

Univerzális programozás

Így neveld a programozód!

Ed. BHAX, DEBRECEN,
2019. március 25., v. 0.1.3

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. április 30.	

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	2019-03-11	A Helló, Chomsky csokor befejezése, és a teljes feladatsor feltöltése githubra.	ratku.dani
0.1.0	2019-03-15	A Helló, Caesar feladatok elkezdése.	ratku.dani
0.1.1	2019-03-17	A Helló, Caesar csokor felének a kidolgozása.	ratku.dani
0.1.2	2019-03-18	A Helló, Caesar feladatcsokor majdnem teljes elkészítése.	ratku.dani
0.1.3	2019-03-25	A Helló, Mandelbrot csokor kötelezőinek elkészítése.	ratku.dani
0.1.4	2019-03-25	A Helló, Mandelbrot csokor kötelezőinek elkészítése.	ratku.dani
0.1.5	2019-03-25	A Helló, Mandelbrot csokor kötelezőinek elkészítése.	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	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	8
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	9
2.6. Helló, Google!	9
2.7. 100 éves a Brun tétel	9
2.8. A Monty Hall probléma	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	11
3.3. Hivatkozási nyelv	12
3.4. Saját lexikális elemző	12
3.5. Leetspeak	12
3.6. A források olvasása	12
3.7. Logikus	14
3.8. Deklaráció	14

4. Helló, Caesar!	16
4.1. double ** háromszögmátrix	16
4.2. C EXOR titkosító	16
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 <code>std::complex</code> osztállyal	19
5.3. Biomorfok	20
5.4. A Mandelbrot halmaz CUDA megvalósítása	20
5.5. Mandelbrot nagyító és utazó C++ nyelven	20
5.6. Mandelbrot nagyító és utazó Java nyelven	21
6. Helló, Welch!	22
6.1. Első osztályom	22
6.2. LZW	22
6.3. Fabejárás	23
6.4. Tag a gyökér	23
6.5. Mutató a gyökér	23
6.6. Mozgató szemantika	23
7. Helló, Conway!	25
7.1. Hangyaszimulációk	25
7.2. Java életjáték	25
7.3. Qt C++ életjáték	26
7.4. BrainB Benchmark	26
8. Helló, Schwarzenegger!	27
8.1. Szoftmax Py MNIST	27
8.2. Mély MNIST	27
8.3. Minecraft-MALMÖ	28

9. Helló, Chaitin!	29
9.1. Iteratív és rekurzív faktoriális Lisp-ben	29
9.2. Gimp Scheme Script-fu: króm effekt	30
9.3. Gimp Scheme Script-fu: név mandala	30
10. Helló, Gutenberg!	31
10.1. Programozási alapfogalmak	31
10.2. Programozás bevezetés	32
10.3. Programozás	32
III. Második felvonás	34
11. Helló, Arroway!	36
11.1. A BPP algoritmus Java megvalósítása	36
11.2. Java osztályok a Pi-ben	36
IV. Irodalomjegyzék	37
11.3. Általános	38
11.4. C	38
11.5. C++	38
11.6. Lisp	38

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 (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:

[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 hívnunk 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 engedélyezük, 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 gráfiával megadva írd meg ezt a gépet!

Megoldás forrása:

[Ábra a feladathoz. 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:

(A feladat megoldása során a Dicső mentorom, Pataki Donát volt)

Első: $\tilde{S} (S \rightarrow aXbc) aXbc (Xb \rightarrow bX) abXc (Xc \rightarrow Ybcc) abYbcc (bY \rightarrow Yb) aYbbcc (aY \rightarrow aa) aabbcc$

Második: $S (S \rightarrow aXbc) aXbc (Xb \rightarrow bX) abXc (Xc \rightarrow Ybcc) abYbcc (bY \rightarrow Yb) aYbbcc (aY \rightarrow aaX) aaXbbcc (Xb \rightarrow bX) aabXbcc (Xb \rightarrow bX) aabbXcc (Xc \rightarrow Ybcc) aabbYbcc (bY \rightarrow Yb) aabYbbccc (bY \rightarrow Yb) aaYbbccc (aY \rightarrow aa) aaabbccc$

Az egésznek az alapja a matematikai logika ismerete, amellyel megtudunk kreálni egy olyan nyelvezetet ami megoldásként szolgál és szerepel a feladat megvalósításához.

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 hasonló 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 megváránérésre, 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. -> Itt a ciklusunk, ami a for elmegy addig, amíg az i változó kisebb mint az n változó és érvényes visszatérített értéket nem kap. Ekkor s és d mutató lesz, ahogy a szegmensből adódóan a első adatra mutató mutató. Itt viszont kell a két utóbb említett változó a működéshez, mert önmagában nem elég a leírt részlet.

vi. -> Kivetítjük printf-fel a függvényünket visszajövő értékét, ami az f-ből adódik. A kiértékelés sorrendje nincs megadva szabályszerűen, így nem tudja értelmezni az általunk használt programnyelv. Azaz bugos lesz.

vii. -> Megint csak a függvény visszakapott értékét írjuk ki a printf-fel. Ezt továbbítjuk az "a" nevezetű változónak. Így az "a" az f-fel fog processeket végrehajtani, ami jelen esetben csak egy koppintás, majd ezt írjuk ki.

viii. -> Kiírjuk újból a visszatérített értéket a függvényből, és itt az "f" makró lesz.

A feladat teljes kidolgozása során a kötelező olvasmányok függvényelemzési részletét vettem alapul.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

```
)$
```

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

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

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 exist, a létezik kifejezést jelöli. A wedge az implikáció és a supset pedig a konjunkció és a neg pedig a negált.. A program pedig AR nyelven íródott.

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

II. -> Bármely x esetén létezik olyan y, amelynél ha x kisebb, akkor y prím szám lesz, és ha y prím szám, akkor annak második utánai is prím szám lesz.

III. -> Létezik olyan y, amelynél bármely x esetén az x prím ,és x kisebb, mint y.

IV. -> 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 forrása:

[A feladatok deklarálva](#)

A feladat megoldásában más segített, tutorált, de még így sem értem igazán, szóval majd befejezem. A feladat megoldása során a megint csak a kötelezően meghatározott fejezetek alapján dolgoztam ki, amelyek a deklarációs feladatforrásban találhatók meg.

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 forrása:

[Mátrix megoldása](#)

A programunk egy négyzetes mátrixról szó, amelynek a fő tulajdonsága az, hogy oszlopainak és sorainak száma megegyezik, és a főátlója alatt csupa nullákat kapunk, bár a kódunkban majd az átló alatti rész lesz csupa nulla. Először is deklaráljuk a két fő változónkat, ami a sorok száma lesz, és a tm mutatót, ami double típusú. Az if-ben szereplő részt megírva, a tm változó eredménye a malloc paranccsal megkapott tárterületet, ami a sor, szorozva 8 bájtnyi mérettel rendelkezik. A malloc-ot rá lehet kényszeríteni, hogy bármit visszaadjon, mert alapesetben void*-ot adna vissza. Ezután vizsgáljuk, hogy a malloc tud-e egyáltalán területet foglalni, mert ha nem, akkor a return -1-e visszaugrunk a program elejére. További kidolgozás alatt.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása:

[C Exor feladat.](#)

[C Exor feladat github-on.](#)

Ez a feladat a titkosításról szól, melynek a kulcsa az EXOR, azaz a kizáró vagy logikai elem. A program működéséhez szükségünk van, egy bárki által elkészített szöveget tartalmazó fájlra, mondjuk egy txt-re, amit majd a program átkódol az általunk meghatározott kulccsal, ami egy nyolc karakteres sztringként lesz kezelve, ami azért fontos, mert a vissza kódoláshoz is ekkora hosszúságú kulcs kell, amivel majd a későbbiekben dekódolni is tudjuk. Maga a forrásnak a megírását a kulcs méretének és a buffer méretének meghatározásával kezdjük, amelyek konstansok lesznek. A main-be deklarálnunk kell a kulcs és a buffer tömböket, amik a szükséges kulcsot és az eredményt tárolják. A deklarált két int, a a kulcsunk aktuális

részét/elemét mutatja, amivel végbe megy majd a művelet, és a második pedig, a beolvasott bájtoknak az összegét mondja meg. A harmadik int, azaz a kulcs_meret során használjuk strlen és a strncpy függvényekre, amelyeknek a lényege, hogy lerögzíti a hosszát stringben, amíg a ncpy pedig a végső másolatát rögzíti szintén stringben. Ezután egy végső while és for ciklussal folyamatosan olvasunk a bemenetről, és tároljuk a beolvasottakat a bufferben, amíg már nem tudunk több mindent beolvasni. Ekkor a ciklusban az olvasó 0 értéket ad vissza, amivel a program véget ér. A program megírása után a gcc-vel lefordítom, majd a futtatáshoz a ./programneve kulcs <a szöveges fájl, amiből kódolunk> "a fájl, amibe a titkosított szövegrészlet kerül bele".

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

[Ezen az oldalon található meg a megoldás.](#)

Ebben az esetben ugyan azt akarjuk elérni, mint a C nyelvű titkosító helyzetében. Azaz hogy az általunk, vagy a felhasználó által meghatározott kulccsal egy szintén saját választott szöveges dokumentumot kódolunk le. A szerkezete és működése ezáltal azonos lesz, az előbb említett nyelvben megírt társával. Fordítani a javac programneve.java . Majd a program futtatásához a következő kell: java programneve kulcs > titkosított.szöveg .

4.4. C EXOR törő

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

Megoldás forrása:

[C Exor törés.](#)

[C Exor törés github-on.](#)

Ez a program, az előbb megcsinált kódoló programunk által legenerált szöveges fájlt tudja dekódolni az általunk előzőleg megadott kulccsal, amely nyolc karakterből álló szting. A program alapvető definiálásból indul, ami a maximális szöveg terjedelmet, a bájtokban mért memória tárolást, a kulcs méretét, ami megint csak az említett nyolc karakterből álló sztring lehet, és végül a szabadszoftverű operációs rendszerforrást. Ezek után az első változónk, az átlagos szóhossz, amit a beleimplementált for ciklussal a szóközök segítségével számítunk ki. Ebben az esetben az "sz" változó, a szóközök számát jelenti. A számítás végén a return utasítással úgy adjuk vissza az értéket, hogy elosztjuk a hosszúságot a szóközök megszámlált értékével. A következő szegmens a szöveg tisztaságának vizsgálata, amivel csökkentjük a törések potenciálját. A középső komponens maga az exor része lesz, amiben a for ciklussal bájtanként hajtjuk végre a műveletet. A benne levő "%" jel által lesz a kulcs mindig aktuális. Az elkövetkezendő részlet pedig a törés végrehajtása lesz. Ehhez be kell olvasni a titkos fájlt, amit egy while ciklussal érünk el addig, amíg csak van adat, ha már nincs, akkor a read 0 értékkel tér vissza, és leáll az utasítás. A rengeteg egybeágyazott for ciklussal elérjük, hogy az összes lehetséges kulcs álljon elő. Végül, ha sikerül az exortörés, kiírjuk a kulccsal a tiszta szöveget. A lefordítása alapesetű, és a futtatása megegyezik a titkosító programéval.

4.5. Neurális OR, AND és EXOR kapu

R

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

Neurális

Neurális háló ahol and, or és exor kapuk vannak implementálva. Az and és or működése magától értetődő. Az elobbinél ha mindkettő 1 akkor lesz az eredményük 1. Az utóbbinál pedig ha az egyik 1 akkor lesz az eredményük 1. Az exor pedig nem jelent mást minthogy a két érték nem egyezik meg, tehát 0 és 1 esetén lesz az eredmény 1 és ez fordított esetben is igaz. A neurális hálóról még érdemes annyit megjegyezni, hogy 3 részre lehet osztani. Van az input layer, ahol megkapja az adatokat. Ezt követi egy vagy több hidden layer, ahol a varázslat történik. Majd végül jön az output layer, ahol ha minden jól működik a megfelelő értéket kapjuk vissza.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása:

Visszaterjesztés.

(A feladat megoldásában Dicső tutorom, Pataki Donát segített)

A perceptron nem más mint egy algoritmus model, ami az emberi agy működését próbálja utánozni. Hasonló a neurális háléhoz azonban van pár különbség. Ugyanúgy input után elkezd varázsolni és jobb esetben és megfelelő mintavétel után helyes eredményt ad vissza. Azonban a közbelső értékeknek van súlya amit még adott konstansokkal is ki lehet egészíteni. Az így kapott súlyokat összeadja és ha ez eléri egy bizonyos szintet, akkor a program adott része aktiválódik. És egy a lineáris folyamat ismétlődik amíg el nem jut a válaszig.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás forrása:

[Mandelbrot halmaz kiszámolása.](#)

Mielőtt a feladatban szereplő halmazzal dolgoznánk, tisztáznunk kell, hogy mi az a fraktál, és milyen kapcsolatban állnak a Mandelbrot-halmazzal. A fraktálok lényegében olyan alakzatok, melyek végtelenül komplexek. Két fő tulajdonságuk van, az egyik, hogy a legtöbb geometria alakzattal ellentétben a fraktálok szélei "szakadozottak", nem egyenletesek. A másik tulajdonságuk pedig, hogy nagyon hasonlítanak egymásra. Ha egy kör határfelületét folyamatosan nagyítjuk, egy idő után kisimul(a csúcsokat leszámítva), megkülönböztethetetlené válik egy egyenestől. Ezzel szemben a fraktálok első tulajdonsága, mi szerint határfelületük szakadozott, megmarad, függetlenül a nagyítás mértékétől. A Mandelbrot halmaz is a fraktálok közé tartozik. Ezt és a hozzá tartozó szabályt Benoit Mandelbrot fedezte fel 1979-ben. A halmaz komplex számokból áll, és az ezekből álló sorozat konvergens, azaz korlátos. Ezeket a számokat ábrázolva a komplex számsíkon kapjuk meg a Mandelbrot-halmaz híres farktálját.

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

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás forrása:

[Mandelbrot halmaz `std::complex` osztállyal.](#)

Az előző feladatot fogjuk megoldani, csak most egy kicsit másképpen. Ahogy láttad, az előbb a komplex számokat két változóban tároltuk, egyikben a valós, másikon pedig a képzetes részét. De az informatikusok lusták, mindek használnánk 2 változót, ha lehet egyet is. Ezt teszi számunkra lehetővé a `complex` library, melynek segítségével a gép képes kezelni ezeket a számokat. Na itt néhány dolog eltér az előző feladathoz képest. Itt használjuk elsőnek a `complex` típust, ami `double`-ket tartalmaz, és két részből áll, a valós és az imaginárius részből. Ennek a segítségével definiáljuk a `c` és a `z_n` változókat. Majd, innen már ismerős lehet, kiszámoljuk minden `c` esetén a `z_n`-eket, és ha elérjük az iterációs határt akkor, tudhatjuk, hogy az iteráció konvergens. Ebből következik, hogy a `c` eleme a Mandelbrot halmaznak. A `while`

fejlesztésében látható `abs()` függvény az abszolút értékét adja meg az bemenetként kapott argumentumának. A halmazt létrehozó sorozat képzési szabálya egy az egyben beírható a programba, nincs szükség semmilyen szétbontásra, mint az előző programnál volt, köszönhetően annak, hogy képesek vagyunk kezelni a komplex számokat. Pusz dolog, hogy a a program a futása azt is látjuk, hogy hány százalékát végezte el a számításoknak a gép. Végezetül pedig itt is kiírjuk a png fájl a parancssori argumentumként megadott fájlba a `write` segítségével.

5.3. Biomorfok

Megoldás forrása:

[Biomorfok](#)

A biomorfokra Clifford Pickover talált rá, aki a természet egyik törvényének a feldolgozásának hitte ezt a jelenséget. Ezt Julia halmaznak is nevezzük. A két halmaz között a különbség, hogy a Mandelbrot során a c egy változó, amíg a biomorfok esetén egy állandót alkot majd, azaz a z vizsgálatai során, ugyan az marad.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása:

[CUDA megvalósítás.](#)

Szokásos mandelbrot halmaz azzal a különbséggel, hogy most az nvidia által fejlesztett cuda magokat használjuk. Ezek segítségével a feladatokat lehet párhuzamosítani, viszont cuda csak az nvidia grafikus kártyákban található. Használható c, c++ és fortran nyelveken. Leegyszerűsítve a cuda magok olyanok mint sok kis teljesítményű processzor. Valamint azóta nvidia bejelentette a cuda-x (és talán elérhetővé is tette) könyvtárakat, aminek segítségével különböző területeken lehet hasznosítani a cuda nyújtotta előnyöket.

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:

[Mandelbrot nagyító és utazó C++ nyelven.](#)

Telepíteni: `sudo apt-get install libqt4-dev`

A program a QT GUI-t használja, ennek segítségével tudjuk elkészíteni a Mandelbrot halmazt beutazó programunkat. Ez a GUI az egyik legterjedtebb grafikus interfésze a C++-nak, rengeteg tutorial van róla fent a neten.

Fordítás: Az szükséges 4 fájlnek egy mappában kell lennie. A mappában futtatni kell a `qmake -project` parancsot. Ez létre fog hozni egy `*.pro` fájlt. Ebbe a fájlba be kell írni a következőt: `QT += widgets` sort. Ezután futtatni kell a `qmake *.pro`. Ezután lesz a mappában egy `Makefile`, ezt kell majd használni. Ki adjuk a `make` parancsot, mely létrehoz egy bináris fájlt. Ezt pedig a szokásos módon futtatjuk.

Rengeteg figyelmeztetést ad vissza, de ezzel most nem kell törődni, hiszen a bináris fájl elkészült, melyet futtatunk, és elindul az utazásunk a végtelenbe. Ahhoz, hogy részletesebb képet kapj a ránagyított területről, az "n" billentyűt kell lenyomnod, mely kiszámolja a z-ket a megadott területen. Itt lehet látni, hogyan mosódik össze a Mandelbrot és a Biomorfos téma. A hatmadik kép, már majdnem olyan, mint egy biomorf.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása:

[Mandelbrot Java-ban.](#)

Ugyan az mint a C-s verziója csak a feladatban említett nyelvben.

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 forrása:

[Első osztály java.](#)

[Első osztály C++.](#)

A feladat megvalósítása során megalkotott programunk feladata az, hogy a polár transzformációs algorit-mussal dolgozik. A program tíz alkalommal számol, és ez alapján, hogy tárol-e tag visszatérítést, akkor azt is figyelembe veszi.

6.2. LZW

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

Megoldás forrása:

[Fa építés.](#)

A binfa egy olyan adatszerkezet, aminek csomópontjainak legfeljebb két gyereke lehet. Ezeket jobb és bal gyermekként szokás hívni. Valamint a gyerekek is csomópontként működnek. Azaz a gyerekeknek is lehet-nek további gyerekei, így elég komplex fákat lehet kapni végeredményül. Ezért a binfákat érdemes rekurzív módon bejárni. Továbbá attól függően, hogy milyen a gyerekek eloszlása lehet egy binfa kiegyensúlyozott, főkéletesen kiegyensúlyozott stb. Ezek a tulajdonságok többnyire csak a jobb és bal oldal viszonyát nézik egymáshoz. A program nem csinál mást mint a megfelelő bemenet megkapása után egy fájlba felépít egy binfát.

6.3. Fabejárás

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

Megoldás forrása:

[Fabejárás.](#)

A binfáknak három féle bejárasi módja van. Ezek az inorder, preorder és postorder nevet viselik. A preorder fabejárás során a gyökérből indulunk ki és előbb azok bal majd jobb oldali gyerekeit iratjuk ki. Inorder és postorder során az utolsó bal oldali gyerekből indulunk ki és ha inorder kiíratás szeretnénk akkor azután a gyökeret, majd a jobb oldali gyereket írjuk ki. Ha pedig ezt a sorrendet felcseréljük, azaz a bal gyerek után a jobbat majd a gyökeret írjuk ki, akkor postorder bejárást kapunk. Ezeket a bejárásokat célszerű rekurzívan kezelni, mivel ezek a bejárások ugyanazt csinálják és így elég egyszer megírni és csak újra meghívni a binfa egy részfájára.

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 forrása:

[Tag a gyökér.](#)

Binfaépítés azzal a különbséggel, hogy a csomópont be van ágyazva a gyökérbe és ezért közvetlenül el lehet érni, mivel a csomópont a gyökér része, de lényegében teljesen ugyanaz. Ugyanaz az az alapelve, szóval ezek alapján a kifejtés a fentebb említett feladat kidolgozásában szerepel.

6.5. Mutató a gyökér

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

Megoldás forrása:

[Mutató.](#)

A node és a tree osztályok teljesen el vannak választva egymástól, így a fa működését úgy tudjuk megvalósítani, hogy a gyökér tagod mutatóként továbbadjuk a csomópontnak és így építjük fel a fát minden egyes új csomóponttal, aminek az a nehézsége vagy különbsége, hogy ha a gyökeret akarjuk elérni, azt csak a mutatón keresztül tudjuk megtenni, de lényegében ugyanaz csak meg van bonyolítva.

6.6. Mozgató szemantika

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

Megoldás forrása:

Mozgató szemantika.

A mozgató konstruktor lehetővé teszi, hogy egy object értékeit egy másikba helyezük át. Azaz jelen esetben ha binfás példára nézzük, akkor a mozgató konstruktor segítségével lehetséges, hogy egy már kész binfát egy másikba tegyünk, azonban így az amiből mozgattuk az értékeket üresen marad, mivel nem történik más mint, hogy az értékeket más helyre pakoljuk. Előnye a másolással szemben azonban az, hogy a mozgatás kevesebb erőforrást vesz igénybe mint egy teljes másolatot készíteni az egészből. A c++11 óta van állapotértelmezett mozgató konstruktor is, ami hasonlóan a másolóhoz nem mindig működik tökéletesen, ezért lehetőség van annak működését deklarálni is. A mozgató értékadás hasonló a mozgató konstruktorhoz, ázzal a különbséggel, hogy az utóbbi akkor kerül meghívásra, ha deklarációkor adjuk meg az értéket, míg a mozgató értékadás akkor, ha már egy object deklarálva lett és annak értékét később akarjuk egy másikból áthelyezni.

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 forrása:

[Myrmecologist.](#)

A program lényegében a hangyák feromonokkal történő kommunikációját szimulálja. A `.main` futtatása ezzel a szegmenssel `./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s 3 -c 22` -el lehet. A Kolónia (`AntThread`) egy `QThread` osztályból származtaott osztály, mivel a számításokat a main threadtől, ami a GUI-t kezeli, el akarjuk különíteni, így nem fagy le az Űrlapunk. Ez a Kolónia végzi a hangyák mozgását, alkalmaztatja a környezetre a hangyák által kibocsátott vegyületeket.. Ezekenfelül updateeli a "világot" is, lásd `grid`, csökkenti a világ feromonszintjét. Emiatt nem mondható tiszta OOP programnak a program például. Az `AntWin` nyilván a világot kezeli, olyan dolgokat ad hozzá a funkcionalitáshoz, mint például a világra rács rajzolása, egyes zónák a világon (cell) berajzolása és ezeken felül maga világi tartalom megjelenítése. Kezeli a GUI eseményeket is amiket lekövetünk.

7.2. Java életjáték

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

Megoldás forrása:

[Java életjáték.](#)

A Conway-féle sejtautomatát kellett a feladatban elkészíteni. A program egy 2 dimenziós koordináta-rendszerben dolgozik, amelyben találhatók az ún. sejtek, amelyek vagy "élő" vagy "halott", azaz vagy 0 vagy 1 sejtek. Egy véletlenszerű vagy előre meghatározott állapot indítja el az életjátékot, majd több iteráció fut le (ahol minden iteráció meghatároz egy sejtet). Habár egyszerű szabályokkal és akár véletlenszerűekkel is le lehet futtatni, mindig egyedi formák és viselkedéstípusok emelkednek ki a rácsból.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása:

[C++ életjáték.](#)

Ugyan az a feladat mint az előzőben, csak most Qt-s környezetben C++ nyelven. Működési elve és felépítési elve egy és ugyan az, csak a nyelvi különbségek által található benne más jellegű szegmens.

7.4. BrainB Benchmark

Megoldás forrása:

[BrainB Benchmark.](#)

A BrainB Benchmark feladata az esport tehetségek felkutatása lenne, úgy, hogy feltérképezi az agy kognitív képességeit, és az elért pontszámok alapján össze lehet hasonlítani az egyes egyéneket. Maga a benchmark a "karakter elvesztést" teszteli, vagyis ha a játékban elveszítjük a karakterünket, mennyi ideig tart megtalálunk, és ha megtaláltuk, mennyi ideig tart elveszítenünk. Ideális esetben rövidebb ideig tart megtalálunk, mint elveszteni. A program azt is figyeli, hogy az egyes karakter elvesztésekhez milyen bit/sec képernyő-váltások tartoznak.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

MNIST

TensorFlow

Progpater-es link

Az MNIST egy olyan adatbázis, ahol általunk választott, azaz megadott és kézzel beírt számokat tárol. Ezeket az adatokat a legtöbb esetben képfeldogozó és képeket felismerő programok szokták használni, alkalmazni. A program a feladat leírása alapján, Python magasszintű programozási nyelven íródott, amelyhez a képfelismerés miatt még szükségünk van továbbá a TensorFlow nevezetű kiegészítő programra. Az MNIST egy adatbázis, mely kézzel írott számjegyeket tartalmaz. Ezt általában képfelisemrő progra- mok-hoz szokták felhasználni. A programot Python nyelven fordítjuk, viszont kell hozzá még a TensorFlow is, melyeket a következő sorokkal telepíthetünk:

A telepítés menete, hogy használni tudjuk a TensorFlow-ot a programunk futásához:

I. lépés -> `sudo apt install python3-dev python3-pip`

II. lépés -> `sudo pip3 install -U virtualenv # system-wide install`

III. lépés -> `virtualenv --system-site-packages -p python3 ./venv`

IV. lépés -> `source ./venv/bin/activate # sh, bash, ksh, or zsh`

V. lépés -> `pip install --upgrade pip`

VI. lépés -> `pip install --upgrade tensorflow`

Plusz az ezekből való head-öröket implementálnunk kell a programba.

8.2. Mély MNIST

Python

Megoldás forrása:

[TensorFlow weboldal a feladathoz, mint forrás](#)

[TensorFlow MNIST](#)

Lényegbe foglalva a feladatban szereplő témát, segítségül hívom az előbbi feladat során szerzett ismereteinket. Rövid bevezőként némi összehasonlítás során megállapíthatjuk, hogy a jelenlegi kiértékelendő koncepciónkban egy többretegű soft mnist-et kapunk, ami ebből adódóan pontosabban végzi el a kapott műveleteket amelyeket beleimplementáltunk és meghatároztunk. Továbbá némi "hátrányt" is megfigyelhetünk, amit a megfigyeléseink során a nagyobb erőforrásigényben találhatunk meg. Ismeeretünket kiterjesztve közlendő, hogy a TensorFlow egy nagyon erős könyvtár, ami a széles skálájú számokkal dolgozik. Ezáltal is elősegítve az implementáló és tanuló mély neutrális működést.

Kezdeként fontos tisztázni, hogy az első két sor feltétlenül szükséges, mivel boilerplate-ként szerepel jelen esetben. Ugyanis ezáltal töltünk és olvasunk adatot atomataként. A következő pár sor fogja tartalmazni a kiszámítási gráf építését fogja lekreálni, és a futtató gráfot is. Tovább építve szükséges a Softmax Regressziós Modelt, ahol egy gyökérpontokat készítünk a bemeneti képhez és a célkimeneti osztályokhoz. Ezeket az x és $y_$ változókkal jelöljük. Az első változó értéke a 28×28 -as pixeles MNIST-es képet jelöli. A tanulás maga a model kereteiben belül zajlik majd le.

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

[Minecraft-MALMÖ bhax-blog forrás](#)

A programot a Python programnyelven írtuk meg, amely a Mojang által készített Minecraft nevezetű játékhoz kapcsolódik. A feladat fő szegmense, hogy a játékban megtalálható karakterünknek megadunk egy általunk választott lépésszámot, ami egy bármekkora pozitív egész lehet, de a figyelem arra irányul, hogy a bábnak, azaz a karakternek nem szabad megállnia, pontosabban elakadnia sehol sem. A program működése során ugyebár végig, a haladás során beolvassuk a karakterünk körüli blokkokat (a játék kockából álló világból áll) és amíg lehet, addig egyenesen halad. Az elakadás esetén a beolvasott környezeti blokkokat elemezve azt az utat választjuk ki, amelyik hamarabb esik a keresési folyamatba. Ezt úgy vizsgáljuk meg, hogy a folyamat során a karakterünk irányától jobbra kezdjük az átvizsgálást, és az első lehetőséget kihasználva megyünk tovább előre. De mi van akkor ha ez ne működik? Akkor azt vizsgáljuk meg, hogy van-e olyan blokk (ami eggyel magasabb mint a környezetünk) amire fel tudunk ugrani, és ha ilye van, akkor élünk az ebből adódó lehetőséggel. Ezzel az útkeresési és választó aktív folyamatnak hála, a karakterünk hibátlanul, elakadás mentesen tudja megtenni az általunk, a feladat elején megadott megtenni vágyott lépéseknek a számát.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

[Iterációs kép](#)

[Rekurzív kép](#)

[Feladatok](#)

Először is szeretném bevezetni a feladatban szereplő programozási nyelv általános tudnivalóit, hogy még érthetőbb legyen a feladat későbbi magyarázata és értelmezése az olvasó számára. Az alkalmazása elég széles körben elterjedt, ez alapján pedig egy ismert nyelvről lesz szó. A feladat során a Scheme dialektust kell használnunk. A program, és a feladat megoldásához szükségünk van a Gimp képmoduláló programjában található beépített környezethez, a Script-fu-ra, ami a Szűrők elnevezésű panelben lelhető meg.

A forráshoz hozzárendelt iteratív faktoriálisként megírt programkód bemutatásával kezdem a bekezdést. Itt először is definiáljuk az n -t ami lehetőleg valami olyan nevet kapjon, ami a faktoriális kifejezésre hajaz, ezáltal megkönnyítve a későbbi olvasását a kódchipetnek. Emellé még meg kell adnunk egy product nevezetű változót is, amivel az értékvizsgálat fog lezajlani olyan módon, ami majd a rekurzív faktoriális feladatban is kelleni fog. Egy értékvizsgálati eljárás során megnézzük, hogy a kapott eredmény megegyezik-e eggyel, és ha igen, akkor önmagát adjuk vissza válaszként. Egyéb esetben, ahol az értéke nagyobb lesz az előbb említett számnál, azaz az egynél, akkor kiszámolja a faktoriális, és azt adja meg válaszul.

A feladatban említett rekurzív faktoriális megalkotása a következő módon jelenik meg. Először is az n változót, ami egy szám lesz, definiálnunk kell, lehetőleg faktoriálisnak elnevezve az egyszerűség szempontja és átláthatóság végett. Ezt követően egy értékvizsgálattal megállapítjuk az értékét. Amennyiben a kapott érték nagyobb mint egy, ekkor a faktoriális képletbeli folytatása lesz a következő lépés, ami megadja a kapott n értékének a faktoriálisát. Amennyiben az értéke megegyezik eggyel, ugyanazt fogja visszaadni, azaz önmagát.

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:

Gimpes króm effekt

A címben szereplő feladat során ismételten igénybe vesszük, az előző feladat alapján már jól ismert Gimp nevezetű képszerkesztő programot amelyhez most egy szkript megírásával egy új bővítményt, azaz kiegészítőt fogunk hozzárendelni a következőféleképpen. Ahogy a leírás is megfogalmazza, egy krómium effektet fogunk létrehozni a programhoz. A forrásban megosztott kód hordozza és foglalja magába azokat a beállításokat, amelyekkel elérjük a krómos hatást. Alapjában véve ehhez szükségünk lesz az előző feladatban is használt rekurzív függvényt, de erről majd később fejtek ki részletesebben tudnivalókat. A kis kódrészletünkben szerepelnek olyan tömbök, amelyek a színskálákat ölelik fel, plusz egy rekurzív függvény alapú szegmenst, amely a listát felhasználva vesz ki részeket/elemeket, majd egy ehhez hasonlót ami a szélességét elemzi és intermediál a szöveggel, majd a magasságával is. Ezeket használva érjük el a szkript végleges alakját, és ezzel a célunkat is, hogy elérjük a króm alapú és kinézetű hatást. Legvégül pedig ne felejtsük el, hogy a megírt szkript-et be kell építenünk a képszerkesztő programunkba úgy, hogy a fájlunkat bemásoljuk a Gimp megfelelő, direkt az ilyen célra konstruált mappába, amit majd a későbbiekben a kiegészítők közt találunk meg.

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:

Script

Végkifejlet

A program amelyet megvalósítunk eléggé hasonlít az előző feladatban felvetett koncepcióhoz. Itt is a cél egy szkript megírása annyi különbséggel, hogy most nem egy konkrét színeffektet hozunk létre, hanem a címben is szereplő mandala-t, amelynek az eredete indiai képmegalkotási mód. Első sorban, mint általában most is a változók létrehozásával, és definiálásával kell kezdenünk. Ezek után a szoftverünk átmegy a képmalkotási lépcsőkön, hogy elérjük amit szeretnénk, azaz a mandala megalkotását. Első valós lépésünk a kívánt hatásért fontos meghatározni, hogy milyen szót, és betűtípust választunk ki, hogy erre vegyük igénybe az általunk megírt, vagy kölcsönvett és használt szkriptet. A kódcipet folyamatosan pörög ezalatt a procedúra alatt a háttérben, hogy használja a meghatározott folyamatot a célért.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

1.2. A magas szintű nyelven megírt programokat forrásszövegnek, a nyelvtani szabályokat pedig szintaktikai szabályoknak nevezzük. Fordítóprogrammal tudjuk fordítani, lexikális egységekre bontja tárgyprogramot hoz létre. Ezt majd futtatni kell, amit a futtató rendszer felügyel. Az interpretés technika nem készíti tárgyprogramot, utasításonként elemzi a programot, értelmezi és végrehajtja. Minden nyelvnek megvan a saját szabványa, a hivatkozási nyelve. Az implementációk a fordítóprogramok vagy az interpreterek, egy platformon több is lehet belőlük.

1.3. Imperatív nyelvek: algoritmus mely működteti a processzort, utasítások, változók, eljárásorientált- és objektumorientált nyelvek. Deklaratív nyelvek: nem kötődnek a Neumann architektúrához, funkcionális- és logikai nyelvek

1.4. Jelölésrendszerek: terminális, nem terminális, alternatíva, opció, iteráció. Ezekkel szintaktikai szabályok lehet írni: bal oldalon nem formális, :, majd elemsorozat. Programozni megtanulni papíron nem lehet.

2.4. Az adattípus absztrakt programozási eszköz, más eszköz komponenseként jelenik meg. Az adattípust a tartomány (amilyen típust felvehet), a művelet (amit a tartomány elemein elvégezhet) és a reprezentáció (az értékek tárban való megjelenése) határoz meg. A programozó is definiálhat típusokat. Az alaptípusból tudunk más típust (alaptípust) származtatni, szűkített tartománnyal de ugyanolyan műveletekkel.

2.4.1. Az egész típus belső ábrázolása fixpontos, a valószínű lebegőpontos. A két típus együtt a numerikus típusok, mivel numerikus műveletek hajthatók rajtuk végre. A karakter típus elemei a karakterek, a karakterlánc pedig a sztringek. A logikai típus tartományának elemei a hamis és az igaz értékek. A felsorolás típust saját típusként lehet létrehozni, megadva tartomány elemeit. Sorszámozott típusnak számít az egész, karakteres, logikai és felsorolás típusok (minden elemnek van megelőzője és rákövetkezője). A sorszámozottak egy alaptípusa az intervallum típus.

2.4.2. Összetett típus a tömb és a rekord. A tömb elemei azonos típusúak, lehetnek egy vagy többdimenziósak. A tömb elemeire indexek segítségével tudunk hivatkozni. Absztrakt adatszerkezetek folytonos ábrázolására alkalmas. A rekord tartományának elemei értékcsoporthoz, melyek elemei (mezői) különböző típusúak lehetnek. Minden mezőnek van neve és típusa. A C és a Pascal esetében is a rekord típusa egyszintű (nincsenek almezők), de a mezők típusa lehet összetett. A rekord nevével (eszköznév) az összes mezőre tudunk hivatkozni, ha csak egyre akarunk akkor: eszköznév.mezőnév.

2.4.3. A mutató típus tartományának elemei tárcímek, így valósítjuk meg az indirekt címezést (minden mutató egy címre mutat). Van egy speciális eleme, a NULL, ami nem mutat sehova.

2.5. A nevesített konstansnak van neve, típusa és értéke. Mindig deklarálni kell. Szerepe, hogy a sokszor előforduló értékeket beszélő névvel látjuk el, így könnyebben tudunk utalni rá a szövegben, illetve ha az értékét ki akarjuk cserélni, akkor ezt elég egy helyen megtenni.

2.6. A változónak van neve, attribútuma, címe és értéke. A kódban mindig a név jelenik meg, ehhez különböző módon rendeljük hozzá a másik hármat. Az attribútum a változó viselkedését határozza meg a futás alatt (ez többnyire a típust jelenti). Háromfajta deklarációt különböztetünk meg itt: explicit, implicit és automatikus. A változó címe a tárnak azt arészét határozza meg, ahol a változó értéke van. A cím rendelkezhet: statikus, dinamikus és a programozó által vezérelt tárkiosztással. A változó értéke a címen elhelyezkedő bit-kombináció. Az érték meghatározható: érékadó uatsítással. A kezdőértékadás lehet: explicit (a programozó választ a típusnak megfelelő tetszőleges értéket) és automatikus (a programtól kap 0-áskezdőértéket).

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

1.1. Annak a módja, hogy egy programozási nyelvet elsajátítsunk, nem más mint higy programokat írjunk az adott nyelven. A C nyleg tanulását is kezdjük úgy, mint más nyelvek tanulása esetén, a legegyszerűbb programmal a Hello World magyar változatával. Fordítjuk: cc figyel.c, majd futtatjuk: ./a.out. Mivel a cc-vel fordítottuk ezért egy alapértelmezett kimenetbe megy az a.out-ba. A C programok tetszőleges nevű függvényeket tartalmaznak amelyek a számítási műveleteket határozzák meg. Speciális függvény a main (főprogram) mely minden programban elő kell forduljon. Itt hívjuk meg az előre megírt függvényeket, vagy itt írjuk meg. A program olyan sorrendben végzi el az utasításokat ahogy azok a main-be vannak. A függvény neve után szereplő () az argumentum listát tartalmazza, a {} pedig az utasítás listát (lehetnek üresek is). A printf("Figyelem emberek\n") függvényhívás a kimenetbe írja a Figyelem emberek szöveget, a végére pedig egy sortörést.

1.2. Egysoros kommentet a //-el, többsorosat pedig a /* */ jelek segítségével hozunk létre. Az int az egész, a float a lebegő, char a karakter, stb típusú változókat jelöli. Sorok végét ; -vel zárjuk le. A while ciklus működése: akkor áll le, amikor a feltétel hamissá válik. A printf-ben lévő %-al meg tudjuk adni, hogy a kiírandó szám milyen típusú lesz.

1.3. A for ciklus sokban hasonlít a while-hoz, ő is akkor lép ki a ciklusból, ha a feltétel hamissá válik.

1.4. A #define segítségével a program elején szimbolikus állandót vagy változót adunk egy karakterlánchoz.

1.5. A getchar() függvény híváskor karaktert olvas be, a putchar() hívásakor pedig karaktert ír a kimenetre.

10.3. Programozás

[BMECPP]

A könyv első részében (16.oldalig) a C++ nem objektumorientált különbségei, újításai vannak a C nyelvhez képest.

Függvényparaméterek és visszatérési érték: az üres paraméterlista azt jelenti, nincs a függvénynek paramétere, nem pedig azt, hogy tetszőleges számú paraméterrel hívható.

A main függvénynek tudunk parancssor argumentumokat adni, illetve nem kötelező a return 0; használata, ez alapértelmezés.

A bool típus használata átláthatóbb, praktikusabb kódot eredményez.

A C++-ban mindenhol állhat deklaráció, ahol utasítás is.

Függvények túlterhelése: ugyanolyan néven több függvényt is létrehozhatunk, amennyiben különbözik az argumentumlistájuk. A függvényt nem csak a neve, hanem argumentumlistája is meghatározza, ezáltal a névadás egyszerűbbé válik.

Alapértelmezett függvényargumentumok: megadhatunk alapértelmezett függvényargumentumokat is, melyek híváskor elhagyhatók. Arra viszont ügyelnünk kell, hogy milyen sorrendben hagyhatók el.

Refetípussal történő paraméterátadás: a cím szerinti átadás egy módszere, mely sokat egyszerűsíthet a függvényünkön, nem kell mindenhol mutatót használni, ha referenciaátadást alkalmazunk. Itt is vigyázni kell néhány dologra, például arra, hogy egy függvényben sose adjunk vissza pointert vagy referenciát lokális változóra vagy érték szerint átadott paraméterre.

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.