

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. május 08, v. 0.0.4

Copyright © 2019 Dr. Bátfai Norbert

Copyright © 2019 Ladányi Balázs

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

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	Ladányi, Balázs	2019. május 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-03-04	Első fejezet kész.	Balázs
0.0.6	2019-03-11	Második fejezet kész.	Balázs
0.0.7	2019-03-17	Harmadik fejezet kész.	Balázs
0.0.8	2019-03-28	Negyedik fejezet kész.	Balázs

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.9	2019-03-30	Ötödik fejezet kész.	Balázs
0.1.0	2019-04-07	Hatodik fejezet kész.	Balázs
0.1.1	2019-04-12	Hetedik fejezet kész.	Balázs
0.1.2	2019-04-19	Nyolcadik fejezet kész.	Balázs
0.1.3	2019-04-25	Kilencedik fejezet kész.	Balázs
1.0.0	2019-05-08	Első teljes kiadás.	Balázs

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	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	5
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	7
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	8
2.6. Helló, Google!	8
2.7. 100 éves a Brun tétel	8
2.8. A Monty Hall probléma	9
3. Helló, Chomsky!	11
3.1. Decimálisból unárisba átváltó Turing gép	11
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	11
3.3. Hivatkozási nyelv	11
3.4. Saját lexikális elemző	12
3.5. l33t.1	12
3.6. A források olvasása	13
3.7. Logikus	14
3.8. Deklaráció	14

4. Helló, Caesar!	16
4.1. double ** háromszögmátrix	16
4.2. C EXOR titkosító	17
4.3. Java EXOR titkosító	17
4.4. C EXOR törő	19
4.5. Neurális OR, AND és EXOR kapu	19
4.6. Hiba-visszaterjesztéses perceptron	20
5. Helló, Mandelbrot!	21
5.1. A Mandelbrot halmaz	21
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	22
5.3. Biomorfok	22
5.4. A Mandelbrot halmaz CUDA megvalósítása	23
5.5. Mandelbrot nagyító és utazó C++ nyelven	23
5.6. Mandelbrot nagyító és utazó Java nyelven	24
6. Helló, Welch!	26
6.1. Első osztályom	26
6.2. LZW	26
6.3. Fabejárás	27
6.4. Tag a gyökér	29
6.5. Mutató a gyökér	29
6.6. Mozgató szemantika	30
7. Helló, Conway!	31
7.1. Hangyaszimulációk	31
7.2. Java életjáték	31
7.3. Qt C++ életjáték	32
7.4. BrainB Benchmark	33
8. Helló, Schwarzenegger!	35
8.1. Szoftmax Py MNIST	35
8.2. Mély MNIST	35
8.3. Robotpszichológia	35

9. Helló, Chaitin!	36
9.1. Iteratív és rekurzív faktoriális Lisp-ben	36
9.2. Gimp Scheme Script-fu: króm effekt	37
9.3. Gimp Scheme Script-fu: név mandala	37
10. Helló, Gutenberg!	38
10.1. Programozási alapfogalmak - Pici könyv	38
10.2. Programozás bevezetés - KERNIGHANRITCHIE könyv	39
10.3. Programozás - BME C++ könyv	41
III. Második felvonás	42
11. Helló, Arroway!	44
11.1. A BPP algoritmus Java megvalósítása	44
11.2. Java osztályok a Pi-ben	44
IV. Irodalomjegyzék	45
11.3. Általános	46
11.4. C	46
11.5. C++	46
11.6. Lisp	46

Ábrák jegyzéke

2.1. Monty Hall paradoxon - Bátfai Norbert	10
3.1. A program futása	12
4.1. Háromszög - Bátfai Norbert	17
4.2. Igazságtábla	20
5.1. Mandelbrot halmaz	21
5.2. Julia halmaz	23
7.1. Életjáték	33
7.2. BrainB működés közben - Bátfai Norbert	34

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?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

DRAFT

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/tree/master/turing/v%C3%A9gtelenciklus>

Tanulságok, tapasztalatok, magyarázat: Az első dolgunk, hogy egy végtelen ciklust csinálunk, ami leterheli az egyik magunkat maximumra. Ahhoz, hogy a terhelést megszüntessük, be kell iktatnunk egy `sleep()` függvényt, és ehhez includolnunk kell az `unistd.h` könyvtárat. Végül ha az összes magot akarjuk dolgoztatni, szükségünk van az OpenMp-re. Includeolnunk kell az `omp.h` könyvtárat és hozzá kell írunk a `#pragma omp parallel` a kódunkhoz, és azon belül kell elhelyezni a végtelen ciklust.

```
#include <stdio.h>
#include <omp.h>
int main() {
#pragma omp parallel { /* ezzel tudjuk kiküldeni a terhelést minden ↵
    magra */
while(1){}
}
return 0;
}
```

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

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

Megoldás videó:

Megoldás forrása: 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 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: Ilyen Lefagy függvényt nem lehet írni, mert ha a T1000 program lefagy, akkor a függvényünk nem fagy le és true értéket ad vissza, de ha a T1000 nem fagy le, akkor a programunk le fog fagyni.

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: <https://github.com/LadanyiBalazs/Prog1/blob/master/turing/valtozocsere.c>

Tanulságok, tapasztalatok, magyarázat: Felveszünk két változót, elnevezzük a-nak, és b-nek. Az a érték helyére felvesszük az a+b értékét, aztán b helyére a már megváltoztatott a-b értékét, végül az a helyére felvesszük a már megváltoztatott a-b értékét, és ezzel felcseréltük a két változó értékét.

2.4. Labdapattogás

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

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

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/tree/master/turing/labdapattog>

Tanulságok, tapasztalatok, magyarázat: Először le kell kérnünk az ablak méretét az initscr() függvénnyel majd segédváltozók segítségével meghatározzuk a labda helyzetét, a lépegetési értékét, és a terminálablakunk méretét. Hogy a programunk érzékelje a terminálablak méretének változtatását fel kell vennünk

egy `getmaxyx()` függvényt, majd kirajzoljuk a labdát az `mvprintw()` függvénnyel, ahol megadjuk a karakterünket is, ami ebben az esetben egy `O` betű. A `refrest()` függvénnyel frissítjük a programot, hogy a labda minden frissülésnél lépjen egyet. A `usleep()` függvénnyel pedig beállíthatjuk a frissülés sűrűségét, így a labda mozgásának gyorsaságát. `If` függvényekkel vizsgáljuk, hogy elérte-e a labda a terminálablakunk szélét, és ha elérte akkor az ellenkező irányba fog haladni amit a negatív értékekkel adtunk meg neki. Fordításnál hozzá kell rakni egy `-lncurses` kapcsolót a végére. Ez a feladat megírható `if()` függvény nélkül is, és akkor nem figyeli a terminálablak széleit.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/turing/sz%C3%B3hossz.c>

Tanulságok, tapasztalatok, magyarázat: Adott egy szavunk, ami jelen esetben az `1`, és ezt felvesszük az egyik változónkba. Ezt a változót addig shifteljük egyessével, amíg nullát kapunk és be nem telik a memóriánk. Ez után már nem tudunk többet írni, és megkapjuk a szó méretét bitben.

Fordítás: `gcc bitshift.c -o bitshift`

Futtatás: `./bitshift`

2.6. Helló, Google!

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

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat: A Pagerank algoritmus lényege, hogy kiszámolja egy oldalra-rá mutató, és arról kifelé mutató linkek értékét. Ezzel az algoritmussal dolgozik a Google is. Minden weboldalhoz egy számot rendel, ami segítségével rangsorolja őket. Ha egy oldalról hiperlink segítségével átkattintunk egy másik oldalra, akkor az a másik oldal ezért "pontot" kap. Most nekünk 4 oldalunk van, aminek ki kell számolni a pagerank értékét, amik egy négyelemű tömbben helyezkednek el. Három funkciónk van: a `kiir`, a `tavolsag` és a `pagerank` függvény. az első kiírja majd az értéket, a második az oldalak közötti távolság négyzetgyökét adja meg, a harmadik pedig kiszámítja a pagerank értéket. A mainben deklaráljuk a pagerank matrixot, és kiírjuk az értéket.

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

Tanulságok, tapasztalatok, magyarázat: A Brun-tétel azt írja le, hogy az ikerprímszámok reciprokaiból képzett sor összege véges vagy végtelen sor konvergens, ami azt jelenti, hogy ezek a törtek összeadva egy határt adnak ki pontosan vagy azt át nem lépve növekednek. A program ezeknek az ikerprímeknek összegzet reciprokait rajzolja ki.

```
library(matlab)
stp <- function(x) {
  primes = primes(x)
  diff = primes[2:length(primes)]-primes[1:length(primes)-1]
  idx = which(diff==2)
  t1primes = primes[idx]
  t2primes = primes[idx]+2
  rt1plust2 = 1/t1primes+1/t2primes
  return(sum(rt1plust2))
}
x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = stp)
plot(x,y,type="b")
```

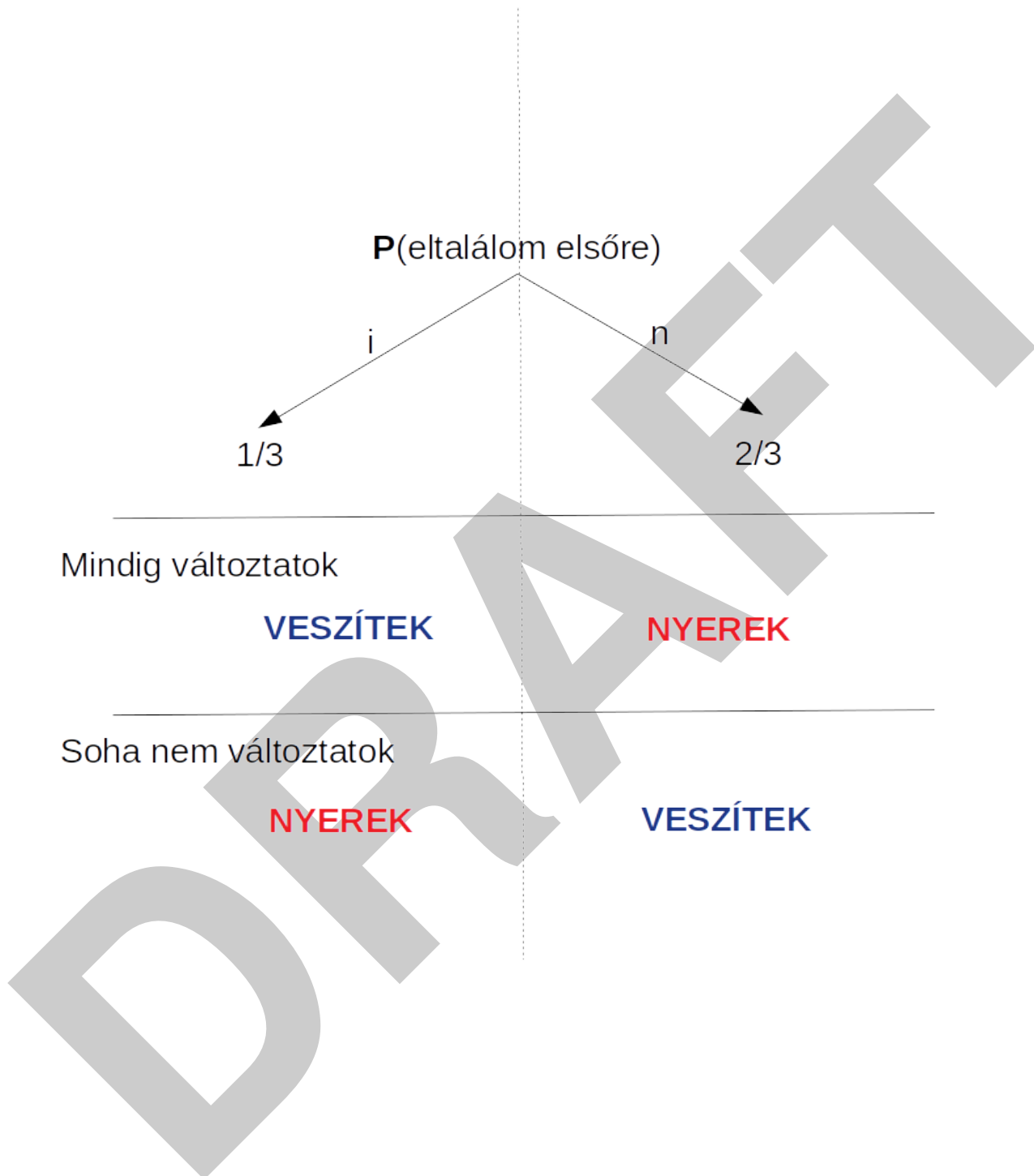
2.8. A Monty Hall probléma

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: A Monty Hall probléma arról szól, hogy van 3 ajtó, ami közül választhatunk (és az egyik mögött nyeremény van). Van lehetőségünk választani a háromból egyet, majd miután ezt megtettük a "játékmester" kinyit a maradék két ajtóból egyet ami mögött nincs semmi. Ezután a hátralévő 2 ajtóból nagyobb esélyünk van eltalálni azt az ajtót amelyik mögött a nyeremény van, hogyha változtatunk a döntésünkön.



3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/chomsky/unaris.c>

Tanulságok, tapasztalatok, magyarázat: Az i -t felvesszük 0-nak és addig léptetjük amég egyenlő nem lesz az adott számmal, majd kiírunk annyi "1" jelet, ahányszor léptettük. Vagyis amekkora a szám, annyi "1" jelet rakunk.

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:

Tanulságok, tapasztalatok, magyarázat: Passzolva Udprog SMNIST for HUMANS lvl 6 alapján.

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:

Tanulságok, tapasztalatok, magyarázat: A két fordítási szabvány, vagyis az C99 (ISO) és a C89 (ANSI) közötti különbséget legkönnyebben en for() ciklusban lehet bemutatni. C99 szerint a for() ciklusban egyszerre lehet deklarálni és léptetni is a változónkat. C89-ben ugyan ezt nem tehetjük meg. Ahhoz, hogy melyik szabványt alkalmazzuk, fordításkor a végére kell írunk, hogy -std=c99 vagy -std=c89.

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: <https://github.com/LadanyiBalazs/Prog1/blob/master/chomsky/lex.l>

Tanulságok, tapasztalatok, magyarázat: Meg kell határoznunk, hogy a lexerünk miként kategorizálja az egyes karaktereket, illetve stringeket. Először includoljuk a C nyelvben használt könyvtárat, ls felvesszük a változókat, amik számolni fogják, hogy a karakterekből és számokból mennyi fordul elő. Meg kell adnunk a lexerünknek, hogy milyen karakterhalmazokra mit adjon válaszul. Először az egyjegyű számokat nézi, aztán a betűket, a többi karaktert, a többjegyű számokat, majd a törtszámokat és végül a szavakat. A végén a programunk lezárja magát és kiírja a megoldást.

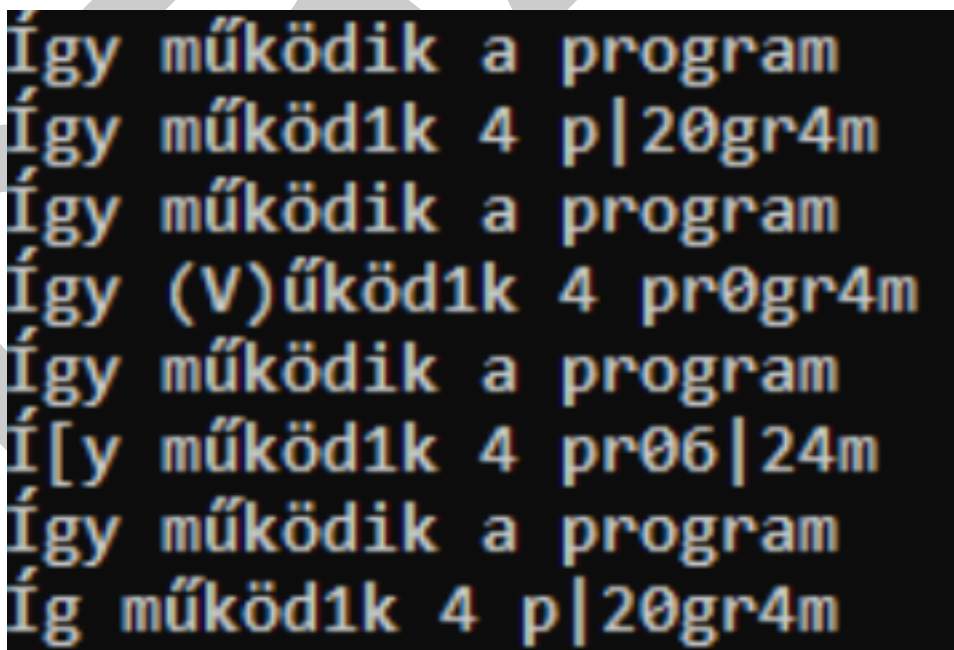
3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/chomsky/l33t.l>

Tanulságok, tapasztalatok, magyarázat: Meg kell határoznunk, hogy milyen karaktereket, milyen karakterekre cseréljen ki. A felhasználónak be kell gépelnie egy 50 karakteres szöveget, amit mi felveszünk egy char változóba. A kapott stringet karakterenként vizsgáljuk. A switch függvénnyel kicseréltetjük a kicserélendő karaktereket, végül kiíratjuk a végeredményt.



3.1. ábra. A program futása

Tutorált: Ilosvay Áron

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelolo)==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 jelkezelolo függvény kezelje. (Miótan a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)



Bugok

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

i.

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

ii.

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

iii.

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

iv.

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

v.

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

vi.

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

vii.

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

viii.

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

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/chomsky/signal.c>

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat: A signal manuálja leírja, hogy 2 értékre van szükségünk, egy signalra és egy handlerre. A signalunk ebben az esetben a 2-es, azaz a terminate signal, amit a CTRL+C gombkombinációval adhatunk ki Linux alatt. A voiddal megírjuk a kezelő függvényt, hogy a függvényünk kérjen egy int változót, ami tárolja az elkapott signal számát. A printf függvényen belül megadhatjuk mit írjon ki ha le akarjuk lőni.

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

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

Tanulságok, tapasztalatok, magyarázat: 1) Minden x -re van olyan y , hogy x kisebb mint y , és akkor y prím. Ez annyit jelent, hogy sok prímszám létezik. 2) Minden x -re van olyan y , hogy x kisebb mint y , és akkor y ikerprím. Ez annyit jelent, hogy sok ikerprím létezik. 3) Van olyan y , amire minden x prím, ebből következik, hogy x kisebb mint y . 4) Van olyan y , amire minden x nagyobb, mint y , ebből következik, hogy x nem prím. A 3) és 4) annyit jelent, hogy véges sok prímszám van.

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:

Tanulságok, tapasztalatok, magyarázat: A fent látható deklarációt, ha belerakjuk egy mainbe, lefordul. Az `int` (integer) a programozási nyelvben az egészekre vonatkozó változó.

4. fejezet

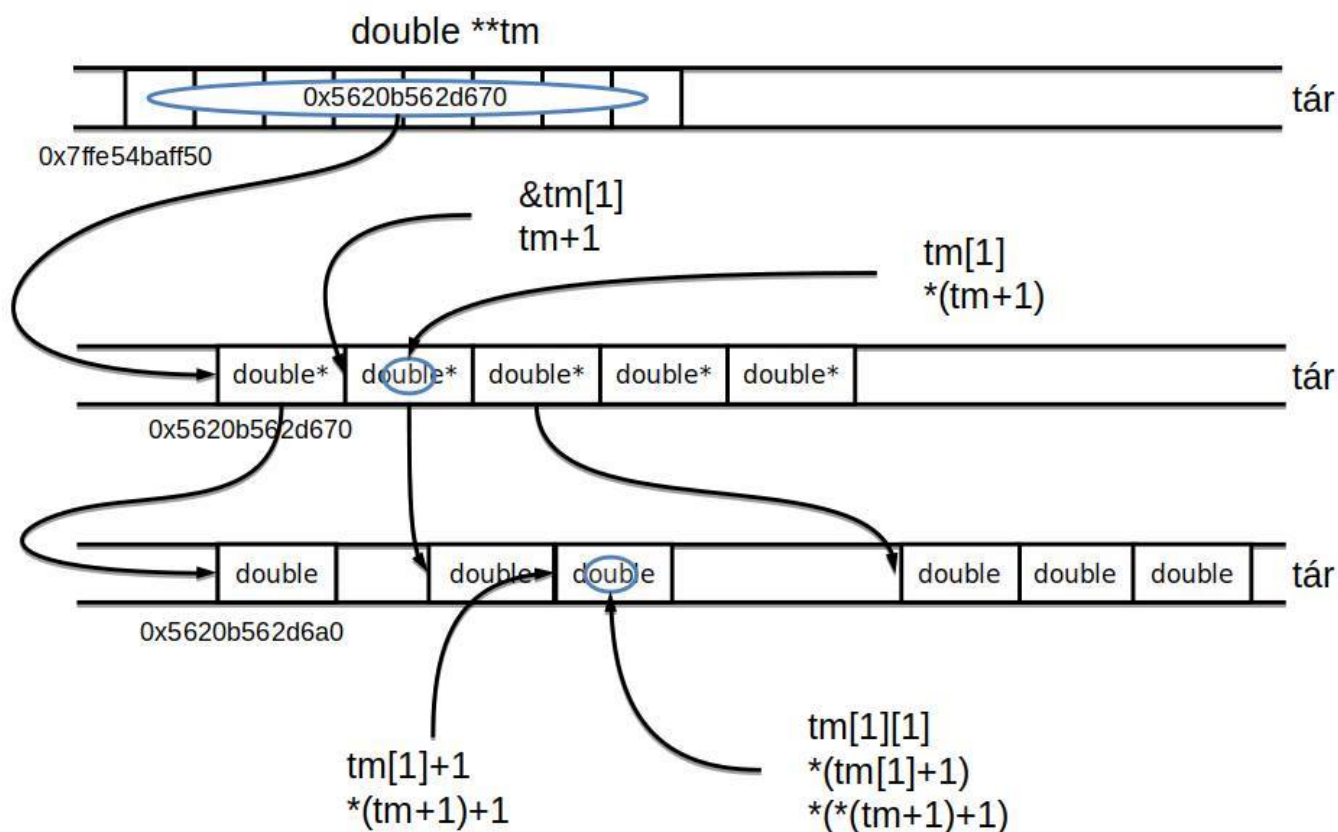
Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Ceasar/double.c>

Tanulságok, tapasztalatok, magyarázat: A háromszögmátrixunk 5 sorból fog állni. A malloc függvényt használjuk, amivel megvizsgáljuk, hogy van-e elég hely a memóriában a műveleteknek, és ez egy pointert fog nekünk visszaadni. Ha nincs elég memória a program kilép, és 1-es hibakódot ad ki. A for cikluson addig megyünk amég a sorok számán végig nem érünk, ami a mi esetünkben 5. Mindíg egyel arrébb mutatunk a memóriában, minden sorba egyel több helyet foglalunk le. Kiírjuk a tm[] tömb első elemét, és elkészítjük a for ciklus segítségével a mátrixunkat, majd ki is íratjuk.



4.1. ábra. Háromszög - Bátfai Norbert

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Ceasar/exor.c>

Tanulságok, tapasztalatok, magyarázat: A C exor titkosító a megadott kulcsunk alapján, összekeveri a szövegben lévő karaktereket, és egy értelmetlen "szöveget" ad vissza. A `while()` ciklusban beolvassuk az inputfájlt, majd a `for()` ciklus végigmegy a beolvasott szöveget és exorosán titkosítja a kulcs alapján. Végül ezt kiadjuk az outputnak.

Tutorált: Nagy László Mihály

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Ceasar/exor.java>

Tanulságok, tapasztalatok, magyarázat: A Java és C titkosító közt nincs nagy különbség. Lényegében csak annyi, hogy itt az exor függvényt classba kell rakni.

```
public class ExorTitkosito {

    public ExorTitkosito(String kulcsSzoveg,
        java.io.InputStream bejovoCsatorna,
        java.io.OutputStream kimenoCsatorna)
        throws java.io.IOException {

        byte [] kulcs = kulcsSzoveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBajtok = 0;

        while((olvasottBajtok =
            bejovoCsatorna.read(buffer)) != -1) {

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

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

            }

            kimenoCsatorna.write(buffer, 0, olvasottBajtok);

        }

    }

}
```

Ezután a mainbe meghívjuk a classt és a végén elkapjuk a try-catch blokkal a hibákat, ha van.

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

    try {

        new ExorTitkosito(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: <https://github.com/LadanyiBalazs/Prog1/blob/master/Cesar/toro.c>

Tanulságok, tapasztalatok, magyarázat: A program abc sorrendbe végigmegy az összes lehetőségen, hogy megtalálja a megfelelőt. Közbe folyamatosan ellenőrizzük, hogy a szövegünk megfelel-e már a lista_lehet funkción belül megtalálható szavaknak. Először megadjuk a kulcsméretet, az olvasási buffert és a beolvasó titkos szöveg maximális méretét. Includoljuk a megfelelő könyvtárakat. Az átlagos_szohossz megadja, mekkora a szövegben előforduló átlagos szóhossz, és et vissza adja egy double változóban. A tiszta_lehet funkció abban segít, hogy ne kelljen megnézni az összes lehetséges kulcsot. Az exor függvény töri fel a szöveget, az exor_tores pedig nézi, hogy sikerült-e. A mainbe meghívjuk a függvényeket, beolvassuk a titkosított szöveget, majd végigmegyünk az összes lehetséges kulcson. Közben a tiszta_lehet funkció ellenőrzi, hogy tiszta-e már a szövegünk.

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

Tanulságok, tapasztalatok, magyarázat: A neuron egy olyan agysejt, ami összegyűlt az elektromos jeleket, feldolgozza, majd szétküldi. Akkor küldi tovább, hogyha összegyűlt nála, annyi jel, ami meghaladja a küszöböt. Mi a kapuval foglalkozunk, ami az aktivációs függvény. Az OR és az AND kapu teljesen független egymástól, így beírhatjuk mind a kettő is, és ugyan azon hálózaton két kimenete lesz. Az EXOR-nál csak át kell írni az OR-t vagy az ORAND-et EXOR-ra. Beállíthatjuk a kódba, hogy a rejtett részeket is mutassa, és akkor sokkal részletesebb ábrát kapunk, amit ezzel a kóddal tehetünk meg: **hidden=c(6,4,6)**.

OR

b \ a	1	0
1	0	1
0	1	0

AND

b \ a	1	0
1	1	0
0	0	0

XOR

b \ a	1	0
1	1	1
0	1	0

4.2. ábra. Igazságtábla

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Ceasar/Perceptron.cpp>

Tanulságok, tapasztalatok, magyarázat: A perceptron gépi tanulást végző algoritmus, ami képes arra, hogy két lineárisan szétválasztható bemeneti halmazt szétválasszon. Egy lineáris kombinációt valósít meg, ami segíti a hibaképzést.

5. fejezet

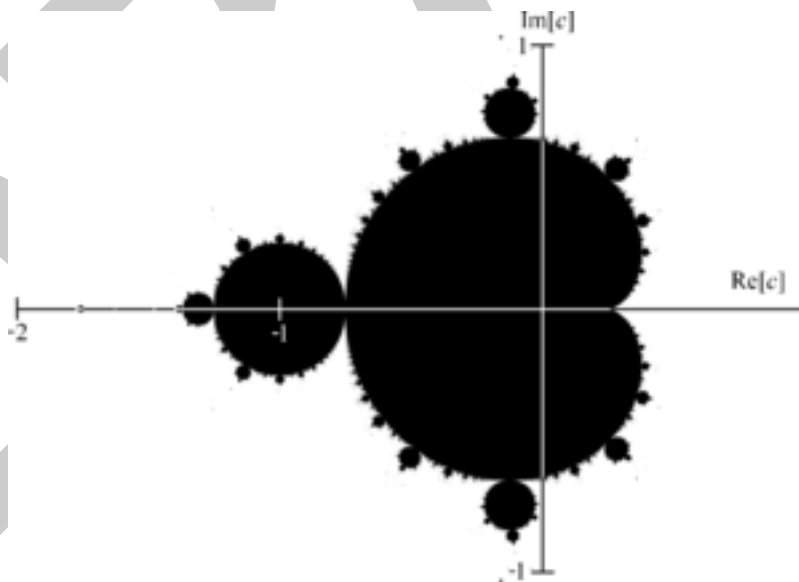
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Mandelbrot/halmaz.cpp>

A mandelbrot halmaz azokból a c komplex számokból áll, amelyekre az alábbi sorozat nem divergál: $x_1 := c$ és $x_{n+1} = (x_n)^2 + c$. Ezzel a programmal elkészítjük a mandelbrót halmaz kétdimenziós ábráját. A for ciklussal végigmegyünk a magasságon és a szélességen, majd kiszámítjuk a csomópontokhoz tartozó komplex számokat. A while ciklusban az iterációkat számoljuk ki, és amíg ez nem éri el a határt, addig nincs benne a halmazba, ha viszont elérte akkor benne van. A fordításhoz szükségünk van egy libpng könyvtárhoz, és includolnunk kell a png++/png.hpp header-t.



5.1. ábra. Mandelbrot halmaz

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

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Mandelbrot/complex.cpp>

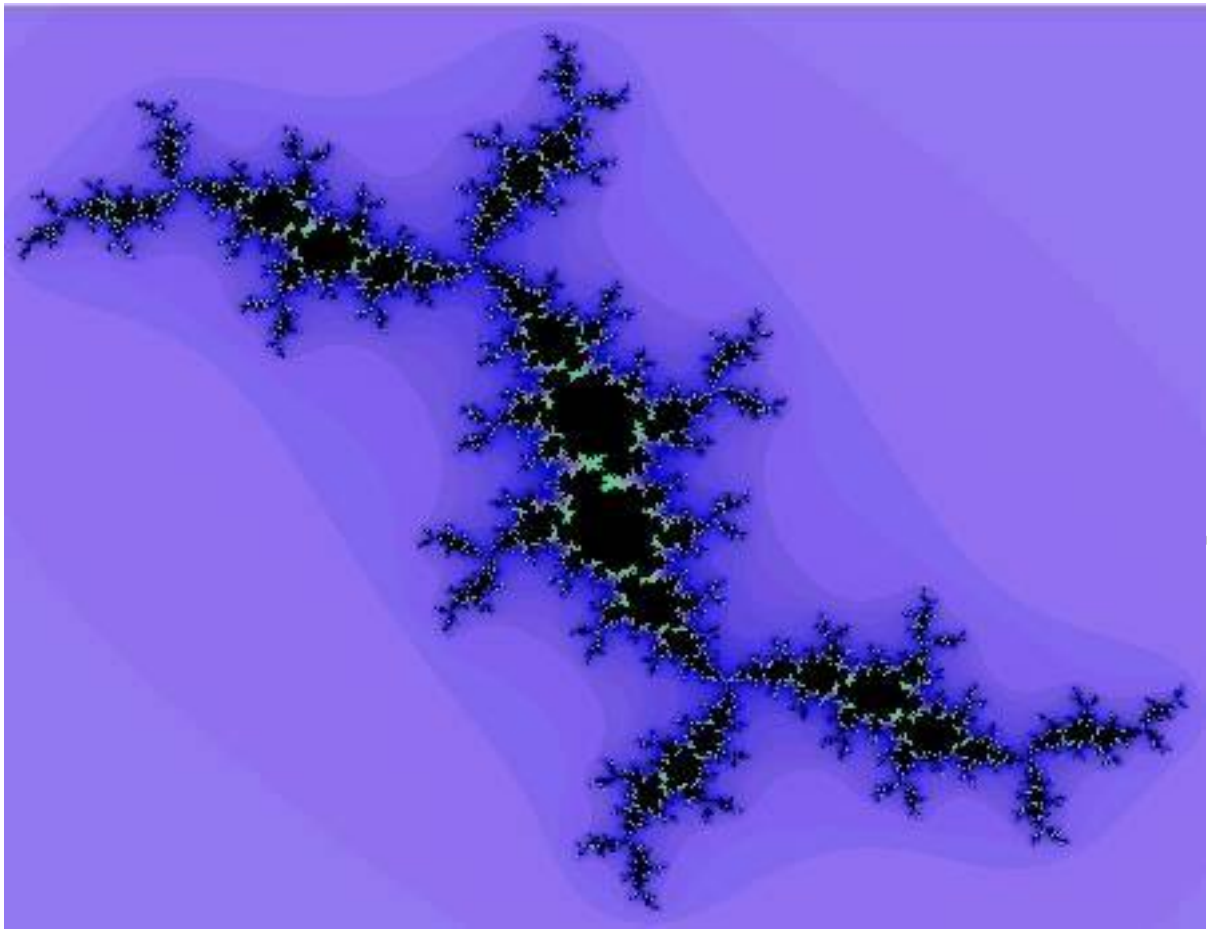
Az előző feladatot fejlesztjük tovább, megadott paraméterekkel. A mainben felvesszük a két változónkat (`argc`, `*argv[]`), amire azért van szükség, mert megadott adatokkal dolgozunk. Deklaráljuk a kép szélességét, magasságát és iterációs határát. Felvesszük az x és y tengely minimum és maximum értékét. Az `argv[0]` lesz a legenerált fájlformátumunk, tehát amilyen formátuma mentjük úgy kell futtatni. Most jön képbe a `png++` könyvtár, aminek megmondjuk a kép magasságát, és szélességét. A `for` ciklusunkat végigmegyünk a számokon, és kiszínezzük a mandelbrot halmaz alapjául szolgáló sorozatot. Ha minden sikeresen lefutott, elmentjük a képet az `argv[1]`-be megadott néven. Közbe van egy százalékmérő, hogy a felhasználó tudja hol tart a program.

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat: Az előző feladat programján kell kisebb módosításokat végeznünk. Átneveztünk és hozzáadtunk néhány változót, így visszavetíthető a komplex számsíkon. Az R pedig egy tetszőleges valós szám. Ki is kell bővíteni az argumentumtartományunkat az új változóknak megfelelően. Hozzáadunk egy komplexelő függvényt ami a `reZ` és `imZ` változókat veszi alapul. Ha a `z_n` (amit harmadikra emeltünk, és hozzáadtuk a `cc-t`) nagyobb, mint az R , akkor leáll a for ciklus, és az iteráció az i lesz. Ezután már csak le kell generálnunk a képet.



5.2. ábra. Julia halmaz

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Mandelbrot/CUDA.cpp>

Ez a program a videokártyánk CUDA magját fogja használni a mandelbrot halmaz legenerálásához. Annál gyorsabban fog generálódni, minél több CUDA magunk van. Ez csak NVIDIA kártyákkal lehetséges, azért találták ki, hogy grafikai feladatokat gyorsabban lehessen végrehajtani. Szükség lesz a libqt4-dev csomag letöltésére, amit linuxon a "sudo apt install libqt4-dev" paranccsal tehetünk meg. Három parancsot kell beírunk: qmake -project - ami létrehoz egy .pro file-t. Ebbe bele kell írni, hogy QT += widgets, majd erre a file-ra kiadjuk neki a qmake parancsot. Ezután tuffjuk futtatni a programot, és egérrel lehet benne nagyítani is. Az n billentyű lenyomásával kiszámolhatjuk a komplex számot, ami egy részletesebb képet ad.

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: https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazsal

Mandelbrot halmaz, de most bele tudunk nagyítani néhány helyen. Ehhez szükségünk lesz a libqt4-dev csomagra (sudo apt-get install libqt4-dev), valamint 5 bemeneti file-ra, aminek egy mappában kell lennie. Kiadjuk a qmake parancsot ami csinál nekünk egy Make file-t, amit a make parancsal futtatunk. Majd a ./Frak parancsal futtatjuk a programot. 4 ablakot kapunk amiből 3 be van egy random helyre zoomolva.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Mandelbrot/nagyito.java>

Az előző feladatban használt C++ nagyítót most Java-ban implementéljuk. Ehhez szükségünk van "sudo apt-get install openjdk-8-jdk"-ra. Nagyításkor az egérrel választjuk ki, hogy hol nagyítsunk bele a programba. A ball klikk lenyomásával rámegyünk a Mandelbrot halmaz valamelyik részére, és nagyítunk, majd rámegyünk es másíkra, és azt is nagyítjuk, és így tovább. Ha a nagyítások után veszítenénk a halmazunk pontosságából, az n gomb lenyomásával növelni tudjuk a számítások határát, így a pontosság is nő. Amíg fut a program, az s billentyű lenyomásával lészíthetünk képet, amit ugyan abba a mappába ment el.

```
public void pillanatfelvétel() {
    // Az elementendő 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 hálóval és az
    // aktuális nagyítási pontossággal:
    new MandelbrotHalmazNagyító(-2.0, .7, -1.35, 1.35, 600, 255);
}
}
```

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/polargen/>, <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezd/elsőjava/PolarGen.java>

Tanulságok, tapasztalatok, magyarázat: Java-ban szükségünk lesz az OpenJDK 6-os verzióra. A JDK src.zip állományban a java/util/Random.java forrásban láthatjuk, hogy a Sun programozói is hasonlóan oldották meg a feladatot. C++-ban pedig fontos, hogy egy számítási lépés két normális elosztású számot állít elő, vagyis minden második híváskor kell számolnunk, többinél pedig az előző eredményt visszatérítünk, ami sokban könnyíti munkánkat.

6.2. LZW

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

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Welch/lzw.cpp>

Az LZW egy veszteségmentes tömörítés algoritmus. A tömörítési eljárásnak az alapja az, hogy a kódoló csak egy szótárbeli indexet küld ki. A fa minden csomópontjának 2 részfa lehet, ami a bal és a jobb részfa. Megadunk a programnak egy szöveges fájlt, amit a az lebont binfa szerkezetre, majd ezt kiadja egy általunk megadott kimenetre. Létrehozuk a gyökeret, majd a gyerekeket balra és jobbra. Ahoz, hogy megtudjuk mennyi származtatás van, megnézzük az átlagos ághosszát, ami a szórás összege, valamint az átlagot és az mélységet. inicializálnunk kell az átla() függvényt és a szórást. A kiir függvény a fabejárásér felel. Végül felszabadítjuk az elemeket. Welch 8 bites sorozatát kódolja 12 bites kóddá. Inorder fabejárást alkalmazunk, ami azt jelenti, hogy a bal oldalt olvassuk be először, majd a rootot, és végül a jobb oldalt.

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/ziv/z_post.c#l229https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/ziv/z_pre.c

Az előző feladat után, itt meg kell változtatnunk a fabejárás sorrendjét. Először preorderre, ami annyit takar, hogy mostmár nem a bal oldallal kezdünk, hanem a rootal, és utána jön a bal és a jobb oldal. Majd átírjuk postorderre is, ahol pedig a bal oldallal kezdünk, a jobb oldallal folytatjuk, és a rootal fejezzük be. Minden bejárást külön-külön kell futtatni.

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

Univerzális programozás 65 / 92
}

output:
-----0 (1)
-----0 (2)
-----0 (3)
-----1 (4)
-----1 (3)
-----1 (4)
-----0 (5)
---/ (0)
-----1 (2)
-----0 (3)
-----1 (3)
-----1 (1)
-----0 (2)
-----1 (3)
-----1 (2)
depth = 5
mean = 3.33333
var = 1.0328
//preorder
void kiir (Csomopont * elem, std::ostream & os)
```

```

{
if (elem != NULL)
{
++melyseg;
for (int i = 0; i < melyseg; ++i)
os << "---";
os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
kiir ( elem->nullasGyermek (), os);
kiir ( elem->egyenesGyermek (), os);
--melyseg;
}
}
output:
---/ (0)
-----0 (1)
-----0 (2)
-----0 (3)
-----1 (4)
-----1 (3)
-----1 (4)
-----0 (5)
-----1 (2)
-----0 (3)
-----1 (3)
Univerzális programozás 66 / 92
-----1 (1)
-----0 (2)
-----1 (3)
-----1 (2)
depth = 5
mean = 3.33333
var = 1.0328
//postorder
void kiir (Csomopont * elem, std::ostream & os)
{
if (elem != NULL)
{
++melyseg;
for (int i = 0; i < melyseg; ++i)
os << "---";
kiir ( elem->nullasGyermek (), os);
kiir ( elem->egyenesGyermek (), os);
os << elem->getBetu () << "(" << melyseg - 1 << ")" << std::endl;
--melyseg;
}
}
output:
-----0 (1)

```

```
-----0 (2)
-----0 (3)
-----1 (4)
-----1 (3)
-----1 (4)
-----0 (5)
-----1 (2)
-----0 (3)
-----1 (3)
-----1 (1)
-----0 (2)
-----1 (3)
-----1 (2)
---/ (0)
depth = 5
mean = 3.33333
var = 1.0328
```

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!

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Welch/lzw.cpp>

Az LZWBinFa-t bele kell ültetni egy C++ osztályba, amihez kellenifog nekünk egy LZWBinFa osztály. Létrehozunk egy beágyazott csomópont osztályt, amibe belerakjuk a fa egy csomópontjának jellemzését. Ezt azért csináljuk, mert a fa részeként szeretnénk vele számolni, ebből adódóan nem szánunk neki külön helyet. Ebben az osztályban a fa gyökere nem egy pointer, hanem egy objektum, ami a "/" szimbólumot tartalmazza, maga a fa viszont pointer, és az LZW fa azon csomópontjára mutat, amit az LZW algoritmus mond. Ez a konstruktor ráállítja a gyökérre a fa mutatót. Túl kell terhelni a operátort, hogy beadhassuk a fának a b inputot, ami "0" és "1" karakter lehet.

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Welch/z3a8.cpp>

Az előző forrásban a gyökeret objektumként kezeltük, most azonban mutató lesz ez is, a fával együtt. Mindenhol törölnünk kell a referencia jeleket, mivel mostmár referenciaként adjuk át a gyökeret.

```
LZWBinFa ()
{
    gyoker = new Csomopont ();
```



```
fa = gyoker;  
}
```

A szabadit() függvénybe mutatót kell állítanunk a gyermekekre.

```
~LZWBInFa ()  
{  
    szabadit (gyoker->egyenesGyermekek ());  
    szabadit (gyoker->nullasGyermekek ());  
    szabadit (gyoker);  
}
```

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Welch/mozgato.cpp>

Létrehozunk egy operátort, ami lemásolja saját magát, abban az esetben ha nem a gyökeret tartalmazza. A rekurzioIndutasa függvény indítja el a rekurziót, ha van nullás gyermeke akkor azon fut tovább, ha van egyes gyermeke akkor arra is meghívásra kerül. A fő eljárást maga a rekurzioAzAgakon függvény végzi el, ez fut át az összes ágon, és létrehozza az új csomópontokat. Az alap BinFa másolódik a BinFa2-be.

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: <https://github.com/LadanyiBalazs/Prog1/tree/master/Conway/Ant>

Tanulságok, tapasztalatok, magyarázat: Fordításkor a qmake-et használjuk amit a letöltött QT keretrendszerben találhatunk. A P billentyű lenyomásával lehet pause-olni, vagyis szüneteltetni, a Q lenyomásával pedig kilép a programból.

futtatási parancsok: **`~/home/.../qmake myrmecologist.pro , -make ./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s 3 -c 22`**

Az Ant osztályba a hangya tulajdonságai vannak megadva. A két koordinátája, x és y, a dir (direction) pedig az irányt határozza meg amibe tart. A + jelek azt jelölik, hogy public elérésűek, tehát az osztályon kívülről is el lehet érni őket, valamint látszódik is osztályon kívülről. A - jellek pedig private elérésűek, tehát csak az osztálon belülről lehet elérni őket. Az ants egy vektor, ami a hangyákat tárolja.

Az Antwin tulajdonságba az ablak mérete van megadva pixelben, szélesség és hosszúság. A cellwidth és a cellheight pedig a hangyák, vagy cellák méretét adják meg. Az antThread a hangyaszámításokat végzi. A grids kirajzolja a 2 rácsot, amiből a gridlx egyet tárol. A closeEvent() (kikapcsolási esemény) meghívja az AntThread osztályból a finish() (befejezés)-t ami leállítja az ablakot azzal, hogy a running-ot hamis értékre állítja. A keyPressEvent() (gombnyomás esemény) a gombnyomásokat dolgozza fel, a paintEvent() (festés esemény) pedig a hangyák színezéséért felel.

Az Antthread osztály hasonló az Antwin-hez, csak a tulajdonságok ki vannak egészítve. Az evaporation (párolgás) a hangyák számát tárolja egy cellában. Puklikus függvényeink a run(), a finish() és a isRunning() ami egy igaz vagy hamis értéket adhat vissza. Nem publikus a setPheromon() ami a feromonok beállítását szolgálja, MoveAnts() ami a hangyák mozgatásáért felel és a newDir() ami az irányt adja meg.

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/blob/master/Conway/életjatek.java>

Tanulságok, tapasztalatok, magyarázat: John Horton Conway életjátékának szabályai: van egy élettér, ami cellákból áll és sejtek lakják, minden cella egy sejt. Egy sejt akkor él ha legalább 2 élő szomszédja van, ha annál kevesebb, meghal. Egy halott sejt akkor éled fel, ha 3 élő szomszédja lesz. A sejtek egy halmazát alakzatnak hívjuk, az egyik ilyen a sikló. A sikló egy adott irányba halad, amit a siklóágyú lő ki a végtelenbe. A programot egy objektumként írjuk meg, amit később a main függvényben hívunk meg, paraméterként pedig átadjuk az objektum konstruktorának az oszlop és sor méretet. Fordításnál és futtatásnál a szöveggé kódolást UTF-8-ra kell állítani. `javac -encoding UTF-8 életjatek.java, -java életjatek`

Az "S" billentyűvel képet készíthetünk a sejttérről. Az "N"-el növegni tudjuk a sejtek méretét, "K" val pedig csökkenteni. A "G"-vel gyorsítjuk, az "L"-el pedig lassítjuk a szimulációt. Az egér jobb klikkjével felélesztjük vagy megöljük a sejtet.

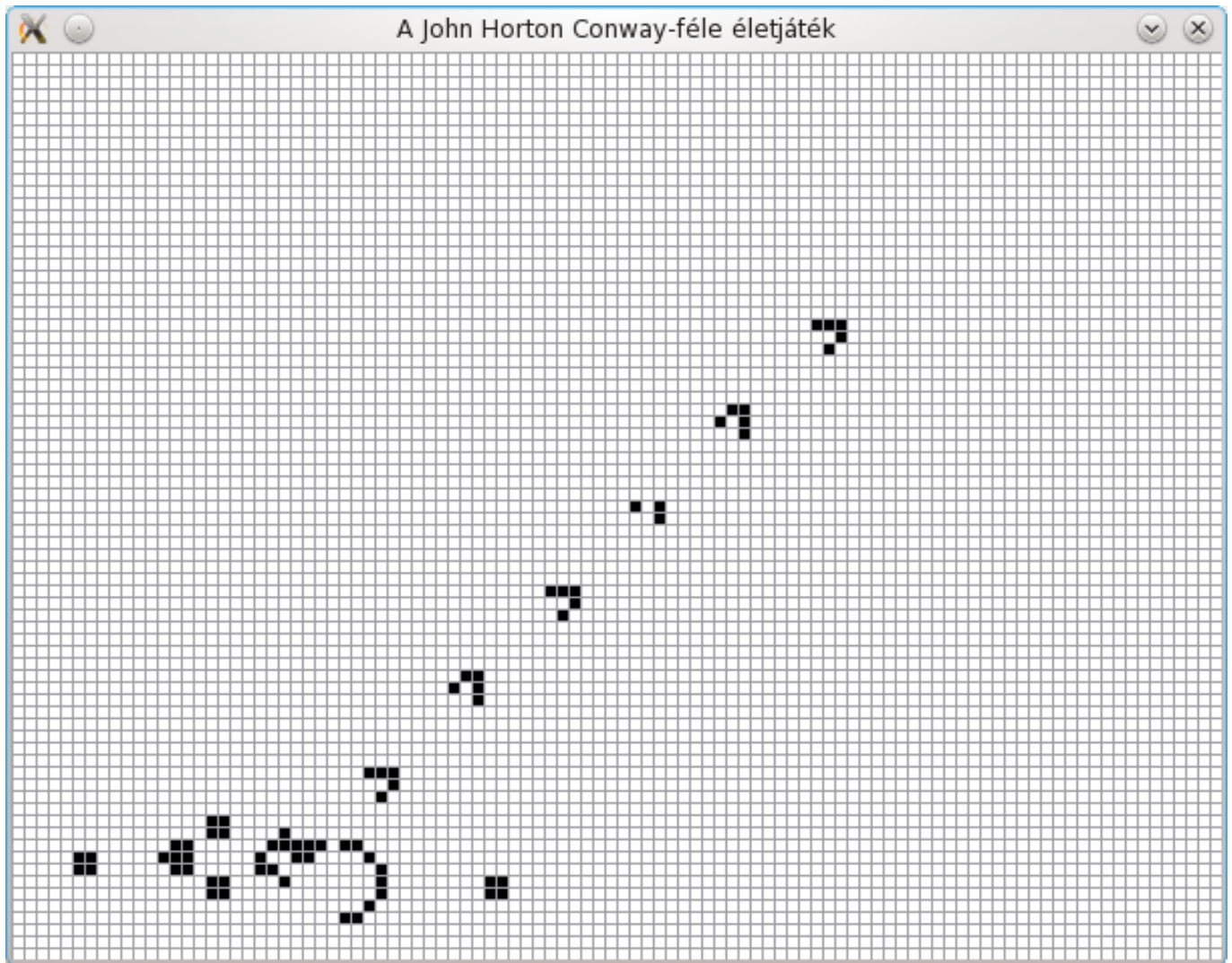
7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://github.com/LadanyiBalazs/Prog1/tree/master/Conway/Eletjatek>

Tanulságok, tapasztalatok, magyarázat: A szabályok ugyan azok, mint java-ban. A fordítás lesz különböző, mert a QT keretrendszert használunk. A **qmake -project** paranccsal legeneráljuk a .pro file-t. Majd ezt futtatjuk, ami legenerálja a make file-t. Végül a **make** paranccsal fordítjuk le a programunkat. Futtatni pedig **./életjatek** paranccsal tudjuk.



7.1. ábra. Életjáték

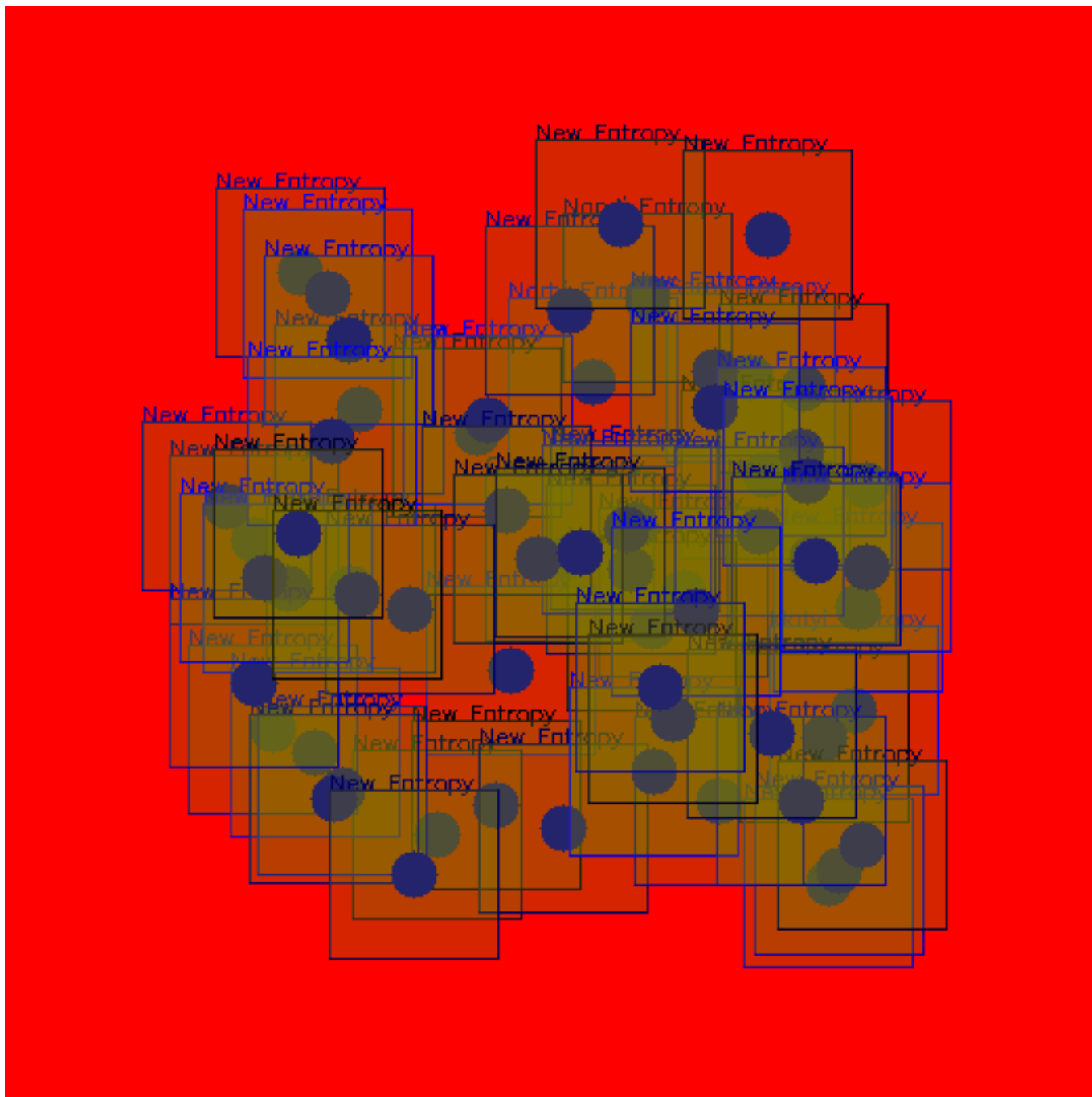
7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat: A program célja, az e-sportolók tesztelése. Az egérgombot lenyomva kell tartani és követni kell a négyzeten lévő kört. Minél tovább tudjuk benne tartani az egerünket annál több négyzet jelenik meg hogy bezavarjon. Ha elveszítjük eltűnnek a zavaró négyzetek és lassul a mozgás.

fordítás, futtatás: `/home/.../qmake brainB.pro, make, ./brainB`



7.2. ábra. BrainB működés közben - Bátfa Norbert

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat: Ehhez a programhoz szükségünk van egy Python fejlesztői környezetre, és a Tensorflowra, amit gépi tanuláshoz kell használni. Az MNIST egy olyan adatbázis, ami segítséget nyújt a tanításban képelemző programoknak és neurális hálózatoknak is. Ebben az adatbázisban képek vannak, amik mintaként segítenek a bemeneti képek elemzésében. Ebben az esetben 60000 képpel tanítjuk a hálózatot, és 10000 képpel, pedig vizsgáljuk a pontosságát.

A programot tanulása után teszteljük az adatbázisban lévő képek segítségével. A mi esetünkben ezek a hármastól-kilencesik terjedő képek. A futtatási parancs: **python softmax.py**

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat: Ezt a feladatot passzolnám.

8.3. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat: Ezt a feladatot passzolnám.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

A List nyelvet más gondolkodásmóddal kell programoznunk. A műveleteket úgynevezett fordított lengyel módszerrel írjuk fel, vagyis elől vannak a műveleti jelek és utána a számok. A zárójelek közti parancsokat veszi figyelembe. A függvények definiálásához a `define` kulcsszót írjuk a parancs elejére, utána a függvény nevét, majd a paramétereket.

A faktoriális iteratív módon, úgy számoljuk ki, hogy egy `do` ciklussal az `i`-t 1-től léptetjük és megszorozzuk a `num` értékével, ezt addig amég `i` nem lesz nagyobb `n`-nél.

```
> (define (fact n) (do ((i 1 (+ 1 i)) (num 1 (* i num))) ((> i n) ←  
    num)))  
fact  
> (fact 5)  
120
```

Rekurzíval egy kicsivel egyszerűbb a megoldás. Itt is először definiáljuk és elnevezzük a függvényt, majd az `if`-el megnézzük, hogy `n` kisebb-e mint 2. Ha `n` kisebb mint egy akkor beszorozza 1-el és véget ér a feladat.

```
> (define (fakt n) (if (< n 1) 1 (* n(fakt(- n 1)))))  
fakt  
> (fakt 5)  
120
```

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: A Script-fu termináljából tudjuk futtatni a programunkat, ahova bemásoljuk a forrást és megadjuk magát a szöveget, a betűtípust, a szövegnek a méretét, színét, a képnek a méretét és a színátmenetet is.

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

Majd adunk neki egy keretet, ahol szintén megadjuk pár paramétert.

```
(script-fu-bhax-chrome-border "Programozás" "Sans" 110 768 576 300 ' (255 0 ←  
0) "Crown molding" 6)
```

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: A mandala kirajzolása Lispben úgy történik, hogy a szöveget forgatjuk. Ahhoz, hogy gimpben tudjuk elvégezni ezt a feladatot, be kell másolnunk a Lisp forrást, a gimp program script mappájába. Meg tudjuk neki adni, hogy mit írjon ki, majd középre igazítjuk, és megadhatunk neki még más paramétereket is, mint például a betűméretet, betűtípust és színeket.

Futtatjuk a programot a Script-fu terminálból:

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


10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak - Pici könyv

Bevezetés:

A könyvünk azzal indít, hogy leírja a követelményt, ami a bevezetés az informatikába nevű tantárgy, és a folytatást is megadja.

1. Fejezet::

A programozási nyelvek alapjait tisztázza a könyv, megismerkedünk néhány alap fogalommal. Három részre osztjuk a programozási nyelveket: gépi nyelv, assembly szintű nyelv és magas szintű nyelv. A következő alfejezetben osztályokra bontjuk a programnyelveket: Imperatív nyelvek, Deklaratív nyelvek és máselví nyelvek. Ezeknek a csoportoknak tisztázzuk a tulajdonságait és alcsoportjait. Ezután A jegyzetben alkalmazott formális jelölésrendszereket nézzük meg, amelynek három típusa van: terminális, nem terminális és alternatíva. Az utolsó alfejezetben a programnyelvekről alapjairól van egy összegző.

2. Fejezet:

Ebben a fejezetben egy programozási nyelv alapeszközeit, alapfogalmait ismerjük meg. A fejezet elején megnézzük a karakterkészleteket, hogy miket lehet használni. A folytatásban a lexikális egységekről tanulunk. Olvashatunk a többkarakteres szimbólumokról, a szimbolikus nevekről, a címkékről, megjegyzésekről és iterálokról, rengeteg példával. A következő alfejezetünkben megnézzük a forrásszöveg összeállításának általános szabályait, majd az adattípusokat. először az egyszerű majd az összetett típusokat. Ezután tanulmányozzuk a nevesített konstansokat, a változókat és a fejezet végén részletesen végignézzük az alapelemeket az egyes nyelvekben.

3. Fejezet:

A harmadik fejezet a kifejezésekről szól. Elsőként olvashatunk az általános kifejezésekről, majd a C nyelvben lévő kifejezésekről.

4. Fejezet:

A negyedik fejezet az utasításokról szól, aminknek 9 fajtája van: 1. Értékadó utasítás, 2. Üres utasítás, 3. Ugró utasítás, 4. Elágaztató utasítások, 5. Ciklusszervező utasítások, 6. Hívó utasítás, 7. Vezérlésátadó utasítások, 8. I/O utasítások, 9. Egyéb utasítások.

5. Fejezet:

Az ötödik fejezetben a programok szerkezetéről van szó, ahol válaszokat kapunk általános kérdésekre. Megnézzük az alap programok felépítését. A fejezet további részében olvashatunk a hívási láncról és a rekurzióról, a másodlagos belépési pontokról, a paraméterkiértékelésről, paraméterátadásról, a blokkokról, a hatásköréről, a fordítási egységről és az egyenes nyelvek eszközeiről.

6. Fejezet:

A hatodik fejezet, ami egy kifejezetten rövid fejezet, az absztrakt adattípusokról szól, amik megvalósítják a bezárást vagy információ rejtést.

7. Fejezet:

: A hetedik fejezet a csomagról szól, ami az a programegység, amely egyaránt szolgálja a procedurális és az adatabsztrakciót. Ennek eszközei: típus, változó, nevesített konstans, saját kivétel, alprogram, csomag.

8. Fejezet:

A nyolcadik fejezet az ada fordításról szól. Megnézzük a pragákat majd a fordítási egységeket, ami adában lehet: alprogram specifikáció, alprogram törzs, csomag specifikáció, csomag törzs, fordítási alegység, valamint ezek tetszőleges kombinációja.

9. Fejezet:

A kilencedik fejezet a kivételkezelésről szól. Leginkább a PL/I kivételkezeléséről és az ada kivételkezeléséről.

10. Fejezet:

A tizedik fejezet a generikus programozásról szól, ami az újrafelhasználhatóság és így a procedurális absztrakció eszköze.

11. Fejezet:

A tizenegyedik fejezet a párhuzamos programozás elvégzéséről szól, és, hogy milyen eszközökre van szükség ehhez.

12. Fejezet:

A tizenkettedik fejezet a taszkról szól, ami szükséges a párhuzamos programozás megvalósításához. Erről részletesen olvashatunk, és példákkal is alá van támasztva.

13. Fejezet:

A tizenharmadik fejezet az input/outputról, és annak eszközeiről, ami a Fortran, Cobol, PL/I, Pascal, ADA és a C.

14. Fejezet:

A tizenegyedik és egyben utolsó fejezet az implementációs kérdésekről szól. Megnézzük különböző területeit és kapcsolóit.

10.2. Programozás bevezetés - KERNIGHANRITCHIE könyv

[KERNIGHANRITCHIE]

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

1. Fejezet:

A könyv az alapfogalmakkal és ismeretekkel kezdi el a bemutatkozást (változók, állandók, aritmetika, vezérlési szerkezetek, függvények, ill. az egyszerű adatbevitel és -kivitel), hogy az olvasó minél előbb képes legyen programot írni. A legegyszerűbb programmal kezdünk: a "Hello World!"-el. Annyi különbséggel, hogy itt "Hello mindenki!"-t írunk ki. A második feladatban a változókkal ismerkedünk meg, egy Fahrenheit-fok Celsius-fok táblázat kiíró programmal. Ebben a fejezetben olvashatunk a kommentekről is először. A következő feladatban a `for()` ciklusról van szó. Ugyan azt a táblázatot használjuk mint az előző feladatnál. A következő feladatban a szimbolikus állandókat nézzük meg. Az ötödik alfejezetünkben a karakterekkel foglalkozó függvényekkel foglalkozunk (beolvasás, kiírás, számlálása, sorok számlálása, szavak számlálása). A következő feladatban a tömböket tanulmányozzuk, a programunk megszámolja a karaktereket és üres helyeket egy beolvasott szövegben. Ezután függvényekkel dolgozunk, egy függvény segítségével hatványozunk. A fejezet végén megnézzük az argumentumokat és részletesebben tanulunk a tömbökről és a változókról.

2. Fejezet:

A második fejezetben a típusokról, operátorokról és fejezetekről van szó. Megismerkedünk a változónevekkel, adattípusokkal és méretekkel, állandókkal, deklarációkkal, aritmetikai operátorokkal, relációs és logikai operátorokkal, típuskonverziókkal, inkrementáló és dekrementáló kifejezésekkel, feltételes kifejezésekkel és a `precendia` és a kifejezés kiértékelési sorrendjével. Ebbe a fejezetbe már nem konkrét programokat írunk meg, hanem kódcsipeteken keresztül mutatja be a könyv a dolgokat.

3. Fejezet:

A harmadik fejezetben a vezérlési szerkezetek ismerkedhetünk meg. Bemutatja a könyv az olyan utasításokat mint `if-else`, `else-if`, `switch`, `while` és `for` ciklus, `do-while`, `break` és `continue`, `goto` és a címkék.

4. Fejezet:

A negyedik fejezetben a függvények és programok szerkezetét mutatja be a könyv, egy bővebb formában. Először megismerkedünk a függvényekkel kapcsolatos alapfogalmakkal, és egyszerű függvényekkel, majd rátérünk a nem egész értékkel visszatérő függvényekre is. Folytatjuk a külső változókkal és az érvényességi tartomány szabályaival, majd megnézzük a header állományokat, a statikus változókat, regiszterváltozókat, blokkstruktúrákat, változók inicializálását, rekurziót és végül a C előfeldolgozó rendszert.

5. Fejezet:

Az ötödik fejezetben a mutatókkal és tömbökkel ismerkedünk meg. A könyv bemutatja a mutatókat, címeket, függvényargumentumokat és tömböket három alfejezeten keresztül, és van néhány ábra is, amely sokat segíthet a megértésükben. Ezek után megnézzük a címaritmetikát, karaktermutatókat és függvényeket, mutatótömböket és mutatókat megcímző mutatókat, többdimenziós tömböket, mutatótömbök inicializálását, mutatók és többdimenziós tömböket, parancssor-argumentumokat, függvényeket megcímző mutatókat és megnézzük a bonyolultabb deklarációkat is.

6. Fejezet:

A hatodik fejezetben a fruktúrákkal ismerkedhetünk meg. Az elején kezdi a könyv, mint minden mással is. Először megismerkedünk az alapfogalmakkal, aztán továbbmegyünk az alap struktúrákra és függvényekre. Ezek után rátérünk a struktúratömbökre, struktúrákat kijelölő mutatókra, önhivatkozó struktúrákra, keresésre táblázatban, a `typedef` utasításokra, `union`okra és a bitmezőkre. Ennél a fejezetnél is vannak ábrák az elején, ami segíti a struktúrák megértését.

7. Fejezet:

A hetedik fejezetben az adatbevitelről és az adatkivitelről tanulhatunk. Ahogy már megszokhattunk az alapokkal indít a könyv, a standard adatbevitel és kivitellel. Aztán rátérünk már a formátumozott adatki-

vitelre a printf függvény segítségével, majd megnézzük a változó hosszúságú argumentumisták kezelését, formátumozott adatbevitelt a scanf függvénnyel, hozzáférést az adatállományokhoz, hibakezelést az stderr és exit függvényekkel, szövegsorok beolvadását és kiírását és a további könyvtári függvényeket. Bemutatja a könyv az alapvető hibákat, így nekünk már nem kell belefutnunk ezekbe.

8. Fejezet:

A nyolcadik és egyben utolsó fejezetben tanulmányozzuk a kapcsolódást a UNIX operációs rendszerhez. Ebben a fejezetben olyan dolgokkal ismerkedünk meg, mint az állományleírók, alacson szintű adatbevitel és adatkivitel amelyet a read és write függvényekkel segít elő, az open, create, close és unlink rendszerhívásokkal és a véletlenszerű hozzáféréssel az lseek függvénnyel. Az utolsó három alfejezetben pedig példákon keresztül mutatja be a könyv az fopen és getc függvények megvalósítását, katalógus kiírását és tárterület lefoglaló programot.

Összegzés:

Összességében a könyv jól bemutatja a C programnyelvhez szükséges dolgokat. Mindent az alapoktól indít, így nem szükséges a könyv olvasása előtt semmilyen tudással rendelkezni, ahhoz, hogy megértsük. Rengeteg példát és kódcsipetet használ a könyv, amelyek ráadásul be is vannak kommentezve, így mindenről világos, hogy mit csinál.

10.3. Programozás - BME C++ könyv

[BMECPP]

Kivételkezelés

A fejezetben először érzékelteti a könyv, a hagyományos hibakezelés problémáit, majd ismerteti a kivétel alapú megoldás szintaktikáját és mechanizmusát. Ezt megmutatja néhány gyakorlati példán keresztül

Mi most a kivételek használatának alapjaival fogunk foglalkozni. Először megismerkedünk ahagyományos hibakezeléssel, ahol egy hibás esetek kezelésére jellemző klasszikus megoldást láthatunk. Láthatjuk, hogy az összes hibás esetet a mainen belül kezeljük.

A későbbiekben a kivételkezelés alapjairól lesz szó, ami biztosítja, hogy, ha hibát találtunk, akkor a "futás" azonnal a hibakezelő ágon folytatódik. Azonban nem csak hibák, hanem bármely kivételes esetben használható a megoldás. A mainbe egy try-catch blokkot kell használnunk. A try-ba beleírjuk a normál működés kódját, a catch-br pedig a hibakezelés kódot. A throw kulcsszóval kivételt dobunk, amit a catch elkap. A ValidateAndPrepare függvény pData paramétere NULL, count paramétere pedig kisebb 0-nál. A save függvényben a fájl megnyitása nem sikerül, és az fopen NULL pointer tér vissza.

Az try-catch blokkok egymásba is ágyazhatók. Így lehetőség van arra, hogy alacsonyabb szinten is kezeljük a kivételt magasabb szint mellett.

Az elkapott kivételt a throw kulcsszó paraméter nélküli alkalmazásával is ujradozhat. A hívási verem visszacserélésének azt nevezzük, amikor egy kivétel dobásakor, annak elkapásáig a függvény hívási láncban lefelé haladva az egyes függvények lokális változói felszabadulnak. A kivétel dobása és elkapása közt kód futhat le.

III. rész

Második felvonás

DRAFT

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

DRAFT

11.3. Általános

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

11.4. C

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

11.5. C++

[BMECPP] Benedek, Zoltán & Levendovszky, 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.