

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, Bátfai, Margaréta, Ács Aranyi, Ádám	2020. március 31.	

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?	6
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	9
2.6. Helló, Google!	9
2.7. A Monty Hall probléma	9
2.8. 100 éves a Brun tétel	10
3. Helló, Chomsky!	11
3.1. Decimálisból unárisba átváltó Turing gép	11
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	12
3.3. Hivatkozási nyelv	12
3.4. Saját lexikális elemző	12
3.5. Leetspeak	13
3.6. A források olvasása	13
3.7. Logikus	15
3.8. Deklaráció	15

4. Helló, Caesar!	18
4.1. double** háromszögmátrix	18
4.2. C EXOR titkosító	19
4.3. Java EXOR titkosító	20
4.4. C EXOR törő	21
4.5. Neurális OR, AND és EXOR kapu	23
4.6. Hiba-visszaterjesztéssel perceptron	23
5. Helló, Mandelbrot!	25
5.1. A Mandelbrot halmaz	25
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	26
5.3. Biomorfok	27
5.4. A Mandelbrot halmaz CUDA megvalósítása	28
5.5. Mandelbrot nagyító és utazó C++ nyelven	29
5.6. Mandelbrot nagyító és utazó Java nyelven	29
6. Helló, Welch!	30
6.1. Első osztályom	30
6.2. LZW	30
6.3. Fabejárás	30
6.4. Tag a gyökér	30
6.5. Mutató a gyökér	31
6.6. Mozgató szemantika	31
7. Helló, Conway!	32
7.1. Hangyaszimulációk	32
7.2. Java életjáték	32
7.3. Qt C++ életjáték	32
7.4. BrainB Benchmark	33
8. Helló, Schwarzenegger!	34
8.1. Szoftmax Py MNIST	34
8.2. Mély MNIST	34
8.3. Minecraft-MALMÖ	34

9. Helló, Chaitin!	35
9.1. Iteratív és rekurzív faktoriális Lisp-ben	35
9.2. Gimp Scheme Script-fu: króm effekt	35
9.3. Gimp Scheme Script-fu: név mandala	35
10. Helló, Gutenberg!	36
10.1. Programozási alapfogalmak	36
10.2. Programozás bevezetés	36
10.3. Programozás	36
III. Második felvonás	37
11. Helló, Arroway!	39
11.1. A BPP algoritmus Java megvalósítása	39
11.2. Java osztályok a Pi-ben	39
IV. Irodalomjegyzék	40
11.3. Általános	41
11.4. C	41
11.5. C++	41
11.6. Lisp	41

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 alkálni 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 (hogy 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 dlatex 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
dlatex 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 'dlatex' 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**, 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 forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyTuring/ciklus.c

Egy magot 100%-on dolgoztatni egész egyszerű feladat, hiszen ha egy szimpla while ciklust megírunk, az alapvetően így működik. Egy magot 0%-on dolgoztatni sem egy egetrengető kihívás, viszont itt már kell minimálisan gondolkodni. De hamar rájövünk, hogy a sleep(x) parancs kiadásával x másodpercig nem használja a processzort a program. Viszont ha nincs parancs a cikluson belül, a processzor ugyanúgy 100 %-on dolgozik, ami abból adódik, hogy az operációs rendszer elvégzendő feladatnak tekinti és neki adja az összes processzoridőt. Viszont az összes magot 100 %-on dolgoztatni nehezebb feladatnak bizonyult. Az egyik ismerősöm tanácsára az OpenMP-t kezdtem el nézegetni a feladat megoldásához. Pár fórumbejegyzés és egy kis utánajárás után pedig sikerült a feladatot megoldanom.

A programot roppant egyszerű használni. Ha egy magot szeretnénk 100%-ban dolgoztatni, akkor semmit nem kell módosítani, szimplán csak le kell fordítani és futtatni.

Ha egy magot szeretnénk 100%-ban dolgoztatni, akkor vegyük ki a `//`-t a

```
//sleep(1)
```

függvényhívásból.

Ha pedig az összes magot szeretnénk 100%-ban dolgoztatni, akkor ugyanúgy a `//`-t kell kitörölni a következő helyről:

```
#pragma omp parallel while
```

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:

Nem tudunk olyan programot írni, ami minden más programról eldönti, hogy van-e benne végtelen ciklus. Mivel, ha tudnánk, akkor már valószínűleg lett volna olyan ember, aki ezt a programot megírja.

De tegyük fel, hogy megírjuk ezt a programot, aminek a neve legyen eldöntő. Annak a programnak a neve, amelyről el kell dönteni, hogy van-e benne végtelen ciklus, legyen eldöntendő. Nyilván az eldöntő bemeneti argumentuma lesz az eldöntendő. Ahhoz, hogy eldöntő megállapítsa, hogy van-e eldöntendőben végtelen ciklus, futtatnia kell az eldöntendő kérdéses részleteit. Ekkor ha az eldöntendő programban nincs végtelen ciklus, eldöntő hamissal tér vissza, ami azt jelenti, hogy nincs eldöntendőben végtelen ciklus.

Azonban ha az eldöntendő programban tényleg van egy végtelen ciklus, és azt eldöntő futtatja, hogy megbizonyosodjon róla, akkor eldöntő maga is egy végtelen ciklussá válik. Éppen ezért eldöntő sose fog igazsággal visszatérni, mert minden ilyen esetben ő is 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/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyenTuring/felcserelo.c

Annak ellenére, hogy nem használhatunk logikai utasításokat/kifejezéseket, ez egy egyszerű feladatnak bizonyul egy kezdő programozó számára is. Itt most négy módját is bemutatom.

A program bekér két számot és eredményül a kettő értékének a felcseréltjét adja vissza.

```
int main()
{
    int a;
    int b;
    int c;
    printf("Kerem a felcserelni kívánt számokat! \n");
    scanf("%d", a);
    scanf("%d", b);
    printf("A ket szam: \n");
    printf("A = " "%d", a);
    printf("B = " "%d", b);

    //Első változat:
    /*
        c = a;
        a = b;
        b = c;
    */
    //Második változat:
    /*
        a = a-b;

```



```
        b = a+b;
        a = b-a;
    */
    //Harmadik változat:
    /*
        a = a*b;
        b = a/b;
        a = a/b;
    */
    //Negyedik változat:
    /*
        a=a^b;
        b=a^b;
        a=a^b;
    */
    printf("A ket szam felcserelve: \n");
    printf("A = " "%d", a);
    printf("B = " "%d", b);

    return 0;
}
```

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 [Bátfai Norbert](#) tulajdona.

Ez egy egyszerű program, ami a grafikus megjelenítést imitálja. Egy labdának álcázott o betűt mozgat egy while ciklus segítségével. Ez a program egy nagyon jó kezdet a grafikus felületű programokkal való megismerkedéshez.

A program két fő részből áll. Az egyik egy függvény, ami a labdát rajzolja ki a konzolra, A másik pedig maga a main.

A main-ben először létrehozunk egy maxX és egy maxY változót, amiket át is adunk a tx és a ty tömbök méretének.

Ezután két for ciklus végigmegy a két tömbön, a második, és az utolsó elemek értéke -1 lesz, a többi elem pedig 1

végül pedig egy while ciklus és a függvény segítségével kiírja a konzolra a labdát.

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: <https://www.youtube.com/watch?v=xLu3I6lBH-E>

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bogo.cpp

A BogoMips a processzorunk sebességének meghatározásához használatos mértékegység. Azt mondja meg, hogy a számítógép processzora mekkora szóhosszal dolgozik.

Ezt a $XOR \wedge$ művelet segítségével számolja ki a program, ami a kizáró vagy művelete. Az int értékének 1-et adunk, és addig shifteljük balra, ameddig lehet, vagyis amíg az int értéke egyenlő nem lesz nullával.

Közben egy másik változóval számoljuk, hogy hányszor shiftelt balra az int, ezzel meghatározva a szóhosszt. Az én esetemben az eredmény 32 lett, ami a virtuális gép miatt van. Valójában 64 bit-es a processzorom ami 8 bajtnak felel meg.

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/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/pagerank.c

Én a Bevezetés a Programozásba nevű tárgyon már átnézett [PageRank](#) programot vettem alapnak, és azt alakítottam át C++-ból C-re.

2.7. 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

A megoldás Bátfai Norbert tulajdona.

A Monty Hall problémát még középiskolában ismertem meg, sok különböző változata van. Az általam ismert történetben Monty Hall egy műsorvezető volt, akinél a nyertes játékosok választhattak három darab ajtó közül. A háromból két ajtó mögött 1-1 darab kecske, míg a harmadik ajtó mögött egy sportautó volt. A játékos választott egy ajtót, majd Monty Hall, aki tudta, hogy melyik ajtó mögött van az autó, kinyitott egy másik ajtót, ami mögött egy kecske lapult. Ezek után a játékosnak lehetősége volt változtatni a döntésén, vagy maradhatott az eredetileg kiválasztott ajtónál. A kérdés az, hogy mely esetben van több esélye megnyerni az autót? A legtöbb ember azt mondaná, hogy 50-50% esélye van megnyerni az autót, hiszen vagy

az egyik ajtó mögött van az autó, vagy a másik mögött. Ekkor persze hiába magyarázzuk, hogy $1/3$ esélye van megnyerni az autót, ha nem vált, és $2/3$ ha vált, a legtöbb embert elég nehéz meggyőzni erről. Ekkor kell kicsit átalakítani a kérdést. Ha van 1 millió ajtó, ebből kiválaszt a játékos 1-et, majd kinyitnak 999,998 ajtót, amik mögött kecske van, akkor melyik esetben van több esélye a játékosnak megnyerni az autót? Ilyenkor már a legtöbb ember egyértelműnek tartja, hogy vált, de van olyan ismerősöm, aki még ekkor is azt mondta, hogy 50-50% esélye van megnyerni az autót, ha vált ha nem. Ez a program ennek a játéknak a nyerési eseteit szimulálja. Tízmillió esetből hányszor nyer az, aki mindig vált, és az aki egyáltalán nem vált.

2.8. 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

A megoldás Bátfai Norbert tulajdona.

Mint tudjuk, léteznek a prímszámok. Ezek olyan számok, amik csak 1-el és önmagukkal oszthatóak. Valamint léteznek az ikerprímek. Ezek pedig olyan prímszámpárok, amiknek a különbsége pontosan 2. Ha minden ikerprím reciprokának az összegének vesszük a sorozatát, akkor ez a sorozat egy számhoz konvergál. Ez a szám a Brun-konstans. Nem tudjuk azt, hogy az ikerprímek száma véges vagy végtelen e, de ez nem okoz gondot, hiszen elvileg ha végtelen se lépi túl az összegük a Brun-konstanst. Na most be kell vallanom, hogy számtalan olyan ember létezik a földön, aki nálam jobban ért a matematikához. Viszont nekem erről egy elég érdekes dolog jutott eszembe, ami nem más, mint Zeno paradoxona. E szerint x utat teszünk meg, hogy elérjük a célunkat. Ezek alapján megteszünk $1/2x$ utat + $1/4x$ utat + $1/8x$ utat + $1/16x$ utat... Ha ezekből képzünk egy sorozatot, az a sorozat 1-hez fog konvergálni, Éppen ezért soha nem érünk el oda, ahova megyünk. Maga a tétel matematikailag helyes, azonban a való életben tudjuk, hogy ez nem így működik.

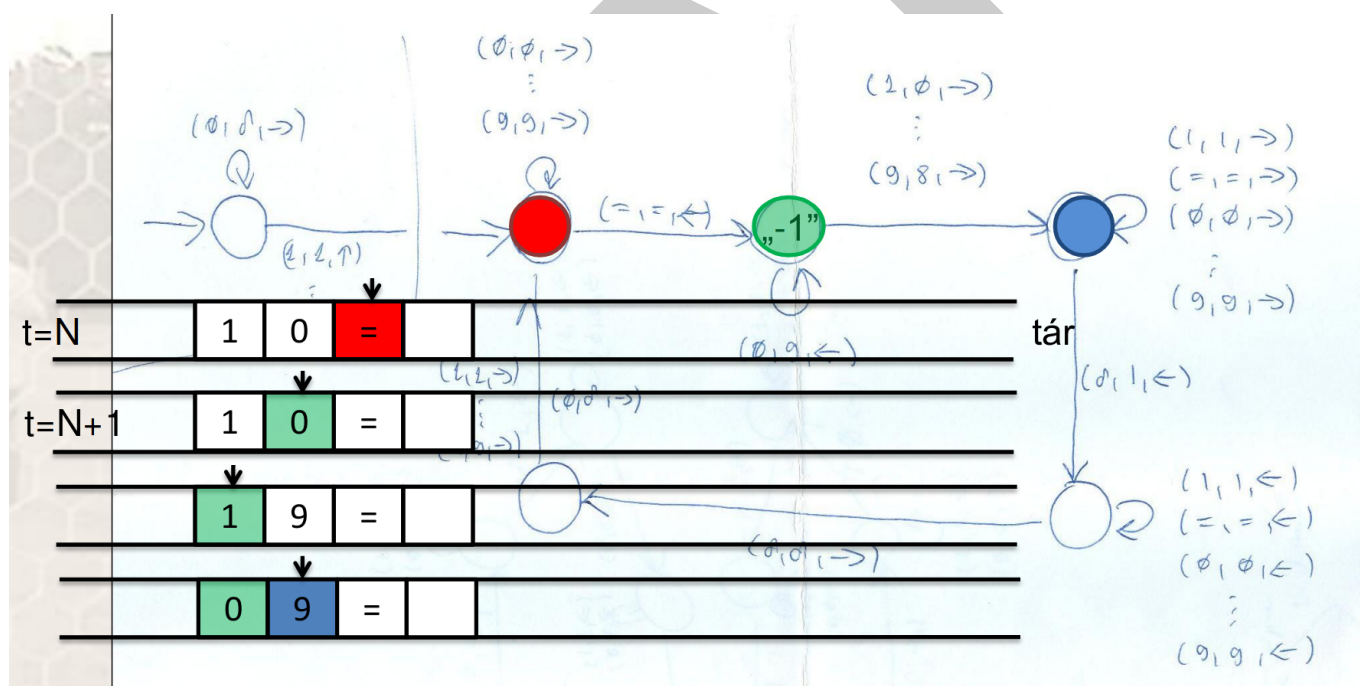
3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:



Megoldás forrása: A kép nem az én tulajdonom, hanem a Magas szintű programozási nyelvek 1 nevű tárgyon kivetített előadás fóliáiról másoltam.

Az unáris számrendszer csupa 1-esekből vagy vonalakból áll. Ilyen például, amikor a kezünkön számolunk, vagy amikor az óvodákban a gyerekek a pálcikákkal rakják ki a dolgokat. Pontosan annyi 1-es vagy pálcika, vagy akármilyen jel kell, amennyi maga a szám értéke. például ha az 50-es számot szeretnénk felírni unárisban, akkor 50 darab 1-est kellene leírni egymás után, ezért ebben a számrendszerben csak a természetes számokat tudjuk ábrázolni.

Egy ilyen decimálisból unárisba átváltó Turing gépet mutat a fenti ábra is. Ez a gép a kapott szám utolsó számjegyéből von le egyeseket. ha a számjegy 0 akkor 9db-ot von le, ha 5 akkor 4-et. Ezzel együtt a levont

egyesekeket a tába helyezi. Ezt minden számjeggyel megismétli.

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:

Az $a^n b^n c^n$ nyelv nem környezetfüggetlen. Na de először értsük meg, hogy mit is jelent ez a nyelv.

Az $a^n b^n c^n$ tulajdonképpen annyit jelent, hogy n darab a , majd n darab b , majd végül n darab c áll egymás után. Ezek a terminális szimbólumok. A szabály alapján a környezetfüggő nyelveknél bal oldalon csak egy önmagában álló nem-terminális szimbólum állhat. Azonban nem létezik olyan képzési szabály ami alapján ez a szabály teljesíthető, éppen ezért ez a nyelv nem környezetfüggetlen.

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: <https://youtu.be/Dkp4gI9JZNs>

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyiChomsky/nyelvek.c

A BNF (Backus-Naur-forma) használatával környezetfüggetlen nyelveket lehet leírni. Nagyon sok programozási nyelvek szintaxisai is BNF-ben vannak leírva.

A programozási nyelveknek is van nyelvtana, illetve nyelvtani szabályaik. Az egyik ilyen szabály C89-ben az, hogy a for ciklus fejrésében nem lehetett változót deklarálni, éppen ezért ha a következőképpen szeretnénk lefordítani a fenti programot:

```
gcc -o nyelv nyelv.c -std=c89
```

Akkor a következő hibaüzenetet kapjuk:

```
nyelv.c:3:2: error: 'for' loop initial declarations are only allowed in C99 or C11 mode
```

Ez pontosan leírja nekünk, hogy a for ciklusban deklarálni csak c99 vagy c11 módban lehet.

Éppen ezért ha `-std=c89` helyett nem írunk semmit, vagy `-std=c99-et` írunk, akkor a program gond nélkül lefordul.

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/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyChomsky/lex.l

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

A lexer-rel szövegelemző programokat lehet generálni az általunk megadott szabályok alapján. A program különböző részeit % jelekkel kell elválasztani egymástól. Itt a numbers változóban fogjuk számolni a valós számok darabszámát. Majd megmondjuk, hogy a digit egy 0 és 9 között lévő számot jelöl. Ezek után jön az a kódrészlet, ami megmondja a lexernek, hogy a valós számokat számolja meg. Végül pedig kiíratjuk a valós számok darabszámát. A futtatáshoz először is telepítenünk kell a lex-et majd a forrásban található programot kell megírni.

Majd azt a következőképp kell lefordítanunk:

```
lex -o lex.c lex.l
```

```
gcc -o lex lex.c -lfl
```

Ezzel megkapjuk a https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyChomsky/lex.c oldalon található programot, ami a feladat megoldása.

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás videó: https://youtu.be/06C_PqDpD_k

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyChomsky/lex.l

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

A leet vagy a saját formájában leírva l337 egy, az internettel együtt elterjedt szleng nyelv, amiben a betűket különböző számokként, és egyéb ASCII karakterekként, a számokat pedig különböző betűkként ábrázoljuk.

Itt is érvényes az a szabály, hogy a program egyes részeit % jelekkel kell elválasztani egymástól. Itt a legelső részben a cipher struktúrában meg vannak adva a karakterek leet formái

A második részben történik az érdemi munka, először a szöveget kisbetűssé alakítja a program, majd pedig végigmegy a szövegen, és minden karaktert a neki megfelelő leet formájú karakterre alakítja át.

A harmadik és egyben utolsó részben található a main amiben a lex meghívása történik.

3.6. A források olvasása

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

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

**Bugok**

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

i.
`if(signal(SIGINT, SIG_IGN) != SIG_IGN)
 signal(SIGINT, jelkezelő);`

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:

1: Ha kapunk egy INTERACT szignált, akkor a jelkezelő függvénnyel eldöntjük, hogy mihez kezdünk azzal a szignállal, mit reagáljon rá a program.

2: Egy for ciklus ami nullától négyig megy, és a ciklus törzsében lévő művelet elvégzése előtt nő az értéke eggyel.

3: Szintén egy for ciklus, ami szintén nullától négyig megy, viszont itt már a ciklus törzsében lévő műveletek elvégzése után növekszik az értéke.

4: Egy for ciklus, ami berakja a tomb[i]-edik helyére az i értékénél eggyel nagyobb értéket, és közben i értékét is növeli.

5: Egy for ciklus, ami addig megy, amíg i kisebb mint n, illetve amíg a d és s pointerek értékei megegyeznek.

6: Kiírunk két, az f nevű függvény által generált számot. Az egyik szám az a majd a eggyel megnövelt értékének a feldolgozásából jön létre, míg a másik szám a+1 és a feldolgozásából. Fontos a sorrend.

7: Szintén két számot írunk ki, az egyik szám az f nevű függvény által feldolgozott a nevű számból előállt érték, a másik pedig a értéke.

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

Első értelmezése: Minden számra igaz, hogy létezik tőle nagyobb y prímszám.

Második értelmezése: Minden számra igaz, hogy létezik egy olyan tőle nagyobb y prímszám, hogy $y+2$ is prímszám.

Harmadik értelmezése: Létezik olyan szám, amitől minden prímszám kisebb.

Negyedik értelmezése: Létezik olyan szám, amitől egyik kisebb szám se prímszám.

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; //létrehoz egy egész típusú változót`
- `int *b = &a; //egy pointer, ami a memóriacímére hivatkozik`
- `int &r = a; //egy referencia a-ra`
- `int c[5]; //5 elemű tömb aminek c a neve`
- `int (&tr)[5] = c; //egy tr nevű referencia c-re`
- `int *d[5]; //egy 5 elemű pointerekből álló tömb`
- `int *h (); //Egy egészre mutató mutatót visszaadó függvény`
- `int *(*l) (); //Egy egészre mutató mutatóra mutató mutatót visszaadó ↔
függvény`
- `int (*v (int c)) (int a, int b) //Függvénytmutató, ami egy egészet ↔
visszaadó függvényre mutató mutatóval visszatérő függvény`
- `int ((*z) (int)) (int, int); //Függvénytmutató, ami egy egészet visszaadó ↔
függvényre mutató mutatót visszaadó függvényre mutat`

Megoldás videó:

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyChomsky/deklaracio.cpp

Egész:

```
int a;
```

Egészre mutató mutató:

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

Egész referenciája:

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

Itt fontos megjegyezni, hogy c-ben nincs referencia, ezért ezt a kódcsipetet érdemes g++-al fordítani gcc helyett.

Egészek tömbje:

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

Egészek tömbjének referenciája (nem az első elemé):

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

Egészre mutató mutatók tömbje:

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

Egészre mutató mutatót visszaadó függvény:

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

Egészre mutató mutatót visszaadó függvényre mutató mutató:

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

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

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

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

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

4. fejezet

Helló, Caesar!

4.1. double** háromszögmátrix

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

Megoldás videó: https://youtu.be/1_EcjAmCTHA

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_Igy_Caesar/tm.c

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

Először is egy alap fogalom. Az alsó háromszög mátrixnak ugyanannyi sora van, mint oszlopa. Ezen kívül még egy nagyon fontos tényezője az is, hogy a főátlója felett csak 0 szerepel.

Általában az ilyen mátrixokat, ha tömbökben tároljuk, akkor nincs értelme a nullákat is tárolni a többi, számunkra érdekes elemmel együtt, éppen ezért ezeket nem is tároljuk. Amikor egy ilyen tömböt vissza szeretnénk alakítani az eredeti alakjára, akkor sorfolytonosan írjuk fel az elemeit. ez mindössze annyit jelent, hogy a mátrix első sorába az első elemet írjuk fel, a második sorába a 2. és 3. elemet, és így tovább minden sorban eggyel több elemet írunk fel mint az előző sorban.

Ebben a programban egy ilyen alsó háromszög mátrixot hozunk létre egy

```
double **
```

segítségével. Ez egy pointerre mutató pointer, ami tökéletes a többdimenziós tömbök használatához.

Ezek után a következő kis programrészlet:

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

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

```

    {
        return -1;
    }
}

```

Ellenőrzi, hogy történt-e valamilyen memóriahiba, (pl. nincs-e tele a memória) és ha történt, akkor -1-el tér vissza.

Ellenkező esetben a program a

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

képletet használva feltölti a tömböt. Ezután két egymásba ágyazott for ciklus segítségével kiírja azt. Ezek után módosítunk a tömb egyes elemein, majd megint kiírjuk őket.

Legvégül pedig a

```

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

```

függvény használatával felszabadítjuk a tömbnek lefoglalt helyet.

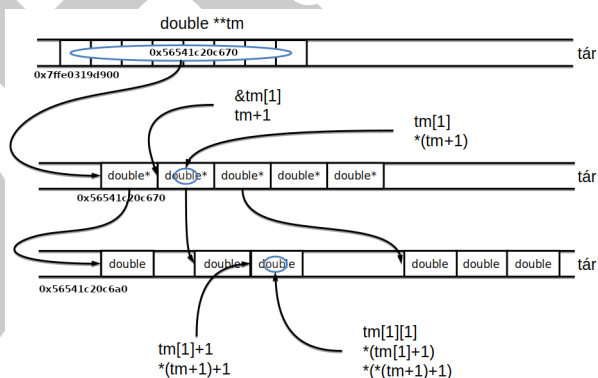
A program futtatásnál a következő memóriacímeket írta ki:

```

./tm
0x7ffe0319d900
0x56541c20c670
0x56541c20c6a0

```

Aminek a jelentése:



A képnek az alapját Bátfai Norbert Biztosította, én azt módosítottam.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó: <https://youtu.be/osQ0iZu4Swg>

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyiCaesar/exor.c

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

Ez a fajta titkosítás a kizáró vagy műveleten alapul. A megadott kulcs, és a forrásfájl karaktereit kizáró vaggal titkosítva egy szöveget úgy tudunk titkosítani, hogy egy olvashatatlan karaktermasszát kapunk végeredményül. Viszont aki ismeri a kulcsot az ugyan olyan egyszerűen vissza tudja alakítani a szöveget az eredeti alakjára úgy, hogy még egyszer lefuttatja a programot, de a titkosított forrást adja meg titkosítandóként, ezzel visszakapva az eredeti szöveget. Így más nem tudja elolvasni a titkainkat, csak az, aki ismeri hozzá a kulcsot. (legalábbis egyelőre. Két feladattal később már más lesz a helyzet.)

Először is a

```
#define MAX_KULCS 100
#define BUFFER_MERET 256
```

használatával megadjuk a maximális kulcs és buffer méretet. a main osztály első argumentuma a kulcs lesz, míg a második az maga a szöveg, amit titkosítani szeretnénk.

A következő ciklusok használatával:

```
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);
}
```

program végigmegy a bemeneti adatok (titkosítandó fájl) karakterein, és mindegyiket titkosítja a kulcs használatával, és kiírja a végeredményt.

A program használata: `./exor kulcs <titkosítandó fájl> titkosított fájl`

Erre egy példa: `./exor 12345678 <lista> titkoslista`

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó: <https://youtu.be/P7PbIIRZiSA>

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyiCaesar/exor.java

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

Itt az előző feladatban megírt EXOR titkosítót írjuk át java programozási nyelvre. Ehhez importálnunk kell az input/output streamet, ez ahhoz kell, hogy olvasni tudjuk a bemeneti fájlt, illetve, hogy írni tudjuk a kimeneti fájlt.

A main-ben megpróbáljuk a try-al beolvasni az args (argumentumok) tömbbe azt a fájlt, amit titkosítani szeretnénk, és ha ez nem sikerült, akkor "elkapjuk" a hibát a catch szerkezettel, és kiíratjuk, hogy mi a hiba:

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

        try {

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

        } catch (java.io.IOException e) {

            e.printStackTrace();

        }

    }
}
```

Ha viszont sikerült beolvasni a fájlt, akkor az ExorTitkosító nevű függvényt meghívva előállítjuk a titkosított szöveget. a System.in illetve System.out a bemenő és a kimenő fájlra utalnak.

Először is a függvény átadja a program a kulcs nevű tömbnek a bemenő szöveget, és létrehoz egy buffer nevű tömböt is 256-os mérettel. Erre az EXOR művelethez lesz szükség.

Végül a program egy while-ba épített for ciklus segítségével végigmegy a szövegen, és minden egyes karakternek meghatározza a titkosított verzióját, és kiírja azt a kimeneti fájlba.

4.4. C EXOR törő

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

Megoldás videó: <https://youtu.be/oii26G1ubGk>

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgylCaesar/tores.c

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

Önnel is előfordult már az, hogy elfelejtette egy EXOR-ral titkosított fájl kulcsát? Ön is akart már kutakodni mások fájljai között, de nem tudott, mert EXOR-ra voltak titkosítva a fájlok?

Ne szenvedjen tovább. Az EXOR törő biztos megoldást nyújt önnek! Csupán annyit kell tudnia, hogy hány karakterből áll a kulcs, és máris használhatja ezt a fenomenálist programot. A felhasználó ostobaságaiért és azok jogi következményeiért felelősséget nem vállalunk.

A működése roppant egyszerű. Mivel nem ismerjük a kulcsot, ezért a program az összes lehetséges kombinációt végigpróbálja. A következőkben bemutatott példában a kulcs 8 darab karakterből áll.

Legelőször a program a következő while ciklus:

```
while ((olvasott_bajtok =
        read (0, (void *) p,
              (p - titkos + OLVASAS_BUFFER <
               MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS -
              p)))
    p += olvasott_bajtok;
```

Használatával beolvassa a feltörni kívánt fájlt, majd a maradék helyet a bufferben egy for ciklust használva

```
for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
titkos[p - titkos + i] = '\\0';
```

feltölti 0 értékekkel.

Ezek után egy halom (ami jelen esetben 8) for ciklussal:

```
#pragma omp parallel for private(kulcs)
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;
    exor_tores (kulcs, KULCS_MERET, titkos, ←
                p - titkos);
}
```

Megpróbálja a program előállítani az eredeti szöveget. Azonban több kombináció is ad eredményt, éppen ezért nekünk kell kitalálni, hogy a kapott eredmények közül melyik a helyes. Kis érdekesség, hogy ezek a for ciklusok az összes magot dolgoztatni fogják, ezzel jelentősen lecsökkentve a töréshez szükséges időt.

Ha a kulcs nem 8 karakterből áll, akkor se essünk pánikba! Csupán néhány (pontosan 3) szekcióban kell módosítani a program kódját. Ezek a következők:

Először is a program fejében a

```
[#define KULCS_MERET 8
```

sorban a 8-at át kell írni arra a számra, amennyi karakterből áll a kulcs.

Majd a 70. és 71. sorokban lévő

```
[ printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],kulcs[6],kulcs[7], ←
    buffer);
```

utasításokban annyi **%c** és **kulcs[n]** legyen, amennyi karakterből áll a kulcs.

Végül pedig az előzőekben már látott for ciklus halmon kell módosítanunk úgy, hogy pontosan annyi **for** ciklus, és pontosan annyi **kulcs[n] = xi**; legyen a programban, amennyi karakterből áll a kulcs.

Most hogy ezt mind tudjuk, a programot a következőképpen kell fordítani: **gcc tores.c -fopenmp -o tores -std=c99**

És futtatni: **./tores <titkosfájl**

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

A megoldás forrása Bátfai Norbert tulajdona.

A neurális hálózatokban például a machine learning esetében, a neuronok egy gráfban elhelyezkedve egymással kommunikálnak úgynevezett "Activation function", magyarul Aktivációs függvény segítségével.

Léteznek bemeneti, kimeneti, és rejtett neuronok is.

A bemeneti neuronok kapják meg a bemenetet. Itt több különböző fajta neuront is meg lehet különböztetni. Vannak egybemenetű és több bemenetű neuronok is. Ezeknek a neuronoknak nincs különösebb feldolgozó feladatuk, továbbítják a bemenetet a többi neuronnak.

A kimeneti neuronok amik a környezetnek adják tovább a kapott információt.

A rejtett neuronoknak pedig a bemenete és a kimenete is csakis más neuronokhoz kapcsolódik.

Ezek alapján egy neurális hálónak legalább két rétegből kell állnia. Egy bemenetiből, és egy kimenetiből. Felső határ, azaz hogy a bemeneti és a kimeneti neuronok között hány darab további réteg helyezkedik el, elviekben nincs.

Először minden neuron megkapja a saját bemeneteit, és minden neuron ebből a bemenetből előállít egy úgynevezett súlyozott összeget, és ezt az értéket vezeti végig az aktivációs függvényen. Egy példa lehet az, hogy ha a súlyozott összeg pozitív lesz, akkor az érték 1, míg ha a súlyozott összeg negatív, akkor az érték -1 lesz.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

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

A megoldás forrása Bátfai Norbert tulajdona.

Perceptronról a Mesterséges Intelligenciák, és a neurális hálók témakörében lehet szó. Ellenőrzi a bemenetet, és egy feltétel alapján eldönti, hogy mi legyen a kimenet, Egy példa:

Van három bemeneti adatunk amikhez pozitív egész számokat várunk. Ha a három bemeneti számból kettő kisebb mint nulla, akkor a kimeneti adat -1 lesz, ha viszont a háromból legalább kettő pozitív szám, akkor a számok összege lesz a kimeneti adat.

Ekkor kimondhatjuk, hogy 1 a hibahatár, mert ekkor még megkapjuk az általunk kért dolgot, viszont ha már kettőt hibáztunk akkor már -1 lesz a válasz.

Ezt a hibahatárt szokták finomhangolni. Nagyon magas hibahatárnál kezdenek, és egyre kisebbé teszik egészen addig amíg elfogadható a hibák mennyisége.

Persze a mi három bemeneti adatos példánknál nem sokat lehet finomhangolni, de ha több millió bemeneti adatról beszélünk, ott ez egy elég fontos dolog.

DRAFT

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó: <https://www.youtube.com/watch?v=hHKhGdCN2R0>

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_Igy/Mandelbrot/mandelbrot.cpp

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

Mielőtt bármihez hozzáfazdenénk egy nagyon fontos információ. Ahhoz, hogy leforduljon a programunk, szükséges a png++. Ezt a legegyszerűbben a **sudo apt install png++** paranccsal lehet megtenni. Most hogy ezt letudtuk, jöhet pár alapvető információ.

A Mandelbrot halmaz lényege (legalábbis számomra) az, hogy komplex számokkal, és egy egyenlettel dolgozik. Azok a számok amelyek kielégítik ezt az egyenletet egy nagyon szép képet alkotnak, ha levetítjük őket egy kétdimenziós síkra. Akit ez bővebben vagy részletesebben érdekel azoknak ajánlom a különböző weboldalakat, én nem fogom tovább boncolgatni, mert én magam sem értem.

A program legelején includeoljuk a png++-t, hiszen nagyrészt ezt fogja használni a program.

```
#include <png++-0.2.9/png.hpp>
```

Ezek után létrehozunk végleges értékeket N-nek és M-nek, valamint megadjuk X és Y lehetséges minimum és maximum értékét is.

```
#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35
```

két egymásba ágyazott for ciklus használatával megadjuk a C, a Z, és a Zuj nevű Komplex számok valós és imaginárius értékét. Ezek korábban lettek létrehozva a mainen belül, és a Komplex nevű struktúrához tartoznak.

```
struct Komplex
{
    double re, im;
};
```

```
struct Komplex C, Z, Zuj;
```

Végül pedig a `GeneratePNG(tomb)` nevű függvény használatával a program legenerálja a PNG fájlt. pixelről pixelre.

A programot a következőképpen tudjuk fordítani: **`g++ mandelbrot.cpp -lpng16 -o mandelbrot`** Futtatni pedig a szokásos módon **`./mandelbrot`** parancsal tudjuk.

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

Megoldás videó:

Megoldás forrása: https://github.com/Viktorpalfy/bhax/blob/master/bhax/thematic_tutorials/bhax_textbook_IgyN/Mandelbrot/mandelbrotkomplex.cpp

A megoldás forrása nem az én tulajdonom. Az eredeti forrás megtalálható az [UDPROG](#) repóban.

Ebben a feladatban a végeredmény ugyan az kellene hogy legyen, mint az előző feladatban. Illetve azóta még a mandelbrot halmaz lényege sem változott, ezért azt nem írnám le újra.

A `png++` ebben az esetben is kelleni fog, így ha nincs leszedve, akkor pillants az előző feladat magyarázatára, ahol megtalálod a szükséges dolgokat ahhoz, hogy le tudjon fordulni a program.

Ebben az esetben az `std::complex` osztályt fogjuk használni a program megvalósításához. Ez az osztály, ahogy a neve is utal rá, a komplex számok kezelése miatt jött létre.

A program által használt függvényei a következők:

A `real(C)` a komplex szám valós részét határozza meg.

A `imag(C)` a komplex szám képzetes részét határozza meg.

Legelőször a program

```
#include <png++-0.2.9/png.hpp>
#include <complex>
```

beincludeolja a `png++`-t és a `complex` osztályt

Ezek után az előző feladathoz hasonlóan itt is megadjuk a végleges értékeket az `N`, `M` valamint `X` és `Y` maximum és minimum értékeinek.

Legnagyobb részben ennek a feladatnak a megoldása megegyezik az előző feladat megoldásával, ezért azt nem írnám le újra, inkább arra koncentrálnék, hogy miben más ez a forrás mint az előző.

Az érdemi különbség a két forrás között az az, hogy itt az `std::complex` osztályt használva, már nem kell létrehoznunk egy saját struktúrát a komplex számoknak.

E helyett szimplán létrehozzuk a `double` típusú komplex számokat a következőképpen:

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

Illetve a for cikluson belül sem a struktúrán belüli elemek imaginárius és valós részére hivatkozunk, hanem a `real()` és `imag()` nevű függvényeket meghívva mondjuk meg a komplex szám részeinek értékét.

```
real(C) = MINX + j * dx;  
imag(C) = MAXY - i * dy;
```

A programot fordítani és futtatni ugyan úgy kell, mint az előző feladatot.

5.3. Biomorfok

Megoldás videó: <https://www.youtube.com/watch?v=nUAG7xP54AQ>

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

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

A két előző feladathoz hasonlóan itt is szükségünk van a `png++` ra, ezért ha még nem szedted le akkor pillants rá a az első feladat magyarázatára, ahol részletesen le vannak írva az ehez szükséges parancsok.

Ez egy olyan mandelbrot program, ahol maga a user adja meg a határokat. Előnye hogy az eredetihez képest teljesen más képeket kapunk, hátránya viszont hogy ha a user nem tudja, hogy mit csinál akkor az egész kép egy nagy fekete semmi lesz.

Először is

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

includeoljuk az `iostream`et a `png++`t és a `complex` osztályt.

A main argumentumai a bemeneti adatok, amikből előállítjuk magát a képet.

Ellenőrzi a program, hogy megfelelő mennyiségű bemeneti értéket adott e meg a felhasználó, és ha nem, akkor felvilágosítja, hogy hogyan kell használni a programot.

```
if ( argc == 12 )  
{  
    szelesseg = atoi ( argv[2] );  
    magassag = atoi ( argv[3] );  
    iteraciosHatar = atoi ( argv[4] );  
    xmin = atof ( argv[5] );  
    xmax = atof ( argv[6] );  
    ymin = atof ( argv[7] );  
    ymax = atof ( argv[8] );  
    reC = atof ( argv[9] );  
    imC = atof ( argv[10] );  
    R = atof ( argv[11] );  
}  
else
```

```
{  
    std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↵  
        d reC imC R" << std::endl;  
    return -1;  
}
```

Ha viszont megfelelő mennyiségű argumentumot adott meg a felhasználó, akkor létrehozza a képet aminek a szélessége és a magassága a felhasználó által megadott szélesség és magasság lesz.

```
png::image < png::rgb_pixel > kep ( szelesseg, magassag );
```

Ezek után a program két egymásba ágyazott for ciklus segítségével kiszámolja, és létrehozza a képet, és el is menti a felhasználó által megadott néven.

A fordítása az előző két programhoz hasonlóan működik, a futtatása azonban már így néz ki:

./3.1.3 fajlnev szelesseg magassag n a b c d reC imC R Erre egy példa:

./3.1.3 biomorf.png 800 800 10 -2 2 -2 2 .285 0 10

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása: https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/CUDA/mandelpngc_60x60_100.

A megoldás forrása Bátfai Norbert tulajdona.

Először is közérdekű közlemény, hogy ennek a programnak a sikeres fordításához szükségünk lesz egy CUDA magokat használó NVIDIA kártyára, illetve az nvidia-cuda-toolkit re amit a következő parancsal tudunk feltenni:

sudo apt install nvidia-cuda-toolkit

Ez a program ugyanúgy a mandelbrot halmazt rajzolja ki, mint az előzőek, azonban itt egy nagyon fontos különbség az, hogy míg az előző feladatoknál a képet a CPU számolta ki és készítette el, addig itt, az NVIDIA kártyák CUDA magjait használjuk a kép kiszámításához.

Ez azért fontos, mert az előző feladatoknál egyetlen egy mag dolgozott és számolt ki mindent, addig itt az én GTX 1660 SUPER videokártyám esetében 1408 darab cuda mag számolná és rajzolná ki a képet. Azért írom feltételes módban, mert a virtuális gép egyik hátránya, hogy nem tudja a gép minden részét kihasználni.

Ez nyilvánvalóan egy sokszor gyorsabb futási időt eredményez. Az én esetemben például amikor CPU-val futtattam volna a programot akkor egy kb 20 másodperces futási idő jött volna ki.

Azonban ha már a fentebb említett GTX 1660 SUPER kártyát használva futtatom a programot, akkor már egy kicsit hamarabb lefut a program.

A programot a gcc helyett az nvcc nevű paranccsal kell fordítani. Futtatni pedig a szokásos módon.

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 videó:

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás Forrása: https://github.com/Viktorpalfy/bhax/tree/master/bhax/thematic_tutorials/bhax_textbook_IgylMandelbrot/MandelJava

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

A feladat az ezt megelőző feladat átírása Java nyelvre. A GUI megírásához szükség van egy keretrendszerre, ami jelen esetben az Abstract Window Toolkit lesz.

Először nézzük a *Mandelbrothalmaz.java* fájlt.

A main-ben a MandelbrotHalmaz() meghívásával létrehozunk egy új halmazt a megadott paraméterekkel. Ezek a paraméterek a tartományok koordinátái, a halmazt tartalmazó tömb szélessége, és a számítás pontossága.

Utána a felhasználó tevékenységeit figyeli a program, és megfelelően reagál rájuk, valamint a GUI ablak tulajdonságait adja meg, illetve kirajzolja magának a halmaznak a képét.

A következő fájlunk a *MandelbrothalmazNagyító.java*

A nevéből adódóan ez végzi a halmazon a nagyítás folyamatát, illetve magának a halmaznak a kirajzolását is. Maga a MandelbrotHalmazNagyító osztály figyeli a felhasználó egér tevékenységeit, azzal kapcsolatban, hogy hol szeretné nagyítani a képet, illetve kirajzolja az új, nagyított képet. Ezen kívül ez végzi az elmentendő képek készítését, és elmentését is.

Végül pedig a *MandelbrotIterációk.java* fájl szerepe.

Ez a programrészlet a nagyított mandelbrot halmazok pontjait tartja nyilván. Ez egy számításra létrehozott osztály, ami a kiválasztott ponthoz tartó utat mutatja meg.

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

Megoldás videó:

Megoldás forrása:

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

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:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

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:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

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

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

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

[?]

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

10.3. Programozás

[BMECPP]

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

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.