    if(strcmp(argv[1],"--preorder")==0)
    {
        bfa.preorder(bfa.gyoker);
    }
    else if(strcmp(argv[1],"--inorder")==0)
    {
        bfa.inorder(bfa.gyoker);
    }
    else if(strcmp(argv[1],"--postorder")==0)
    {
        bfa.postorder(bfa.gyoker);
    }
    else
    {
        usage();
    }
}
else
{
    usage();
}
bfa.destroy_tree(bfa.gyoker);
return 0;
}
"
```

Módosítottunk az előző feladaton annyit, hogy a bejárás miatt létrehoztunk egy gyökérobjektumot. Ezelőtt, ahol a program azt kérte hogy a gyökér referenciáját adjuk meg, most a gyökeret magát kell megadnunk.

Tutoriált: Rémiás Dávid

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

DRAFT

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó: // sejtablak.cpp

```
#include <SFML/System.hpp>
#include <SFML/Graphics.hpp>
#include <vector>
#include <iostream>
using namespace sf;
using std::vector;
using std::cout;
```

```
using std::endl;
class Grid
{
public:
    Grid(unsigned int x = 1000, unsigned int y = 1000, unsigned int diffs = ←
        50) : w(x),h(y),diff(diffs)
    {

    }
    void draw(RenderWindow & window)
    {
        for(int i=0;i<w;i+=diff)
        {
            Vertex line[] =
            {
                sf::Vertex(sf::Vector2f(i,0)),
                sf::Vertex(sf::Vector2f(i, h))
            };
            line[0].color = Color(0,0,0);
            line[1].color = Color(0,0,0);
            window.draw(line, 2, sf::Lines);
        }
        for(int i=0;i<h;i+=diff)
        {
            Vertex line[] =
            {
                sf::Vertex(sf::Vector2f(0,i)),
                sf::Vertex(sf::Vector2f(w,i))
            };
            line[0].color = Color(0,0,0);
            line[1].color = Color(0,0,0);
            window.draw(line, 2, sf::Lines);
        }
    }
    unsigned int w;
    unsigned int h;
    unsigned int diff;
};

class Square
{
public:
    Square()
    {
    }
    Square(int x_pos, int y_pos, float w, bool alive = false)
    {
        square = new RectangleShape(Vector2f(w,w));
        square->setPosition(Vector2f(x_pos,y_pos));
        aliveState = alive;
    }
}
```

```
/*Square (const Square& other )
{
    if(this != &other)
    {
        delete this->square;
        this->square = other.square;
    }
}
Square& operator=(const Square& other)
{
    if(this != &other)
    {
        delete this->square;
        this->square = other.square;
    }
    return *this;
}*/
~Square()
{
    delete square;
}
void update()
{
    if(aliveState)
    {
        square->setFillColor(Color::Black);
    }
    else
    {
        square->setFillColor(Color::White);
    }
}
void setFill(Color c = Color::White)
{
    square->setFillColor(c);
}
void draw(RenderWindow &window)
{
    window.draw(*square);
}
RectangleShape* square;
bool aliveState;
private:

};
vector<vector<Square*>> update(vector<vector<Square*>> v)
{
    vector<vector<Square*>> tmp ;//= v;
    for(int i=0;i<v.size();i++)
```

```

{
    tmp.push_back(vector<Square*>());
    for(int j=0;j<v[0].size();j++)
    {
        tmp[i].push_back(new Square(v[i][j]->square->getPosition().x,v[ ←
            i][j]->square->getPosition().y,v[i][j]->square->getSize().x, ←
            v[i][j]->aliveState));
    }
}
for(int i=0;i<v.size();i++)
{
    for(int j=0;j<v[0].size();j++)
    {
        int live_neighbours = 0;
        live_neighbours += v[(i-1)%v.size()][(j-1)%v[0].size()]-> ←
            aliveState;
        live_neighbours += v[(i-1)%v.size()][(j)%v[0].size()]-> ←
            aliveState;
        live_neighbours += v[(i-1)%v.size()][(j+1)%v[0].size()]-> ←
            aliveState;
        live_neighbours += v[(i)%v.size()][(j-1)%v[0].size()]-> ←
            aliveState;
        live_neighbours += v[(i)%v.size()][(j+1)%v[0].size()]-> ←
            aliveState;
        live_neighbours += v[(i+1)%v.size()][(j-1)%v[0].size()]-> ←
            aliveState;
        live_neighbours += v[(i+1)%v.size()][(j)%v[0].size()]-> ←
            aliveState;
        live_neighbours += v[(i+1)%v.size()][(j+1)%v[0].size()]-> ←
            aliveState;
        //cout <<" X:"<<i << " y:"<< j << " Live neighbours:"<< ←
            live_neighbours<<endl;
        if(v[i][j]->aliveState)
        {
            if(live_neighbours < 2)
            {
                tmp[i][j]->aliveState = false;
            }
            else if(live_neighbours > 3)
            {
                tmp[i][j]->aliveState = false;
            }
        }
        else
        {
            if(live_neighbours == 3)
            {
                tmp[i][j]->aliveState = true;
            }
        }
    }
}

```

```
    }
}
return tmp;
}
void killall(vector<vector<Square*>> &v)
{
    for(int i=0;i<v.size();i++)
    {
        for(int j=0;j<v[0].size();j++)
        {
            v[i][j]->aliveState=false;
        }
    }
}
int main()
{
    RenderWindow window(VideoMode(1000,1000),"Game of Life");
    window.setFramerateLimit(10);
    window.setActive();

    Vector2u size = window.getSize();
    Grid g(size.x,size.y,1000/40);

    int h = g.h/g.diff+1;
    int w = g.w/g.diff+1;
    //Square squares[h][w];
    std::vector<std::vector<Square*>> squares;
    bool edit_mode = true;
    for(int i=0;i<h;i++)
    {
        squares.push_back(vector<Square*>());
        for(int j=0;j<w;j++)
        {
            squares[i].push_back(new Square(i*g.diff+1,j*g.diff+2,g.diff-3) ←
            );
        }
    }
    //squares[4][5]->aliveState=true;
    while (window.isOpen())
    {
        window.clear(sf::Color::White);
        // check all the window's events that were triggered since the last ←
        iteration of the loop
        sf::Event event;
        while (window.pollEvent(event))
        {
            // "close requested" event: we close the window
            if (event.type == sf::Event::Closed)
            {
```



```
        window.close();
    }
    else if(event.type == Event::MouseButtonPressed)
    {
        if(edit_mode && event.mouseButton.button == Mouse::Button::Left)
        {
            /*cout<<event.mouseButton.x<<" "<<event.mouseButton.y<<
            endl;
            cout<<event.mouseButton.x/g.diff<<" "<<event.
            mouseButton.y/g.diff<<endl;*/
            squares[event.mouseButton.x/g.diff][event.mouseButton.y
            /g.diff]->aliveState= !squares[event.mouseButton.x/g
            .diff][event.mouseButton.y/g.diff]->aliveState;
            cout<<"Changed state on entity at X:"<<event.
            mouseButton.x/g.diff<<" Y:"<<event.mouseButton.y/g
            .diff<<" to "<<(squares[event.mouseButton.x/g.
            diff][event.mouseButton.y/g.diff]->aliveState?"
            Alive" : "Dead")<<endl;
        }
    }
    else if(event.type == Event::KeyPressed)
    {
        if(event.key.code == Keyboard::Q)
        {
            cout<<"Close request recieved. Application will exit
            ."<<endl;
            window.close();
        }
        if(edit_mode && event.key.code == Keyboard::C)
        {
            cout<<"Killed all entities."<<endl;
            killall(squares);
        }
        if(event.key.code == Keyboard::E)
        {
            edit_mode = !edit_mode;
            if(edit_mode)
            {
                cout<<"Changed to edit mode."<<endl;
            }
            else
            {
                cout<<"Changed to simulation mode."<<endl;
            }
        }
    }
}
```

```

    }
    /*s.draw(window);
    s.square->setPosition(Vector2f(s.square->getPosition().x+1,s.square->
        ->getPosition().y));*/
    g.draw(window);
    for(int i=0;i<h;i++)
    {
        for( int j=0; j<w;j++)
        {
            squares[i][j]->draw(window);
        }
    }
    window.display();
    if(!edit_mode) squares = update(squares);
    for(int i=0;i<h;i++)
    {
        for( int j=0; j<w;j++)
        {
            squares[i][j]->update();
        }
    }
}

return EXIT_SUCCESS;

```

A játék szabályokra épül: Első : A sejt túléli a kört, ha két vagy három szomszédja van. Második : A sejt elpusztul , ha kettőnél kevesebb , vagy háromnál több szomszédja van. Harmadik : Az új sejt megszületik minden olyan cellában , melynek környezetében pontosan három sejt található. A feladat megoldásához a programban kettő osztályt és három függvényt használunk. Az első osztályunk a "Grid" elnevezést kapta. Ettől lesz az ablak rácsos megjelenésű. Amint ez az ablak elkészül, a `void draw` függvényünk segítségével jelenítjük meg az ablakot. A második osztályunk a "Square" ami azért van , hogy ha rákattintunk a négyzetre magán a rácshálón belül, akkor azt a rácsot kitöltse fekete színnel. Ezeket a négyzeteket a későbbiekben nekünk folyton frissítenünk kell melyhez a "Square"-en belül a `update` fog segíteni. A harmadik nagyobb részünk a "vector" amely vektorral mozgatjuk a kijelölt négyzeteket a megfelelő játékszabályokat alkalmazva.

```

for(int i=0;i<v.size();i++)
{
    for(int j=0;j<v[0].size();j++)
    {
        int live_neighbours = 0;
        live_neighbours += v[(i-1)%v.size()][(j-1)%v[0].size()]->
            aliveState;
        live_neighbours += v[(i-1)%v.size()][(j)%v[0].size()]->
            aliveState;
        live_neighbours += v[(i-1)%v.size()][(j+1)%v[0].size()]->
            aliveState;
        live_neighbours += v[(i)%v.size()][(j-1)%v[0].size()]->
            aliveState;
        live_neighbours += v[(i)%v.size()][(j+1)%v[0].size()]->

```

```
        aliveState;
live_neighbours += v[(i+1)%v.size()][(j-1)%v[0].size()-> ←
        aliveState;
live_neighbours += v[(i+1)%v.size()][(j)%v[0].size()-> ←
        aliveState;
live_neighbours += v[(i+1)%v.size()][(j+1)%v[0].size()-> ←
        aliveState;
//cout <<" X:"<<i << " y:"<< j << " Live neighbours:"<< ←
live_neighbours<<endl;
if(v[i][j]->aliveState)
{
    if(live_neighbours < 2)
    {
        tmp[i][j]->aliveState = false;
    }
    else if(live_neighbours > 3)
    {
        tmp[i][j]->aliveState = false;
    }
}
else
{
    if(live_neighbours == 3)
    {
        tmp[i][j]->aliveState = true;
    }
}
}
return tmp;
}
```

A negyedik része a programunknak, a killall, ami segít a játékosnak megölni az összes sejtet, ha az nem szeretne kilépni a programból, ha új játékot akar kezdeni. Az ötödik és utolsó részünk a main ahol az ablakunk paramétereit megadjuk. Jelenleg 1000x1000 és a "Game of Life" feliratot kapja. A while cikluson belül letöröljük az ablakot, teljesen tiszta, új felületet kapva. A programunk végén meghívjuk a "draw" illetve "update" függvényünket, amelyekkel kirajzoljuk a szerkesztő módban beállított új játékunkat és frissítjük a programot.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

```
/**
 * @brief Benchmarking Cognitive Abilities of the Brain with Computer Games
 *
```

```
* @file BrainBThread.cpp
* @author Norbert Bاتفai <nbatfai@gmail.com>
* @version 6.0.1
*
* @section LICENSE
*
* Copyright (C) 2017, 2018 Norbert Bاتفai, nbatfai@gmail.com
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program. If not, see <http://www.gnu.org/licenses/>.
*
* @section DESCRIPTION
*
*/
#include "BrainBThread.h"
BrainBThread::BrainBThread ( int w, int h )
{
    dispShift = heroRectSize+heroRectSize/2;
    this->w = w - 3 * heroRectSize;
    this->h = h - 3 * heroRectSize;
    std::srand ( std::time ( 0 ) );
    Hero me ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
        100,
        this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - ←
        100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), 9 );
    Hero other1 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100,
        this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        5, "Norbi Entropy" );
    Hero other2 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100,
        this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        3, "Greta Entropy" );
    Hero other4 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100,
        this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ←
        ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ), ←
        5, "Nandi Entropy" );
```

```
        Hero other5 ( this->w / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100,
                    this->h / 2 + 200.0 * std::rand() / ( RAND_MAX + 1.0 ) - 100, 255.0 * std::rand() / ( RAND_MAX + 1.0 ),
                    7, "Matyi Entropy" );
        heroes.push_back ( me );
        heroes.push_back ( other1 );
        heroes.push_back ( other2 );
        heroes.push_back ( other4 );
        heroes.push_back ( other5 );
    }
    BrainBThread::~BrainBThread()
    {
    }
    void BrainBThread::run()
    {
        while ( time < endTime ) {
            QThread::msleep ( delay );
            if ( !paused ) {
                ++time;
                devel();
            }
            draw();
        }
        emit endAndStats ( endTime );
    }
    void BrainBThread::pause()
    {
        paused = !paused;
        if ( paused ) {
            ++nofPaused;
        }
    }
    void BrainBThread::set_paused ( bool p )
    {
        if ( !paused && p ) {
            ++nofPaused;
        }
        paused = p;
    }
}
```

A program jelentős részét a header file képezi. A headerben a hősök tulajdonságait találjuk, tartózkodási hely x és y koordinátákkal, valamint egy "agility" érték, amely a hősünk mozgásáért felelős (hány egységet léphet egyszerre). //a header pontos megértése szükséges A cpp fájlban 4 játékost láthatunk, valamint egy pause funkciót. Tutor: Rémiás Dávid

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/exar
https://progater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Tanulságok, tapasztalatok, magyarázat...



FIGYELEM

A futtáshoz telepíteni kell a Tensorflow megfelelő csomagjait és a Python3 fejlesztői csomagokat is. Kövesd az általad használt Linux disztribúció telepítési útmutatóját hozzá!

Jelenleg(2019.04.21.) így telepíthetjük egy Ubuntu disztrón:

```
sudo apt install python3-dev python3-pip
sudo pip3 install -U virtualenv # system-wide install
virtualenv --system-site-packages -p python3 ./venv
source ./venv/bin/activate # sh, bash, ksh, or zsh
pip install --upgrade pip
pip install --upgrade tensorflow
#ellenőrizzük, hogy helyesen települt-e
python -c "import tensorflow as tf; tf.enable_eager_execution(); print(tf. ←
    reduce_sum(tf.random_normal([1000, 1000])))"
```

Az MNIST egy viszonylag nagy [adatbázis](#) ami tanítóanyagokat biztosít, például, olyan hálózatok(szoftveres) tanítására amiknek feladata, hogy képelemzési technikákkal és háttértudások(tanult minták alapján) segítségével elemezni tudjanak bemeneti képeket. Az adatbázis kézírott arab számokat tartalmaz.

A Tensorflow egy nyílt forrású matematikai könyvtár amit a gépi tanulásban, például neurális hálózatoknál, használnak. Mivel ezeknek a témaköröknek rengeteg erőforrásra van szükség, hogy érdemi munkát hajtsanak végre a Tensorflow képes egyszerre több CPU-n és GPU-n futni(például CUDA segítségével).

Tensorflow vagy hasonló könyvtár nélkül is implementálhatunk különféle gépi tanuláshoz kapcsolódó algoritmust akármilyen programozási nyelvben. Azonban Tensorflow már eleve optimalizált és tesztelt algoritmusokat biztosít a számításainkhoz, saját kóddal *from scratch* nem 100%, hogy elérjük a teljesítményüket. Egyedül a neurális hálózat megírása marad ránk.

Az adatbázist felhasználva 60.000 képpel fogjuk kioktatni a hálózatunkat és 10.000-rel megpróbáljuk megközelíteni, hogy mennyire pontos volt.

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
import argparse
# Import data
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)
import matplotlib.pyplot
FLAGS = None
def main(_):
    mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y = tf.matmul(x, W) + b
```

Az adatok (MNIST adatbázis) behívása és eltárolása után inicializáljuk lépésekben a Tensorflow könyvtárat.

Egy neurális hálózat rétegekből épül fel, lásd Perceptron feladat, ezeket a rétegeket be kell állítanunk annak megfelelően ami el akarunk érni a neurális hálózat működése során. A réteg a bemeneti adatokból különféle információkat olvasnak ki.

Létrehozunk egy tenzort x , ebbe fogjuk küldeni az értékeket, beállítjuk, hogy 784 pixelt tartalmazó képeket fog kapni. Ezeken felül létrehozunk az y értékeit adó függvényhez még szükséges W (súly) és b (bias) változókat. Az y értékei 0-1. Ezzel a softmax modellt hoztuk létre.

```
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
# The raw formulation of cross-entropy,
#
#   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
#                                 reduction_indices=[1]))
#
# can be numerically unstable.
#
# So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
```

```
# outputs of 'y', and then average across the batch.
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits( ↵
    labels = y_, logits = y))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize( ↵
    cross_entropy)
```

A `cross_entropy` sorban a *cost* függvényt állítjuk be, ez arra szolgál, hogy megtudjuk mennyi a különbség az eredeti érték és a becsült érték között, ezt az értéket akarjuk minimalizálni.

Lépésekben történik a tanítás és egy beépített `GradientDescentOptimizer` függvényt használunk fel tanításra. Ez a függvény úgy működik, hogy van egy kezdeti érték amit frissítget egészen addig amíg a *cost* függvény el nem ér egy minimumot, például a legmagasabb pontossági értéket. Ebből látjuk, hogy iteráció függő lesz ez a kalkuláció.

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

Inicializáljuk a sessionünket, feldaraboljuk az adatot batch-ekre tehát részekre és futtatjuk a modellünket. Közben a jelenlegi pontosságot is kiíratjuk.

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

print("-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a ↵
    tovabblepeshez csukd be az ablakat")

img = mnist.test.images[42]
image = img
matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm ↵
    .binary)
matplotlib.pyplot.savefig("4.png")
matplotlib.pyplot.show()
classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
print("-----")
```



```
print("-- A MNIST 11. tesztkepenek felismerese, mutatom a szamot, a ←  
      tovabblepeshez csukd be az ablakat")  
img = mnist.test.images[11]  
image = img  
matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm. ←  
      binary)  
matplotlib.pyplot.savefig("8.png")  
matplotlib.pyplot.show()  
classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})  
print("-- Ezt a halozat ennek ismeri fel: ", classification[0])  
print("-----")  
if __name__ == '__main__':  
    parser = argparse.ArgumentParser()  
    parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ ←  
      mnist/input_data',  
                        help='Directory for storing input data')  
    FLAGS = parser.parse_args()  
    tf.app.run()
```

Az elkészült hálózatunkat teszteljük az MNIST adatbázis néhány képével, az output alapján sikeresen működik és működtettük a Tensorflow projektünket.

A kiválasztott képet megjelenítjük külön ablakban is, hogy összetudjuk mi is hasonlítani a képen látható objektumot az eredmény szerinti objektummal.

8.2. Mély MNIST



PASSZOLÁS 1 - UDPROG SMINST FOR HUMANS feladat alapján

Az UDPROG SMINST for humans bejegyzés feladatát teljesítettem, elértem a level 10-s szintet a felmérésben, így felhasználom egyik passzolási lehetőségemet ennél a feladatnál. Összesen 2 passzolás járt lvl 9 és lvl 10-ért lásd lenti link.

BEJEGYZÉS POSTJÁNAK LINKJE

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ



PASSZOLÁS 2 - UDPROG SMINST FOR HUMANS feladat alapján

Az UDPROG SMINST for humans bejegyzés feladatát teljesítettem, elértem a level 10-s szintet a felmérésben, így felhasználom az utolsó passzolási lehetőségemet ennél a feladatnál. Összesen 2 passzolás járt lvl 9 és lvl 10-ért lásd lenti link.

BEJEGYZÉS POSTJÁNAK LINKJE

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

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

Tanulságok, tapasztalatok, magyarázat...

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat...

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?] z emberek folyton modellekben gondolkodnak, főleg problémák megoldása közben. Ez teljes mértékben igaz a programozásra is, mivel ebben problémákat kell megoldanunk, melyek megoldását nagyban elősegíti egy előre felállított modell, működési elv. A számítógépek processzorai egy előre meghatározott nyelvet felismernek, viszont több magas szintű programnyelv létezik, ezért az ezen nyelveken írt programokat le kell fordítanunk, erre szolgálnak segítségül a fordító programok (gcc, g++ ...stb.) Ezek a fordítóprogramok lexerrel avagy lexikális elemzővel felszereltek, melyek a lefordítandó programkódot hivatottak ellenőrizni. Illetve a fellépő hibákat (legyen az szintaktikai, vagy más egyéb) a felhasználó/programozó felé jelezni. A programnyelveket osztályokba sorolhatjuk, miszerint léteznek : - Imperatív nyelvek : algoritmikus nyelv, utasítások sorozata - Deklaratív nyelvek: pl.: logikai nyelvek - Más nyelvek : ide azokat soroljuk, melyek egy imperatív tényezőt/tulajdonságot tagadnak

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

10.3. Programozás

[BMECPP] Minden programkód, karakterekből áll elő, ezek egy adott karakterkészletből érhetőek el, minden nyelv definiálja a sajátját. (betűk, számjegyek, stb..) Ily módon épülnek fel a lexikális -, a szintaktikai -, a program -, a fordítási egységek, az utasítások, illetve maga a program. A programkódon belüli lexikális egységek, melyeket a fordítóprogram tokenizál a következők : Többkarakteres szimbólum, szimbolikus név, címke, megjegyzés, literál. A szimbolikus neven belül találkozhatunk az azonosítóval, melynek kötelezően betűvel kell kezdődnie, majd betűvel vagy számmal folytatódnia. Programozói eszközök megnevezésére szolgál, majd a későbbiekben az ezekre való hivatkozást is megkönnyíti. Találkozhatunk még a programnyelvekben kulcsszavakkal, melyeknek az adott nyelv tulajdonít jelentést, tehát azonosítói szempontból kötöttek. Általában utasítások elnevezései. Léteznek még standard azonosítók, melyeknek szintén

az adott programnyelv tulajdonít jelentést a kulcsszavakhoz hasonló módon, ezt a programozó átnevezheti, átértelmezheti (C-ben ilyen például a NULL). A programkódok egy hasznos egységét képezik a megjegyzések, kommentek. Ezek segítik a programozónak egy adott szegmens működési elvének leírását, ha esetleg hosszabb a kód és a későbbiekben vissza kell térni rá, de már a programozó nem teljesen emlékszik rá. Problémák feljegyzése adott szegmensekkel kapcsolatban. A kommentek szinte bármilyen nagyságúak lehetnek, nem befolyásolják a program működését.

DRAFT

III. rész

Második felvonás

**Bátf41 Haxor Stream**

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

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

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, Tihamér, *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 UDPROG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

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