

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. március 8., v. 0.0.9

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 Ács Ratku, Dániel	2019. március 11.	

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	2019-03-01	A Helló, Turing csokor megkezdése és a dokumentum formájának a személyre szabása és a saját repository-ba való feltöltés.	ratku.dani
0.0.6	2019-03-04	A Helló, Turing csokor nyolc feladatából az első öt feladatának befejezése.	ratku.dani

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.7	2019-03-05	A Helló, Turing csokor befejezése és a Helló, Chomsky feladatcsokor átfutása.	ratku.dani
0.0.8	2019-03-08	A Helló, Chomsky feladat csokor megkezdése, és a github frissítése.	ratku.dani
0.0.9	2019-03-08	A Helló, Chomsky csokor befejezése, és a teljes feladatsor feltöltése githubra.	ratku.dani

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!	10
3.1. Decimálisból unárisba átváltó Turing gép	10
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	10
3.3. Hivatkozási nyelv	10
3.4. Saját lexikális elemző	11
3.5. Leetspeak	11
3.6. A források olvasása	11
3.7. Logikus	12
3.8. Deklaráció	13

4. Helló, Caesar!	17
4.1. int *** háromszögmátrix	17
4.2. C EXOR titkosító	17
4.3. Java EXOR titkosító	17
4.4. C EXOR törő	17
4.5. Neurális OR, AND és EXOR kapu	18
4.6. Hiba-visszaterjesztéses perceptron	18
5. Helló, Mandelbrot!	19
5.1. A Mandelbrot halmaz	19
5.2. A Mandelbrot halmaz a std::complex osztállyal	19
5.3. Biomorfok	19
5.4. A Mandelbrot halmaz CUDA megvalósítása	19
5.5. Mandelbrot nagyító és utazó C++ nyelven	19
5.6. Mandelbrot nagyító és utazó Java nyelven	20
6. Helló, Welch!	21
6.1. Első osztályom	21
6.2. LZW	21
6.3. Fabejárás	21
6.4. Tag a gyökér	21
6.5. Mutató a gyökér	22
6.6. Mozgató szemantika	22
7. Helló, Conway!	23
7.1. Hangyaszimulációk	23
7.2. Java életjáték	23
7.3. Qt C++ életjáték	23
7.4. BrainB Benchmark	24
8. Helló, Schwarzenegger!	25
8.1. Szoftmax Py MNIST	25
8.2. Szoftmax R MNIST	25
8.3. Mély MNIST	25
8.4. Deep dream	25
8.5. Robotpszichológia	26

9. Helló, Chaitin!	27
9.1. Iteratív és rekurzív faktoriális Lisp-ben	27
9.2. Weizenbaum Eliza programja	27
9.3. Gimp Scheme Script-fu: króm effekt	27
9.4. Gimp Scheme Script-fu: név mandala	27
9.5. Lambda	28
9.6. Omega	28
 III. Második felvonás	 29
10. Helló, Arroway!	31
10.1. A BPP algoritmus Java megvalósítása	31
10.2. Java osztályok a Pi-ben	31
 IV. Irodalomjegyzék	 32
10.3. Általános	33
10.4. C	33
10.5. C++	33
10.6. Lisp	33

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.xsl
Build the book set list...
Build the listings...
XSLT stylesheets DocBook - LaTeX 2e (0.3.10)
=====
Stripping NS from DocBook 5/NG document.
Processing stripped document.
Image 'dblatex' not found
Build bhax-textbook-fdl.pdf
'bhax-textbook-fdl.pdf' successfully built
```

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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 forrása:

[Egy magot altató program.](#)

[Egy magot 100%-ban kihasználó program..](#)

[Minden magot 100%-ban kihasználó program.](#)

A feladat során, az egy magot 100%-osan kihasználó programban, egy olyan do ciklust használtam, amely sosem éri el az abban megadott értéket, ezáltal a végtelenségig futva terheli a processzort. Az összes magot 100%-ban leterhelő program során, meg kellett hívunk az omp könyvtárat, amellyel a "#pragma omp parallel"-t, amivel elérjük, hogy párhuzamosan terheljük a processzor összes szálát. Emellett egy olyan paraméterek nélküli for ciklust, amellyel megkapjuk a kívánt kihasználtsági szintet. A magot altató programhoz meg kellett hívni az unistd.h nevezetű könyvtárat, amivel tudjuk már használni a nem standard sleep függvényt, ami úgy működik, hogy a bele implantált érték ideéig leállítja a processzor egy szálát. Az egész feladatot végtelen ciklusokkal oldottam meg.

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 vgtelen 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.

(A feladat megoldásában Pankotai Kristóf segített) Egy olyan szoftvert kell megírni, ami el tudja dönteni egy adott különböző programról, hogy az lefagy-e, azaz a végtelenségig tud-e futni. Ezt nem lehet elkészíteni, mivel olyan programot nem lehet írni, ami egy másik program végtelenségét vizsgálva megelőzné azt, és annak a végére érkezve eldönthetné, hogy ez egy befagyhatatlan program-e a végtelensége miatt.

2.3. Változók értékének felcserélése

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

Megoldás forrása:

[Változókat felcserélő program.](#)

(A feladat megoldásában Pankotai Kristóf segített) Ennek a feladatnak a megoldásához be kell kérni két számot, amelyet a megadott számokkal különböző műveleteket elvégezve megkapjuk ezeknek a sorrendben felcserélt változatát. A feladatban három művelet szerepel, amelyben a változókat adjuk és vonjuk ki egymásból úgy, hogy a kapott eredmény a feladatnak megfelelő módon jelenjen meg.

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 videón.)

Megoldás forrása:

[Labdapattogtató feladat megoldása.](#)

[Labdapattogtató feladat if nélkül.](#)

Ennek a feladatnak a megoldásához meg kell hívnunk az unistd.h és a curses.h könyvtárat. A void azért kell bele, mert nem kér be semmit a program. Először is inicializálnunk kell az ablakot, amelynél figyelembe vesszük a felhasználó által kihúzott méretet. Deklaráljuk a szükséges változókat, például a kiinduló értéket és a haladás irányát. Majd egy végtelen ciklusban az ablak méretének a változását is figyelembe véve vetítjük ki a labda pattogását. Ezt folyamatosan frissítjük és a usleep-et használva megadva a sebességét

halad a "labda". A pattogás menetét, azaz irányát if elágazásokkal határozzuk meg, szám szerint négygel, amely tartalmazza a fel, le, jobbra, és a balra való irányokat.

Az if elágazások nélküli feladatnál általunk előre megadott ablak, azaz pályaméret generálunk. Ebben for ciklusokkal haladunk az előbb elmített tömbben, ezzel vizsgáljuk hol van a labda helye. A ciklus futásával halad a labda, ami ha eléri valamelyik tengely falát, akkor az adott iránnyal ellentétes irányba kezdi el a mozgást. Ekkor a folyamat előjele is változik. Ez még formalizálásra kerül.

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 forrása:

[Szóhossz feladat megoldása.](#)

Ebben a feladatban meg kellett vizsgálnunk az int típusnak az értékét a gépünkön. Ebben a programban két változóval kell dolgoznunk. Az egyik int értéke nulla, a másik pedig a 0x01, ami a következő sort jelöli. Ezután egy do while ciklussal növeljük a h értékét folyamatosan, egészen addig bitshift/bitwise-olunk egygel, amíg el nem jutunk az n által jelölt értékig. Ekkor a program megáll, és megkapjuk a bit értékét. Ebben az esetben balra bitwise-olásról van szó.

2.6. Helló, Google!

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

Megoldás forrása:

[PageRank feladat.](#)

Ez az algoritmus a Google nevezetű cég által lett kifejlesztve, melynek célja az oldalak "jó"-ságát mutatja be, ezalatt értve a minőségét, amit az adott oldalra mutató linkek száma határozza meg. Ezáltal a ranglistán egyre előrébb kerülve, a keresőben is elsőbbséget élvezhet az adott weboldal. Az emögött rejlő elméletet értem, a kódját viszont még nem teljesen, ezért leírást róla adni még nem szeretnék, hogy mikor majd megértem, akkor jól struktúrálva, és értelmesen megfogalmazva tudjak róla leírást adni. A későbbiekben teljesen ki lesz dolgozva a feladat.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás forrása:

[Ikerprím feladat.](#)

Maga a tétel az ikerprímekről szól, amelyek definiálva úgy tekinthetők, mint az egymást követő prímek amik között a különbség kettő. A feladat megoldása során deklaráljuk a prímeket egy adott számig, majd a

differentiáljukat nézve, a nagyobból kivonjuk a kisebbet. Egy feltétellel a differenciált keresünk a kettesre. Páronként deklaráljuk a kapott prímeket és a tételnek megfelelően reciprokkal szorozva adjuk össze őket egymással. Legvégül pedig a programon belül egy összesített értéket adunk vissza. Bezárólag pedig, matlab könyvtár által a megadott módon tudjuk ábrázolni a kapott eredményt.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása:

[A Monty Hall probléma megoldása.](#)

Maga a Monty Hall probléma egy paradoxon, amelyben feltétel szerint van egy játékos, egy műsorvezető és a játékosnak három ajtó közül kell választania, amelyekből csak az egyik rejt a nyereményt. Miután a játékos választott egy ajtót, a műsorvezető kinyit egy olyan ajtót ami mögött nincs nyeremény, mivel hogy ő tudja melyik ajtó mit rejt. Ezek után megkérdezi a játékost, hogy szeretne-e a választásán módosítani, és ekkor jön a lényeg. A probléma azt tartalmazza, hogy a változtatással megnő a győzelem esélye $1/3$ -ról $2/3$ -ra. Ez azért van, mert azt feltételezzük, hogy a műsorvezető az általunk nem választott két ajtó közül, azért azt nyitotta ki, amelyiket, mert a másik mögött van a jutalom. A programunk úgy épül fel, hogy először is meghatározzuk a kísérletek számát, ami bármennyi lehet, majd létrehozuk a kísérletek és a játékosok nevű változót, ami tartalmazza a választások számát egytől háromig, és a `replace=T`-vel pedig engedélyezzük, hogy a tippek száma többször is előfordulhasson a kísérletek során. A műsorvezető pedig a kísérletek számával lesz megadva. Egy `for` ciklussal megyünk végig a kísérleteken, ebbe viszont implementálnunk kell egy `if` elágazásokkal vizsgáljuk ki a választásokat. Az első `if`-ben a nyertes lehetőség értéke megegyezik a játékos tippjével, ekkor a műsorvezető a három lehetőségéből kivonja a kísérlet értékét. Más esetben pedig, amikor nem találja el egyből a győztes ajtót, akkor a három lehetőségéből kivonja a tippet és ami mögött a nyeremény van. Végül pedig kiírja az így kapott értéket. Következőnek pedig meghatározzuk, hogy változtat-e a játékos vagy nem. A nem esetén megegyezik a kísérlet a játékos tippjével. Ha pedig változtat, akkor egy `for` ciklussal kiválasztjuk azt, ami nem az eredeti választás, és nem is a műsorvezető által nyitott ajtó. A program pedig a kísérletek számának kiíratásával, a nem változtatással való nyereség és a változtatással való nyereség hányadosát, és a kísérletek számához való összes nyereség számát.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet grájával megadva írd meg ezt a gépet!

Megoldás forrása:

[Unárisba váltó Turing gép](#)

A feladat lényege, hogy egy olyan programot hozzunk létre, ami a tízes számrendszerből, azaz decimálisból váltsunk át egyes számrendszerre, azaz unárisba pozitív egész számokat. Először is meghívjuk az alap könyvtárakat, majd deklarálunk két integer változót. Bekérünk egy egész számot, amit egy for ciklussal alakítunk át. Ami megy sorra, és húzza a vonalakat, majd ha a szám öttel osztható lesz maradék nélkül, akkor egy szóközt helyez le.

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...

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 forrása:

[Nyelvi szabvány](#)

(A feladat megoldásában Pankotai Kristóf.) A program a két nyelvi keret változását mutatja be. A C99 szabvány egyik legjelentősebb újítása az volt, hogy a felhasználó immáron képes a C++ nyelv típusú kommentelését használni.

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 forrása:

[Saját lexikális elemző](#)

Ez a program a lefuttatása során egy másik programot állít elő. A lefuttatás menete: `lex -o program.c program.l`. A C-s programuk fordításánál pedig szükség van a végére illeszteni egy `-lfl` tag-et. A működési elve az, hogy az L nyelv lexelve létrehozza azt a C programot, amely egy komplex karaktersorból kitudja szűrni a valós számokat. A programot a két dupla `%` (százalékjel) szedi három részre. Az első harmada tartalmazza a C programba kerülő részt. A középső része tartalmazza a szabályrendszert, és a C-s ciklust. A végső harmadban pedig a komplett main rész van implementálva.

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás forrása:

[Saját l33t chiper](#)

A programunk az L felépítés során, egy másik C forrást hoz létre, melyet a `-lfl` paranccsal kell lefordítani. A kódsor lényege, hogy a betű- és számkészletünkhöz rendel hasonló karaktereket, melyet a futás során kicserél. Ezt úgy éri el, hogy minden betűhöz és számhoz hozzárendel egy négy tagú mátrixot, amelyben jelöljük az általunk vélt megfelelő hasonmás karaktereket.

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 megyránézésre, 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);

```

i. -> Azt jelenti, ha eddig nem volt figyelmen kívül helyezve a SIGINT jel, akkor a jelkezelő függvény kezelje. Fordított esetben hagyjuk figyelmen kívül.

ii. -> Adott egy for ciklus, amelyben az i értékét növeljük egyesével, amíg az értéke eléri a négyet.

iii. -> Ez megegyezik az előzővel, ugyanis for ciklusban nem lesz más értéke, csak a feldolgozás ideje különbözik a két megadott növelési formánál.

iv. -> Ez egy hibás program, mert egy időben deklaráljuk az i-t és hivatkozunk rá tömbként. A végrehajtás sorrendjében van a hiba, mert a sorrendiség nem helyes, nem ismerjük.

v. ->

vi. ->

vii. ->

viii. ->

A feladat teljes megoldása még kidolgozás alatt, mert nem teljesen értem a további függvényeket.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\text{forall } x \text{ } \text{exists } y \text{ } ((x < y) \wedge (y \text{ } \text{prim})))$
```

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

```
$(\text{exists } y \text{ } \text{forall } x \text{ } (x \text{ } \text{prim})) \supset (x < y) \text{ }$
```

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

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

Matlag

A feladat megértéséhez először is szükségünk van az elsőrendű logika ismeretére, ugyanis az adott példa alapján, ez négy logikai állításból áll. A legfontosabb megérteni az alap kifejezéseket. A \forall jelzi az univerzális kvantort, azaz a bármely-bármelyik, az \exists , a létezik kifejezést jelöli. A \wedge az implikáció és a \cup pedig a konjunkció és a \neg pedig a negált.. A program pedig AR nyelven íródott.

I. \rightarrow Bármely x esetén létezik olyan y , amelynél ha x kisebb, akkor y prím szám lesz.

II. \rightarrow Bármely x esetén létezik olyan y , amelynél ha x kisebb, akkor y prím szám lesz, és ha y prím szám, akkor annak második utána is prím szám lesz.

III. \rightarrow Létezik olyan y , amelynél bármely x esetén az x prím ,és x kisebb, mint y .

IV. \rightarrow Létezik olyan y , amelynél bármely y kisebb x esetén az x nagyobb, és x nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

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

- `int c[5];`
- `int (&tr)[5] = c;`
- `int *d[5];`
- `int *h ();`
- `int *(*l) ();`
- `int (*v (int c)) (int a, int b)`
- `int (*(z) (int)) (int, int);`

Megoldás videó:

Megoldás forrása:

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összehasonlítása azt mutatja meg, hogy miért érdemes a **typedef** használata: [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/Chomsky/fptr.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr.c), [bhax/thematic\\_tutorials/bhax\\_textbook\\_IgyNeveldaProgramozod/Chomsky/fptr2.c](https://github.com/bhax/thematic_tutorials/blob/master/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr2.c).

```
#include <stdio.h>

int
sum (int a, int b)
{
 return a + b;
}

int
mul (int a, int b)
{
 return a * b;
}

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

int
```



```
main ()
{
 int (*f) (int, int);

 f = sum;

 printf ("%d\n", f (2, 3));

 int (*(g) (int)) (int, int);

 g = sumormul;

 f = *g (42);

 printf ("%d\n", f (2, 3));

 return 0;
}
```

```
#include <stdio.h>

typedef int (*F) (int, int);
typedef int (*(G) (int)) (int, int);

int
sum (int a, int b)
{
 return a + b;
}

int
mul (int a, int b)
{
 return a * b;
}

F sumormul (int c)
{
 if (c)
 return mul;
 else
 return sum;
}

int
main ()
{
 F f = sum;
```

```
printf ("%d\n", f (2, 3));

G g = sumormul;

f = *g (42);

printf ("%d\n", f (2, 3));

return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

## 4. fejezet

# Helló, Caesar!

### 4.1. int \*\*\* háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 4.4. C EXOR törő

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

Megoldás videó:

---

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R)

Tanulságok, tapasztalatok, magyarázat...

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

### 5.3. Biomorfok

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/Biomorf](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf)

Tanulságok, tapasztalatok, magyarázat...

### 5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

### 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:

## 5.6. Mandelbrot nagyító és utazó Java nyelven

DRAFT

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

Valósítsd meg C++-ban és Java-ban az módosított polártranszformációs algoritmust! A matek háttér teljesen irreleváns, csak annyiban érdekes, hogy az algoritmus egy számítása során két normálist számol ki, az egyiket 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

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 9. fejezet

# Helló, Chaitin!

### 9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

### 9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

### 9.3. 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](https://youtu.be/OKdAkI_c7Sc)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Chrome](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Chrome)

Tanulságok, tapasztalatok, magyarázat...

### 9.4. 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](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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/GIMP_Lisp/Mandala)

Tanulságok, tapasztalatok, magyarázat...

## 9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9.6. Omega

Megoldás videó:

Megoldás forrása:

DRAFT

## **III. rész**

### **Második felvonás**

**Bátf41 Haxor Stream**

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

---

DRAFT

---



## 10. fejezet

# Helló, Arroway!

### 10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 10.2. 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**

## 10.3. Általános

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

## 10.4. C

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

## 10.5. C++

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

## 10.6. Lisp

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