

Univerzális programozás

Így neveld a programozód!

Ed. pkristof1999, Hosszúpályi, 2019. április 27., v. 0.9.0

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 Pankotai, Kristóf	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	Munka elkezdése, adatok bevitele a szerzőkhöz. Touring csokor első 3 feladatának megoldása. Saját repository-ba való feltöltés.	pkristof1999
0.0.6	2019-03-04	Touring csokor következő 4 feladatának megoldása, első 3 finomítása, utolsó megnézése.	pkristof1999

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.7	2019-03-05	Touring csokor befejezése. Chomsky csokor feladatainak tanulmányozása.	pkristof1999
0.0.8	2019-03-08	Chomsky csokor kidolgozásának megkezdése.	pkristof1999
0.0.9	2019-03-11	Chomsky csokor kidolgozásának felfüggesztése/befejezése.	pkristof1999
0.1.0	2019-03-15	Caesar csokor elkezdése.	pkristof1999
0.2.0	2019-03-17	Caesar csokor feladatainak további tanulmányozása, megoldása.	pkristof1999
0.3.0	2019-03-18	Ceaser csokor befejezése legjobb tudás szerint.	pkristof1999
0.4.0	2019-03-25	Mandelbrot csokor labor feladatainak elkészítése, Gutenberg csokor elkezdése.	pkristof1999
0.5.0	2019-03-31	Welch feladatcsokor elkészítése.	pkristof1999
0.6.0	2019-04-6	Welch feladatkörön történő finomítások, Conway csokron történő dolgozás.	pkristof1999
0.7.0	2019-04-12	Schwarzenegger csokor elkészítése.	pkristof1999
0.8.0	2019-04-21	Chaitin csokor tanulmányozása, előző csokrok apróbb javítása, helyesírás ellenőrzés dokumentum szinten.	pkristof1999

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.9.0	2019-04-27	Chaitin csokor kidolgozása, Gutenberg finomítása, kiegészítése, átalakítása.	pkristof1999

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	9
3. Helló, Chomsky!	11
3.1. Decimálisból unárisba átváltó Turing gép	11
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	11
3.3. Hivatkozási nyelv	11
3.4. Saját lexikális elemző	12
3.5. Leetspeak	12
3.6. A források olvasása	12
3.7. Logikus	13
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	17
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	29
9.3. Gimp Scheme Script-fu: név mandala	30
10. Helló, Gutenberg!	31
10.1. Programozási alapfogalmak (Juhász István - Magas szintű programozási nyelvek 1)	31
10.2. Programozás bevezetés (Kernighan Ritchie - A C programozási nyelv)	32
10.3. Programozás (Benedek Zoltán, Levendovszky Tihamér: Szoftverfejlesztés C++ nyelven) .	32
III. Második felvonás	33
11. Helló, Arroway!	35
11.1. A BPP algoritmus Java megvalósítása	35
11.2. Java osztályok a Pi-ben	35
IV. Irodalomjegyzék	36
11.3. Általános	37
11.4. C	37
11.5. C++	37
11.6. Lisp	37

Előszó

Amikor programozónak terveztem állni, ellenezték a környezetemben, mondván, hogy kell szövegszerkesztő meg táblázatkezelő, de az már van... nem lesz programozói munka.

Tévedtek. Hogy egy generáció múlva kell-e még tömegesen hús-vér programozó vagy olcsóbb lesz allokálni igény szerint pár robot programozót a felhőből? A programozók dolgozók lesznek vagy papok? Ki tudhatná ma.

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

C (mutatók), C++ (másoló és mozgató szemantika) és Java (lebutított C++) nyelvekből kell egy jó alap, ezt kell kiegészíteni pár R (vektoros szemlélet), Python (gépi tanulás bevezető), Lisp és Prolog (hogyan lássuk mást is) példával.

Hogyan nyomjuk?

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

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

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

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:

[Minden szál 100%-on](#)

[Egy szál 100%-on](#)

[Egy szál altatása](#)

Tanulságok, tapasztalatok, magyarázat...

Egy mag futtatása 100%-on egy egyszerű üres ciklussal megtehető. Több magnál a végtelen cikluson felül szükség van az OpenMP meghívására és fordításnál pedig az "-fopenmp" parancsra. Altatásnál ciklusban a Sleep() függvényre van szükség, amihez meg kell hívni az unistd.h könyvtárat.

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 forrása: tegyük fel, hogy akkora haxorok vagyunk, hogy meg tudjuk írni a Lefagy függvényt, amely tetszőleges programról el tudja dönteni, hogy van-e benne végtelen ciklus:

```
Program T100
{
    boolean Lefagy(Program P)
    {
        if(P-ben van végtelen ciklus)
            return true;
        else
            return false;
    }
}
```

```
}  
  
main(Input Q)  
{  
    Lefagy(Q)  
}  
}
```

A program futtatása, például akár az előző v. c ilyen pszeudókódjára:

```
T100(t.c.pseudo)  
true
```

akár önmagára

```
T100(T100)  
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra építő Lefagy2 már nem tartalmaz feltételezett, csak konkrét kódot:

```
Program T1000  
{  
  
    boolean Lefagy(Program P)  
    {  
        if(P-ben van végtelen ciklus)  
            return true;  
        else  
            return false;  
    }  
  
    boolean Lefagy2(Program P)  
    {  
        if(Lefagy(P))  
            return true;  
        else  
            for(;;)  
    }  
  
    main(Input Q)  
    {  
        Lefagy2(Q)  
    }  
}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true
- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

akkor most hogy fog működni? Sehogy, mert ilyen `Lefagy` függvényt, azaz a T100 program nem is létezik.

Tanulságok, tapasztalatok, magyarázat...

Elkészíteni egy olyan programot, amely meg tudja nézni, hogy egy másik program lefagy-e (végtelen-e) nem lehetséges, mivel olyan kód nem létezik, amely egy másik program végtelen ciklusát "megelőzve" annak a végére érhetne, és valóban kimondhatná, hogy igen, ez egy olyan program, ami lefagyott (végtelen ciklus).

2.3. Változók értékének felcserélése

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

Megoldás forrása:

[Változók felcserélése](#)

Tanulságok, tapasztalatok, magyarázat...

Változók cseréjénél a felhasználótól 2 darab egész számot bekérünk, majd egy egyszerű összeadás/kivonás sorozattal ezeket megcseréljük, majd kiírjuk.

2.4. Labdapattogás

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

Megoldás forrása:

[Labdapattogatás if-fel](#)

[Labdapattogatás if nélkül](#)

Tanulságok, tapasztalatok, magyarázat...

Az if-es labdapattogatásnál szükségünk van egy új ablak megnyitásához, az ablak méretének lekérdezéséhez, ahhoz hogy hol jár épp a labda és egy kezdőértékhez. A For cikluson belül valósul meg, hogy a program a "getmaxyx" segítségével lekérdezi az ablak méretét, az "mvprintw" függvény kirajzolja a labdát, majd az if elágazásoknál nézi a program, hogy mennyit kell adni a változókhoz. Az if nélküli labdapattogatásnál előre megadott pályamérettel dolgozunk, tömbök és for ciklusok segítségével nézzük, hogy mikor éri el a labda a pálya szélét, és függően attól, hogy melyik oldalt éri el, úgy vált x vagy y plusz előjeltől negatívra, amellyel az irányváltoztatás létrejön.

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](#)

Tanulságok, tapasztalatok, magyarázat...

(Ratku Dániel segítségével) Ebben a feladatban az int típusnak a Bitwise operátorral megkapott értékét kell kiszámolnunk. Itt két különböző int-tel dolgozunk, az egyik értéke 0, a másik pedig 0x01, egy do-while ciklus segítségével a 0 értékű változót folyamatosan növeljük eggyel, egészen addig, amíg el nem érjük a 0x01 változót left shifteléssel.

2.6. Helló, Google!

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

Megoldás forrása: [PageRank](#)

Tanulságok, tapasztalatok, magyarázat...

A PageRank nevezetű algoritmus a Google-höz kapcsolódik. Célja az, hogy azok a weboldalak, amelyekre több kattintás gyűlik össze, előrébb kerüljenek a ranglistán arra alapozva, hogy valószínűleg érdekesebb és relevánsabb tartalom van bennük, tehát a weboldalak "jóságát" veszi figyelembe. A program egy egyszerű 4 weboldalas esetet mutat be mátrixokkal és vektorokkal, habár még teljes mértékben nem látom át, ezért még nem tudom részletezni a kód jelentését.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás forrása: [Brun Tétel](#)

Tanulságok, tapasztalatok, magyarázat...

(Ratku Dániel segítségével) A Brun tétel azt mondja ki, hogy az olyan egymást követő prímek, amelyek különbsége kettő, azoknak a reciprokösszege nem a végtelenbe, hanem egy ún. Brun konstanshoz tart. A programon belül deklaráljuk x-ig a prímszámokat 1-től, illetve 2-től is. Külön felvesszük azt a változót, amely azokat a prímekeket gyűjti ki, amelyeknek a különbsége kettő, majd ezeket eltároljuk. Az eltárolt prímeknek vesszük a reciprokösszegét, majd az összeset összeadjuk egy változóba. Majd a matlab könyvtár segítségével ezt meg is jeleníthetjük.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása: [Monty Hall probléma](#)

Tanulságok, tapasztalatok, magyarázat...

(Ratku Dániel segítségével) A feladat szituációja a következő, adott egy játékos és egy műsorvezető, plusz 3 ajtó, amelyből 2 vereséget és 1 nyereséget rejt. Ezek a "jutalmak" ajtók mögé vannak rejtve és a felhasználónak kell egyet kiválasztania a lehetőségek közül. A feladat során feltételezzük, hogy a műsorvezető pontosan tudja a győztes ajtó számát. A műsornak a felépítése olyan, hogy a játékosnak választania kell egy ajtót, majd a műsort levezető személynek választania kell a választott ajtótól egy különbözőt, amely nem tartalmazza a jutalmat. A probléma abban rejlik, hogy eredetileg $1/3$ -ad az esély a győzelemre, viszont miután a műsorvezető kinyitja az általunk választottól különböző ajtót, azt feltételezzük, hogy azért azt választotta, mert nem abban van a nyeremény, ezáltal pedig a győzelmi esélyünk már $2/3$ -ra nőtt. A kódunkban bármekkora számmal hozhatjuk létre a kísérletek számát, majd ezután deklaráljuk a kísérletek és játékosok nevezetű változót, amelyben ott van a lehetőségek száma és mellékeljük a `replace=T`-t, ami engedélyezi ezeknek az ismétlődését. A műsorvezető egy vektor lesz, aminek a hossza megegyezik a kísérletek számával, a műsor folyását egy for ciklussal kezdjük, ami átmegy az összes kísérleten, majd a beleépített if elágazásokkal megvizsgáljuk, hogy mit választott a játékos. Az eredeti ágnál megegyezik a játékos tippje a helyes ajtóval, ekkor a műsorvezető kivonja a 3 lehetőség közül azt amelyiket választott a játékos, egyéb esetben a házigazda már csak azt az ajtót tudja kiválasztani, ami mögött nem a jutalom van. Következő lépésben meghatározunk 2 esetet, az egyik amikor a játékos változtat, a másik amikor nem. Az utóbbi esetén a tipp megegyezik a győzelemmel. A következő for ciklus a változtatás esetén zajlik, amikor már arra az ajtóra váltunk, ami mögött a problémából adódóan feltételezzük a jutalom hollétét. Legvégül pedig kiírjuk a kísérletek számát.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiájával megadva írd meg ezt a gépet!

Megoldás forrása:

[Unárisba váltó](#)

[Ábra](#)

Tanulságok, tapasztalatok, magyarázat...

A program lényege, hogy decimálisból unárisba, azaz egyes számrendszerbe váltson át pozitív egész számokat. A program egy egyszerű for ciklus, amely a megadott számig vonásokat húz, amelyet minden 5. vonás után szóközzel választ el.

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 forrása:

Tanulságok, tapasztalatok, magyarázat...

A feladat nem kristály tiszta, még megoldás/kigondolás alatt áll, később, ha megértem a lényegét akkor visszatérek rá.

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.

Tanulságok, tapasztalatok, magyarázat...

A BNF (Backus-Naur-forma) segítségével környezetfüggetlen grammatikákat tudunk leírni. Széleskörben alkalmazzák a programozási nyelvek szintaxisának leírására. A C nyelv C99-es kiadása a C89-hez képest sok újítást hozott, viszont az egyik legfontosabb, illetve legegyszerűbb újítás az, hogy C89-hez képest a C99-es nyelvben lehet C++ típusú kommentelést 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:

[Lexikális elemző](#)

Tanulságok, tapasztalatok, magyarázat...

(Ratku Dániel segítségével) 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:

[Leetspeak](#)

Tanulságok, tapasztalatok, magyarázat...

Ez a program l nyelvből készít egy c forrást, amely felismeri a karaktereket és kódolja őket más hasonlóakkal. Az l kód lényegében annyit csinál, hogy minden betűhöz és számhoz hozzárendel egy 4 karaktert tartalmazó tömböt, amelyekből betűnként véletlenszerűen választ majd cseréli ki.

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 megránézésre, 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);
```

Tanulságok, tapasztalatok, magyarázat...

Első: Ha eddig nem volt figyelmen kívül hagyva a SIGINT, akkor a jelkezelő függvény kezelje, ha figyelmen kívül hagyva, akkor maradjon is így.

Második: Egyszerű for ciklus, az i-t 0-tól kezdve addig növelje 1-el amíg el nem éri a 4-et.

Harmadik: Mivel for ciklusban nem számít, hogy i++ vagy ++i, így ez megegyezik az előzővel.

Az ezt követő algoritmusokat nem igazán értem, de a félév további részében, amint megértem visszatérek és kiegészítem.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\texttt{\textbackslash forall} x \texttt{\textbackslash exists} y ((x < y) \texttt{\textbackslash wedge} (y \texttt{\textbackslash text} \{ \texttt{prím}})))$
$(\texttt{\textbackslash forall} x \texttt{\textbackslash exists} y ((x < y) \texttt{\textbackslash wedge} (y \texttt{\textbackslash text} \{ \texttt{prím}})) \texttt{\textbackslash wedge} (SSy \texttt{\textbackslash text} \{ \texttt{prím}})) \leftarrow$
  )$
$(\texttt{\textbackslash exists} y \texttt{\textbackslash forall} x (x \texttt{\textbackslash text} \{ \texttt{prím}}) \texttt{\textbackslash supset} (x < y))$
$(\texttt{\textbackslash exists} y \texttt{\textbackslash forall} x (y < x) \texttt{\textbackslash supset} \texttt{\textbackslash neg} (x \texttt{\textbackslash text} \{ \texttt{prím}})))$
```


Megoldás forrása:

Logikus

Tanulságok, tapasztalatok, magyarázat...

Ez a feladat több elsőrendű logikai állítást fogalmaz meg az ar nyelven, melyekből először vázoljuk, hogy melyik kifejezés mit jelent. Két univerzális kvantor van: az "exist" az azt jelenti, hogy létezik olyan..., a "forall" pedig, hogy bármely...; Továbbá a "neg" a negációt, a "supset" a konjunkciót és a "wedge" pedig az implikációt jelöli.

Az első kifejezés: Bármely x esetén létezik olyan y , ahol ha x kisebb, akkor y prímszám.

Második kifejezés: Bármely x esetén létezik olyan y , ahol ha x kisebb, akkor y prímszám és ha y prímszám, akkor az azt követő utáni szám is prím.

Harmadik kifejezés: Van olyan y , ahol bármely x esetén az x prím és kisebb, mint y .

Negyedik kifejezés: Van olyan y , ahol bármely x -nél az x nagyobb, mint y 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:

[Deklaráció](#)

Tanulságok, tapasztalatok, magyarázat...

Nem teljesen látom át a programot, bár a kódot megkaptam és a későbbiekben ezt még kidolgozom segítséggel.

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:

[Háromszögmátrix](#)

Tanulságok, tapasztalatok, magyarázat...

A program egy olyan négyzetes mátrix (négyzetes mátrix: sorainak és oszlopainak a száma megegyezik) alsó háromszögét számítja ki, melynek a főátló felett kizárólag nulla áll. A program a memóriában 40 bájtot foglal le a malloc függvény segítségével.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása:

[C EXOR titkosító](#)

[Forrás \(SourceForge, UDPROG\)](#)

Tanulságok, tapasztalatok, magyarázat...

Ez a titkosító program az EXOR-ra, azaz a kizáró vagyra épít. A program lényege, hogy egy felhasználó által megadott 8 karakteres int kulcs segítségével (azért 8, mert a későbbiekben a visszafordító program 8 karakteres kulccsal dolgozik) egy szöveget lekódoljunk, amit majd a későbbiekben ugyanazzal a kóddal dekódolni is tudunk. A program két konstans értékkel dolgozik: a kulcs és a buffer maximális méretével, amit a program elején kell definiálni. A program fő ágában deklaráljuk magát a kulcsot és a buffert, és azokat a változókat, amelyek tárolják a kulcs hosszúságát és a beolvasott bájtok mennyiségét. A továbbiakban pedig rögzítjük a felhasználó által megadott kulcsnak a méretét, ez lesz az első parancssori argumentum is. Majd egy while ciklus segítségével megyünk végig a beolvasott bájtokon és EXOR séma segítségével titkosítjuk őket. Lefuttatni pedig a következővel tudjuk: először is a forrást "gcc eClean.c -o ec"-vel fordítjuk majd a következőkben "./ec "kulcs" <tisztaszöveg.txt >titkosszöveg.txt".

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

[Java EXOR titkosító](#)

Forrás: www.tankonyvtar.hu

Tanulságok, tapasztalatok, magyarázat...

Ez az EXOR törő a fentebb megoldott feladat java-ban megírt megfelelője. Működése elvben és gyakorlatban is megegyezik. Lefordítása pedig a következő: "javac -encoding UTF-8 ExorTitkosító.java" majd futtatása pedig: "java ExorTitkosító "kulcs" < titkosított.txt".

4.4. C EXOR törő

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

Megoldás forrása:

[C EXOR törő](#)

Forrás ([SourceForge](#), [UDPROG](#))

Tanulságok, tapasztalatok, magyarázat...

Ez a program a fent EXOR-ral lekódolt szöveget kódolja vissza a megfelelő 8 int hosszúságú kulcs segítségével. Ebben a programban is definiáljuk, hogy mennyi konstansokkal dolgozzon. Kezdetnek az első for ciklusnál kiszámoljuk, hogy mennyi az átlagos szóhossz szóközök segítségével, ami azért fontos, hogy könnyebben válassza el a szavakat. Ezután jöhet az EXOR végrehajtása bájtanként. Amíg van karakter a szöveges fájlban, addig fut a program, majd amint elfogynak kiürítjük a buffert is. Majd a for ciklusokkal minden lehetséges kulcsot előállítunk. Lefuttatni pedig a következővel tudjuk: először is a forrást "gcc tClean.c -o tc"-vel fordítjuk majd a következőkben ".tc "kulcs" <titkosszöveg.txt >tisztaszöveg.txt".

4.5. Neurális OR, AND és EXOR kapu

Megoldás forrása:

[Neurális OR, AND és EXOR kapu](#)

Tanulságok, tapasztalatok, magyarázat...

(Tutorom Pataki Donát) Ezt a programot az R nyelvben írtuk. A program elnevezése utal az agy összetevőjére, a neuronokra, melyek feladata az elektromos összeköttetés. A neurális háló az, "ahol" and, "or" és "exor" kapuk vannak beágyazva. Az "and" és az "or" működése egyértelmű. Az előbbinél ha mindkettő 1 akkor lesz az eredményük az 1. Utóbbinál, ha az egyik 1 akkor lesz az eredményük az 1. "Exor"-nál pedig ha a két érték nem egyezik meg (0 és 1 esetén), akkor lesz az eredmény 1 és ez fordított esetben is igaz. A neurális hálór 3 részre lehet osztani. Van az "input layer", ahol megkapja az adatokat, a "hidden layer", ahol a varázslat történik és az "output layer", ahol ha minden jól működik a megfelelő értékeket kapjuk vissza.

4.6. Hiba-visszaterjesztéses perceptron

Megoldás forrása:

[Hiba-visszaterjesztéses perceptron](#)

Tanulságok, tapasztalatok, magyarázat...

(Tutorom Pataki Donát) A perceptron egy algoritmus model, amely az emberi agy működését próbálja utánozni. Hasonló a neurális hálózathoz azonban van pár különbség. Ugyanúgy "input" után elkezd "varázsolni" és jobb esetben, ha megfelelő a mintavétel, akkor helyes eredményt ad vissza. Azonban a középső értékeknek van súlya, amit még adott konstansokkal is ki lehet egészíteni. Az így kapott súlyokat összeadja és ha ez elér egy bizonyos szintet, akkor a program adott része aktiválódik és egy lineáris folyamat amely addig 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](#)

Tanulságok, tapasztalatok, magyarázat...

A feladat megkezdése előtt fontos tudni, hogy mik is a fraktálok, és hogyan kapcsolódnak a Mandelbrot halmazhoz. A fraktálok olyan alakzatok, amik végtelenül komplexek, két fő tulajdonságuk, hogy a legtöbb geometria alakzattal ellentétben a fraktálok szélei nem egyenletesek, és, hogy nagyon hasonlítanak egymásra. Ha egy kör szélét folyamatosan nagyítjuk, egy idő után egyenesnek látjuk. Ezzel szemben a fraktálok szakadozottak maradnak függetlenül a nagyítás mértékétől. A program futtatásához először is szükségünk lesz a libpng++ könyvtárra, amelyet a következő paranccsal szerezhetünk be: "sudo apt-get install libpng++-dev". Majd a fordítás a következőképp zajlik: "g++ mandelpngt.c++ -lpng16 -O3 -o mandelpngt". A program célja egy png kép létrehozása a Mandelbrot halmazról. Futtatásához a terminálba a következőt két argumentumot kell megadni: "./mandelpngt t.png".

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](#)

Tanulságok, tapasztalatok, magyarázat...

Ez a feladat az előzőt veszi alapul, kiegészítve. Egyik legfontosabb különbség, hogy itt a valós és a képzetes változókat mostmár együtt kezeljük, melyet számunkra lehetővé teszi a library, melynek segítségével a gép képes kezelni ezeket a számokat. Következő pedig, hogy a felhasználó adhatja meg a létrehozandó kép paramétereit, viszont nem okoz hibát az sem, ha nincs megadva érték, ilyenkor az alapértelmezett értékeket használja a program. Fordításhoz a következőre lesz szükségünk: "g++ 3.1.2.cpp -lpng -O3 -o 3.1.2", majd futtatáshoz pedig ". /3.1.2 mandel.png (és a programban is megtalálható értékek)". A libpng++ könyvtárra itt is szintén szükségünk lesz, ahogyan az első feladatban is.

5.3. Biomorfok

Megoldás forrása:

[Biomorfok](#)

Tanulságok, tapasztalatok, magyarázat...

Ez a feladat nagyon hasonlít a Mandelbrot halmaznál megírt programra, viszont itt nem kimondottan a Mandelbrot halmazzal foglalkozunk, hanem egy olyan halmazzal ami nagyon szoros kapcsolatban áll a Julia halmazzal. Fontos tudni hogy a Mandelbrot halmaz tartalmazza az összes lehetséges Julia halmazt és értelemszerűen ez fordítva nem igaz. A Biomorfokat Clifford Pickover fedezte fel miközben egy Julia halmazt rajzoló prgramon dolgozott. A programot a következőképp fordítjuk: "g++ 3.1.3.cpp -lpng -O3 -o 3.1.3", majd futtatjuk: "./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10". Ennél is fontos a libpng++ könyvtár megléte, úgyhogy ha eddig az nincs meg akkor nézzünk vissza az első feladatra.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása:

[A Mandelbrot halmaz CUDA megvalósítása](#)

Tanulságok, tapasztalatok, magyarázat...

Ez a feladat teljesen megegyezik az első feladatban látottakkal, annyi kis különbséggel, hogy a megoldáshoz szükségünk van egy nVidia videokártyához, hogy az nVidia által kifejlesztett nyelven, illetve fordítóval tudjuk használni. A fordító program letöltéséhez szükségünk lesz a következő parancssorra: "sudo apt install nvidia-cuda-toolkit". Ezzel letöltődik a fordításhoz szükséges környezet, nagyon fontos tudni, hogy habár települhet, illetve jónak tűnhet AMD típusú hardveren, ne is fáradjunk a kipróbálásával mert nem fog működni (személyes tapasztalat). Majd miután települt a következőképp fordíthatjuk: "nvcc mandelpngc_60x60_100.cu -lpng16 -O3 -o mandelpngc", ezt követően pedig: "./mandelpngc c.png".

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](#)

[Forrás \(SourceForge, UDPROG\)](#)

Tanulságok, tapasztalatok, magyarázat...

A forráskód a QT GUI-t használja, ennek segítségével tudjuk elkészíteni programunkat. Ez az egyik legertejedtebb garfikus felülete a C++-nak. Fordítás a következőképp megy végbe: a futtatáshoz szükséges 4 fájlnak egy hely kell lenniük. A mappán belül megnyitjuk a terminált, majd beírjuk a "qmake -project" parancsot. Ez létre fog hozni egy "*.pro" fájlt, minekután a fájlba be kell írni a következőt: "QT += widgets sort". Majd futtatni kell a "qmake *.pro" parancsot, azután létrejön a mappában egy ún. "Makefile" (ezt kell majd használni). Terminálba beírjuk a következőt: "make", ami létrehoz egy fájlt, amit a szokásos módon futtatunk. Futás közben az "n" betűt lenyomva nagyíthatjuk ki a szóban forgó képet.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása:

[Mandekbrot nagyító és utazó \(java\)](#)

Tanulságok, tapasztalatok, magyarázat...

Ez a feladat pontosan az előző feladatnak a java nyelven történő megvalósítása.

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

Megoldás forrása:

[Első osztályom \(a mappa tartalmazza a java és a c++ forrásokat is\)](#)(Forrás: UDPROG)

Tanulságok, tapasztalatok, magyarázat...

Ebben a programban polártranszformáció segítségével véletlenszerű számokat tudunk kiszámolni. A Java véletlenszerű-szám generátora is ezt használja. A polártranszformációs algoritmus két "véletlen" értéket generál. A megadott forrásokat összehasonlítva rájöhethetünk, hogy a java kódja egyszerűbb és átláthatóbb.

6.2. LZW

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

Megoldás forrása:

[LZW](#) (Forrás: UDPROG)

Tanulságok, tapasztalatok, magyarázat...

Maga az LZW fa lényege, hogy futás közben egy általunk megadott szöveges dokumentum sorait fogjuk beolvasni, és ezeket az értékeket bináris kóddá alakítjuk, amelyeket egymásba ágyazott for ciklusokkal rendezünk. A program természetesen addig olvas, amíg van nem üres sor és úgy építi fel a fát, hogy mindig leellenőrzi azt, hogy van-e 1-es vagy 0-ás gyermek, ha nincs akkor létrehozza azt, majd visszaugrik a gyökérre. Ellenkező esetben az 1-es és a 0-ás gyermekre lép, és mindaddig lépked a fában, ameddig nem talál egy olyan részfat, ahol létre kellene hozni egy új gyermeket. Ezekután visszaugrik a gyökérre.

6.3. Fabejárás

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

Megoldás forrása:

[LZW \(preorder/posztorder\)](#)(Forrás: UDPROG)

Tanulságok, tapasztalatok, magyarázat...

Ez a feladat megegyezik az előző feladattal, különbség a bejárési mintában keresendő. Míg az alapnak tekinthető Inorder (amely az előző feladatban is megoldásul szolgált) bejárési formája a következő: Inorder (Left, Root, Right), addig a Posztorderé: Postorder (Left, Right, Root), valamint a Preorderé pedig: Preorder (Root, Left, Right). Forrás: <https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/>

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](#)

Tanulságok, tapasztalatok, magyarázat...

A fát felépítő algoritmus megegyezik az előzővel, annyi az eltérés, hogy nem hívunk meg szokásos formát az építéshez, hanem egyszerűen bele "shifteljük" az elemeket a fába.

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 gyökér](#)

Tanulságok, tapasztalatok, magyarázat...

Át kell állítsuk a gyökér létrehozását, a "Heapen" fogunk allokálni neki helyett a "Stack" helyett. Le kell, hogy vegyük az inicializálási listát, mivel a Node konstruktora fog később lefutni. Egyetlen utasítással kell bővítenünk a "destruktor", mégpedig "delete gyoker" a felszabadítások végén.

6.6. Mozgató szemantika

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

Megoldás forrása:

Mozgató szemantika

Tanulságok, tapasztalatok, magyarázat...

Az előző feladatot alapul véve implementáljuk bele a mozgató és mozgató-értékadás alapú konstruktorokat. Mozgató konstruktornál "nullptr"-re állítjuk a BinFa gyökerét, majd erre a BinFára rávisszük az "other" nevű fát, mely lényege, hogy a kapott "lvalue"-ből "rvalue"-t készít, így átvisszük az összes adatát az új fába, ezért felszabadul az a memória, amelyet eddig foglalt és kiürül az "other".

DRAFT

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

Megoldás forrása:

[Hangyaszimulációk \(Forrás: Gitlab.com/nbatfai\)](https://gitlab.com/nbatfai)

Tanulságok, tapasztalatok, magyarázat...

A program a hangyák feromonokkal történő kommunikációját szimulálja. A "main" futtatását a következőképp tegyük: `./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. Az "AntThread" egy "QThread" osztályból származtaott osztály, mivel a számításokat a "main thread"-től, ami a GUI-t kezeli, el akarjuk különíteni, így nem fagy le az Űrlapunk. Ez a Kolónia végzi a hangyák mozgatását, alkalmaztatja a környezetre a hangyák által kibocsátott feromonokat. Ezekenfelül updateli a "világot" is, csökkenti a világ feromonszintjét. Emiatt nem mondható tiszta OOP programnak. 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](#)

Forrás: www.algosome.com

Tanulságok, tapasztalatok, magyarázat...

A Conway-féle sejtautómátát kellett a feladatban elkészíteni. A program egy 2 dimenziós koordináta-rendszerben dolgozik, amelyben találhatók az ún. sejtek, amelyek vagy "élő" vagy "halott" (0 v. 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:

[Qt életjáték \(Forrás: UDPROG\)](#)

Tanulságok, tapasztalatok, magyarázat...

Ebben a feladatban az előző feladat megvalósítása volt a dolgunk c++-ban a Qt környezet segítségével. Ez komplexebb megvalósítás, mint a java-s megoldás, viszont a működési elve ugyanaz.

7.4. BrainB Benchmark

Megoldás forrása:

[BrainB Benchmark"](#)

Tanulságok, tapasztalatok, magyarázat...

A BrainB Benchmark feladata az e-sport 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 megtalálnunk, és ha megtaláltuk, mennyi ideig tart elveszítenünk. Ideális esetben rövidebb ideig tart megtalálnunk, 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

Megoldás forrása:

[Tensorflow](#)

[Porgpáter](#)

[Szoftmax Py MNIST](#)

Tanulságok, tapasztalatok, magyarázat...

Az MNIST egy adatbázis, mely kézzel írott számjegyeket tartalmaz. Ezt általában képfelisemrő programokhoz szokták felhasználni. A programot Python nyelven fordítjuk, viszont kell hozzá még a TensorFlow is, melyeket a következő sorokkal telepíthetünk:

```
sudo apt install python3-dev python3-pip
sudo pip3 install -U virtualenv # system-wide install
virtualenv --system-site-packages -p python3 ./venv
source ./venv/bin/activate # sh, bash, ksh, or zsh
pip install --upgrade pip
pip install --upgrade tensorflow
```

Az elején importáljuk a szükséges könyvtárakat.

8.2. Mély MNIST

Megoldás forrása:

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

[Mély MNIST](#)

Tanulságok, tapasztalatok, magyarázat...

(Tutorom Ratku Dániel) 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 többbéteggő 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. Ismeere-tünket 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 ki-számítási 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 forrása:

[Minecraft-MALMÖ](#)

Tanulságok, tapasztalatok, magyarázat...

Ez egy Python nyelven megírt program a Minecraft nevű játékszoftverhez, melynek a lényege röviden az, hogy a programnak a játékon belül meg kell tennie x lépést úgy, hogy a játékosnak nem szabad elakadnia. A program úgy működik, hogy a körülöttünk lévő blokkokat beolvassa, és amikor elakad, megnézi, hogy hol van a legközelebbi olyan blokk körülötte, amely irányában tovább tud akadálymentesen haladni, ha nincs ilyen akkor pedig olyan blokkot keres amelyre fel tud ugrani (egy szintel fentebb lévő). A keresés úgy megy végbe, hogy a karakter elkezd jobb irányba fordulni, és az első olyan utat választja, amely megfelel a kritériumoknak.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás forrása:

[Iteratív és rekurzív faktoriális Lisp-ben](#)

[Kép az iterációval megírt szkriptről](#)

[Kép a rekurzióval megírt szkriptről](#)

[Megoldáshoz vett minta \(Stackoverflow\)](#)

Tanulságok, tapasztalatok, magyarázat...

A következő kódcsipeteket Lisp-ben fogjuk megírni, amely egy igen elterjedt nyelv. Magát a programot egy Gimp nevezetű képszerkesztő programban készítjük el a Szűrőknél található Script-fu-val. Először a rekurzív faktoriállissal kezdem -mivel ez egyszerűbb-, amely a következő képp épül fel:

Definiálunk egy n számot, amelyet elnevezünk (célszerű fakt-nak vagy faktoriálisnak), majd megvizsgáljuk, hogy nagyobb-e mint 1, ha nem akkor 1-et ad válasznak hívásnál, egyébként pedig folytatjuk a faktoriális képletével, amely majd kiszámolja a megadott érték faktoriálisát.

Következőnek megvizsgáljuk ugyanezt a programot, csak ezúttal az iterációval megírt változatát (ez jóval hosszabb és bonyolultabb is szerintem). Itt a korábbihoz hasonlóan definiált faktoriális mellé még egy product változót is megadunk, majd ugyanúgy mint fentebb, megvizsgáljuk, hogy 1-e a megadott szám, és ha igen akkor a megadott számot írja ki, jelen esetben az 1-et. Egyébként pedig a program az előzőhöz hasonló szabályt megadva kiszámolja nekünk a bevitt érték faktoriálisát.

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 forrása:

[Gimp Scheme Script-fu:króm effekt](#)

Tanulságok, tapasztalatok, magyarázat...

Folytatjuk a Gimpel történő dolgozást ebben a feladatban is az előzőhöz hasonlóan. Ezúttal egy szkriptet fogunk írni, amely króm effektet valósít majd meg a képszerkesztőn belül. Fontos az elején leszögezni, hogy a megírt forrást be kell hogy másoljuk a GIMP azon mappájába, melyekben a szkriptek vannak tárolva. Maga a kód lényegében olyan beállításokat tartalmaz, amelyek összesített hatásával érhetjük el a kívánt króm effektet. Kezdve a króm hatáshoz szükséges színeket tartalmazó tömbök létrehozásával, majd egy az előző feladatban alkalmazott rekurziós függvény mintájára megadott kódcsipetet, melynek lényege itt az, hogy egy definiált listából kivesz egy megadott számú elemet. Folytatva a forrás olvasását ismét egy hasonló függvényt találunk, mely az effekt elkészítéséhez szükséges szöveg szélességét adja meg. Ezután rend szerűen következik a magasság megadása is, amely hasonló alapokon nyugszik mint a szélesség. Majd miután ezek megvannak, a kód végig megy a módosítások megadásán, amellyel elérhetjük a kívánt króm effektet. Legvégül a program vége arra szolgál, hogy a megírt szkriptünket a Gimp is kilistázza a már meglévő szkriptek mellé.

9.3. Gimp Scheme Script-fu: név mandala

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

Megoldás forrása:

[Gimp Scheme Script-fu: név mandala](#)

[Első lépés](#)

[Második lépés](#)

[Eredmény](#)

Tanulságok, tapasztalatok, magyarázat...

Ez a feladat sem másképp kerül megírásra, mint az előző. Ez is egy szkript, mely segítségével ún. Mandalát készíthetünk, mely egy indiai, tibeti és buddhista gyökerekkel rendelkező képalkotási módszer, jelentése: kör, korong, ív.

Itt is hasonlóan definiáljuk a változókat, mint az előző feladatban, majd a program végigfut egy metódus sorozaton, mely megadja az elérni kívánt hatást.

Használatához első dolgunk egy általunk szimpatikusnak vélt betűtípus és szó kiválasztása, majd alkalmazzuk rá a szkriptünket és voilà, kész is.

Eközben a program persze dolgozik, futás közben alkalmazza azokat a módszereket, melyek szükségesek a kívánt végeredmény eléréséhez.

A megadott forrásban lépésről lépésre készítettem képernyőképeket, amelyeket követve tudjuk elkészíteni saját ilyen Mandala-nkat.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak (Juhász István - Magas szintű programozási nyelvek 1)

Alapvetően három szintet különböztetünk meg amikor programozási nyelvekről beszélünk: a gépi szintű, az assembly szintű és a magas szintű. A magas szintű programokat "source code"-nak (forráskódnak) nevezzük, továbbá ezek nyelvtani szabályait szintaktikai, míg értelmezési szabályait szemantikai szabályoknak nevezzük. Ezeket a forrásokat ún. fordítóprogramokkal tudjuk fordítani, mely lexikális elemzés folyamán gépi nyelvet állít elő, amelyet a futtató szoftver működtet a későbbiekben. Létezik egy Interpreteres megoldás is, mely nem gépi nyelvre fordít, hanem utasításonként elemzi a és hajtja végre a megadott forrást. Az összes nyelvnek létezik saját hivatkozási nyelve.

Imperatív nyelvnek nevezzük azt, ha egy olyan algoritmusról van szó, amely működteti a processzort, utasításokat hajt végre. A fő cél az eljárás vagy az objektum végrehajtása. Emellett beszélhetünk még deklaratív nyelvekről is, melyek lényegében logikai, funkcionális nyelvek: szemantikai szabályok megírására szolgálnak. Fontos, hogy itt a programozó nem tud közvetlen hozzáférni a memóriához, vagy azzal műveleteket végezni.

Az adattípus olyan programozási összetevő, mely lehet érték, vagy akár literál is. Absztrakt adattípusról beszélünk, ha a reprezentáció és a műveletek implementációja számunkra ismeretlen. Jellemzően három összetevő alkotja: a tartomány (típus, amit felvesz (akár maga a programozó személy is adhat meg típust)), az ehhez tartozó műveletek (amit azon elemeken tud végrehajtani, melyek részét képezik a megadott tartománynak) és a reprezentáció (az értékek megjelenítése). Az alaptípussal tudunk -habár nem akkora tartománnyal, de megegyező műveletekkel- más típust is leképezni.

Az I/O függ az operációs rendszertől és az implementációtól. Ennek a feladata a memóriába küldeni, vagy onnan fogadni az adatokat. A karakter típus részei a karakterek, melyeknek láncait pedig "string"-nek nevezzük. Az egész típus fixpontos ábrázolású, még a valós típusé lebegő pontos (float point). Ezek a numerikus típusok abból eredően, amilyen műveleteket tudunk velük végrehajtani. Harmadik ilyen típus a logikai, amelynek elemei a 0 vagy 1 (hamis vagy igaz). A mutató típus segítségével egy megcímzett terület értékét érhetjük el. A nevesített konstans három összetevője a név, az érték és a típus.

A változó a fentebb említett imperatív nyelvek egyik fő eszköze, melynek négy komponense van: a cím, érték, az attribútumok és maga a változó neve. Többszörös tárhivatkozásról beszélhetünk akkor, ha egy időben két nem megegyező nevű változó azonos cím és értékkomponenssel rendelkezik.

Az ún. kifejezések szintaktikai metódusok, ezek egészét a típus és az érték adja, komponensei a hagyományos zárójelek "()", az operátor és az operandus. A logikai operátorral bíró kifejezéseknél nem hajtjuk végre az összes műveletet, akkor, ha a művelet elején el lehet dönteni, hogy mi lesz a végeredmény, például ha a művelet első és második része "vagy"-gyal van összekötve, és tudjuk, hogy az első fele igaz, akkor már nem vizsgáljuk a másik felét, mert a végeredmény biztosan igaz.

A forrás lépéseit utasításokkal adhatjuk meg, melyek lehetnek deklarációs vagy végrehajtható típusúak is. Deklarációsnál -a fordító program segítségével- a gépi kód végrehajtható utasításokból jön létre. Utasításra példának a "Ha-különben (if-else)" elágazást tudom felhozni, mely megvizsgál egy feltételt, és ha hamisnak bizonyul, akkor az else ágon halad tovább.

Blokk az a programrész, mely egy másik programrész belsejében helyezkedik el, ugyanis ez nem lehet független/különálló. A blokk nem rendelkezhet paraméterrel, de használható ott, ahol végrehajtható utasítás is jelen van.

10.2. Programozás bevezetés (Kernighan Ritchie - A C programozási nyelv)

Annak a módja, hogy egy programozási nyelvet elsajátítsunk, nem más mint hogy programokat írjunk az adott nyelven. A C nyelg 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. 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.

10.3. Programozás (Benedek Zoltán, Levendovszky Tihamér: Szoftverfejlesztés C++ nyelven)

III. rész

Második felvonás

**Bátf41 Haxor Stream**

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

DRAFT

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, <https://groups.google.com/forum/#!forum/nemespor>, az UDPROG tanulószoba, <https://www.facebook.com/groups/udprog>, a DEAC-Hackers előszoba, <https://www.facebook.com/groups/DEACHackers> (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

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