

Univerzális programozás

Írd meg a saját programozás tankönyvedet!

Ed. BHAX, DEBRECEN,
2019. február 19, v. 0.0.4

Copyright © 2019 Dr. Bátfai Norbert

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

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

<https://www.gnu.org/licenses/fdl.html>

Engedélyt adunk Önnek a jelen dokumentum sokszorosítására, terjesztésére és/vagy módosítására a Free Software Foundation által kiadott GNU FDL 1.3-as, vagy bármely azt követő verziójának feltételei alapján. Nincs Nem Változtatható szakasz, nincs Címlapszöveg, nincs Hátlapszöveg.

<http://gnu.hu/fdl.html>

COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert Ács Kinczel, Gergő	2019. március 27.	

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

Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it.”

—Gregory Chaitin, *META MATH! The Quest for Omega*, [METAMATH]

Tartalomjegyzék

I. Bevezetés	1
1. Vízió	2
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
II. Tematikus feladatok	3
2. Helló, Turing!	5
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	8
2.6. Helló, Google!	9
2.7. 100 éves a Brun tétel	9
2.8. A Monty Hall probléma	9
3. Helló, Chomsky!	10
3.1. Decimálisból unárisba átváltó Turing gép	10
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	11
3.3. Hivatkozási nyelv	12
3.4. Saját lexikális elemző	13
3.5. l33t.1	14
3.6. A források olvasása	14
3.7. Logikus	16
3.8. Deklaráció	16

4. Helló, Caesar!	18
4.1. double ** háromszögmátrix	18
4.2. C EXOR titkosító	19
4.3. Java EXOR titkosító	20
4.4. C EXOR törő	20
4.5. Neurális OR, AND és EXOR kapu	21
4.6. Hiba-visszaterjesztéses perceptron	21
5. Helló, Mandelbrot!	22
5.1. A Mandelbrot halmaz	22
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	22
5.3. Biomorfok	22
5.4. A Mandelbrot halmaz CUDA megvalósítása	23
5.5. Mandelbrot nagyító és utazó C++ nyelven	23
5.6. Mandelbrot nagyító és utazó Java nyelven	23
6. Helló, Welch!	24
6.1. Első osztályom	24
6.2. LZW	24
6.3. Fabejárás	24
6.4. Tag a gyökér	24
6.5. Mutató a gyökér	25
6.6. Mozgató szemantika	25
7. Helló, Conway!	26
7.1. Hangyaszimulációk	26
7.2. Java életjáték	26
7.3. Qt C++ életjáték	26
7.4. BrainB Benchmark	27
8. Helló, Schwarzenegger!	28
8.1. Szoftmax Py MNIST	28
8.2. Szoftmax R MNIST	28
8.3. Mély MNIST	28
8.4. Deep dream	28
8.5. Robotpszichológia	29

9. Helló, Chaitin!	30
9.1. Iteratív és rekurzív faktoriális Lisp-ben	30
9.2. Weizenbaum Eliza programja	30
9.3. Gimp Scheme Script-fu: króm effekt	30
9.4. Gimp Scheme Script-fu: név mandala	30
9.5. Lambda	31
9.6. Omega	31
 III. Második felvonás	 32
10. Helló, Arroway!	34
10.1. A BPP algoritmus Java megvalósítása	34
10.2. Java osztályok a Pi-ben	34
 IV. Irodalomjegyzék	 35
10.3. Általános	36
10.4. C	36
10.5. C++	36
10.6. Lisp	36

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 dblatex programmal a könyved pdf változatát, íme:

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

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



A DocBook XML 5.1 új neked?

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

**Bátf41 Haxor Stream**

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: <https://youtu.be/lvmi6tyz-nI>

Megoldás forrása:

```
0%
#include<stdio.h>
#include<unistd.h>
int main()
{
    for(;;)
    sleep(5);
    return 0;
}

100%
...
    while(1){}
...

Mind 100%
...
    #pragma omp parallel
    for(;;);
    return 0;
...
```

Fordítani pedig a `gcc prognev.c -fopenmp` paranccsal.

Tanulságok, tapasztalatok, magyarázat...

Ezek a programok könnyedén megérthetőek, ezáltal egyszerűen meg lehetett tudni például: miként is tudunk egyszerre több magot felhasználni egy művelethez, melynek akár a későbbiekben még hasznát vehetjük.

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

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

Megoldás videó:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{
    boolean Lefagy(Program P)
    {
```

```
    if(P-ben van végtelen ciklus)
        return true;
    else
        return false;
}

boolean Lefagy2(Program P)
{
    if(Lefagy(P))
        return true;
    else
        for(;;);
}

main(Input Q)
{
    Lefagy2(Q)
}
}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

Ahogy az az előző részben is megjegyezték, egy Lefagy függvényt nem lehet létrehozni, mert hát ha lefagy, abban az esetben nem tud egy üzenetet küldeni, amennyiben pedig nem fagy le alaptól értéktelen a program használata.

2.3. Változók értékének felcserélése

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

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk

Megoldás forrása:

```
#include <stdio.h>
int main()
{
```



```
int x,y;
scanf ("%d %d",&x,&y);
x=x^y;
y=x^y;
x=x^y;
printf("%d %d \n",x,y);
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Több módszer is létezik melynek segítségével segédváltozók és logikai kifejezések nélkül is felcserélhetünk két változót. Ezen módszerek közül az exorral való felcserélést adtam meg példának, de ugyanakkor kivonás-összeadással és szorzás-osztással is meg lehet oldani.

2.4. Labdapattogás

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

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

Megoldás forrása: If-ekkel <https://bhaxor.blog.hu/2018/08/28/labdapattogas>

If-ek nélkül <https://github.com/gergokinczel/unideb/blob/master/pat.c>

Tanulságok, tapasztalatok, magyarázat...

A labdapattogás mint olyan, tulajdonképpen csak a labda két koordinátájának változtatgatása attól függően, hogy milyen irányba kell haladjon a labda. Az if-ekkel történő megoldás során folyamatosan ellenőrizzük, hogy a labda elérte-e már a konzol egyik oldalát, majd aszerint változtatjuk a koordinátákat, hogy melyik oldalt érte el. A nem if-ekkel történő megoldásban pedig különböző matematikai műveletek segítségével tudjuk a labda röppályáját szabályozni.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU

Megoldás forrása:

```
#include <stdio.h>
int main()
{
    int hosz = 0, szo = 1;
    do
    {
```

```
        hossz++;  
    } while(szo <= 1);  
    printf("%d bites egy szo.\n", hossz);  
    return 0;  
}
```

Tanulságok, tapasztalatok, magyarázat...

A program shiftelés segítségével tudja meghatározni az int típus hosszát, mely hasznos lehet ha nagyon nagy számokkal dolgozunk, megtudván a típus határait.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása: <https://github.com/gergokinczel/unideb/blob/master/pagerank.c>

Tanulságok, tapasztalatok, magyarázat...

A Page Rank talán az egyik legfontosabb algoritmus, melyet egy keresőmotor használhat, így tehát még a Google is ezt az algoritmust használja. Főbb funkciója az oldalak "ranking"-je azaz rangsorolása/értékelése. Egy oldalnak annál nagyobb a rangja minél több értékes oldal mutat rá, ugyanis annál kevesebbet ér egy másik oldalra való mutató minél több kifelé hivatkozó link van az adott oldalon. Szóval ha egy oldalra több száz oldal mutat rá, de minden egyes rámutatás mellett az oldalakon több száz másik hivatkozás is van, ennek az oldalnak kisebb lesz a rangja, mint egy olyan oldalnak melyre csak pár száz oldal mutat, de minden egyes oldal mely rá mutat nem tartalmaz több hivatkozást.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó: <https://youtu.be/xbYhp9G6VqQ>

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

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben_a_monty_hall-paradoxon_kapcsan

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

Tanulságok, tapasztalatok, magyarázat...

Elsőre nekem is nehéz volt belátnom, hogy valóban érdemes-e váltani az eredeti döntés után és szkeptikus voltam a matematikai bebizonyításról. A szimuláció viszont rámutat ennek helyességére és el lehet ismerni, hogy a változtatás valóban megnöveli esélyeinket.

3. fejezet

Helló, Chomsky!

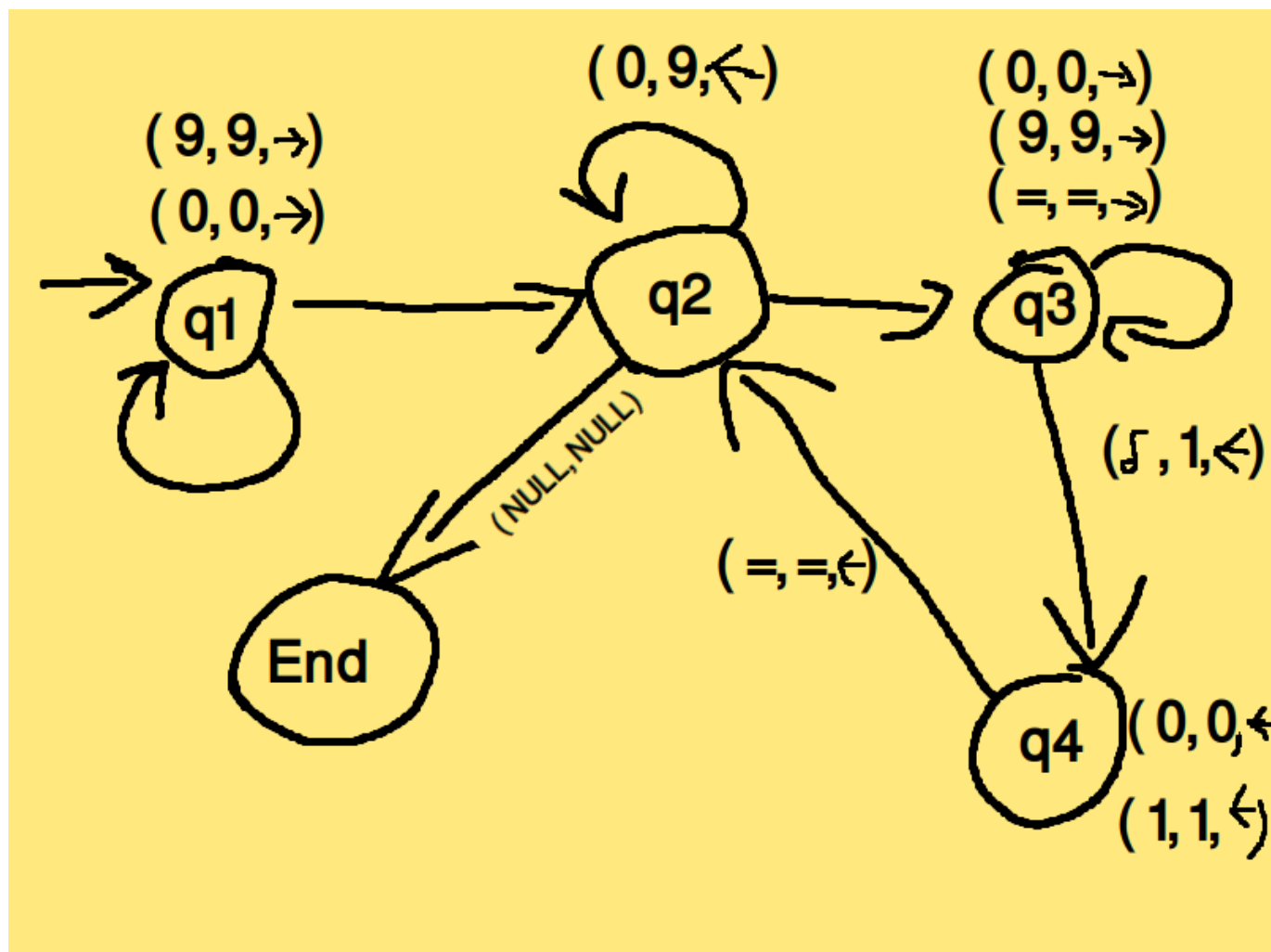
3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
int main() {
    int n;
    printf("Adjon meg egy szamot: ");
    scanf("%d", &n);
    for(int i=0; i<n; i++)
    {
        if(i%5==0 && i!=0) printf(" ");
        if(i%50==0)
            printf("\n");
        printf("I");
    }
    printf("\n");
    return 0;
}
```



Tanulságok, tapasztalatok, magyarázat...

A Turing gép egy matematikai modellje a számításnak ami definiál egy absztrakt gépet, mely szimbólumokat manipulál egy szalagon miközben több feltételt kielégít. A Turing gépet Alan Turing találta fel 1936-ban. Ő egy "a-machine"(automatic machine)-automata gépnek nevezte. Ezzel a modellel képes volt két esszenciális kérdésre válaszolni: létezik e egy olyan gép, mely meg tudja határozni, hogy az éppen elvégzésben levő feladat "körkörös"(pl.lefagyott, vagy nem tudja folytatni számításait); ugyanakkor létezik e egy olyan gép, amely meg tudja határozni, hogy a szalagján levő gép valaha is nyomtat e egy adott szimbólumot.

Ebben a példában decimálisból unárisba úgy váltjuk a számokat, hogy amennyi a decimális szám annyszor hajtunk végbe egy ismétlődő ciklust és annyi ugyanannyi darab 1 karaktert fogunk a képernyőre íratni ötössével elválasztva.

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:

S, X, Y legyenek változók

a, b, c legyenek konstansok

$S \rightarrow abc, S \rightarrow aXbc, Xb \rightarrow bX, Xc \rightarrow Ybcc, bY \rightarrow Yb, aY \rightarrow aaX, aY \leftrightarrow ->aa$

$S \ (S \rightarrow aXbc)$
 $aXbc \ (Xb \rightarrow bX)$
 $abXc \ (Xc \rightarrow Ybcc)$
 $abYbcc \ (bY \rightarrow Yb)$
 $AYbbcc \ (aY \rightarrow aa)$
 $aabbcc$

A, B, C legyenek változók
 a, b, c legyenek konstansok

$A \rightarrow aAB, A \rightarrow aC, CB \rightarrow bCc, cB \rightarrow Bc, C \rightarrow bc$

$A \ (A \rightarrow aAB)$
 $aAB \ (A \rightarrow aC)$
 $aaCB \ (CB \rightarrow bCc)$
 $aabCc \ (C \rightarrow bc)$
 $aabbcc$

Tanulságok, tapasztalatok, magyarázat...

A fent leírt generatív nyelvtan és Chomsky-hierarchia kidolgozása, Noam Chomsky amerikai nyelvész kutató nevéhez fűződik. Ezt a modellt formális nyelvek létrehozására használják. Az alapján működik, hogy a nyelv részit kifejezőerő szerint osztályozza, és az erősebb osztályok elemei a gyengébb osztályok elemeinek létrehozására képesek.

A generatív nyelvtannak négy fontos eleme van: terminális szimbólumok(konstansok), nem terminális jelek(változók), kezdőszimbólum(egy kitüntetett szimbólum 'S') és helyettesítési szabályok.

3.3. Hivatkozási nyelv

A [\[KERNIGHANRITCHIE\]](#) könyv C referencia-kézikönyv/Utasítások melléklete alapján definiáld BNF-ben a C utasítás fogalmát! Majd mutass be olyan kódcsipeteket, amelyek adott szabvánnyal nem fordulnak (például C89), mással (például C99) igen.

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
int main()
{
    for(int i=0; i<5; i++)
        printf("o");
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A következő módon néz ki a C utasítás fogalma ha BNF-ben definiáljuk:

```
<utasítás> ::=
    <címkézett utasítás>      ::= <azonosító> | "case" | "default"
    <kifejezésutasítás>      ::= <kifejezés>
    <összetett utasítás>     ::= <deklarációs lista> | <utasítás ←
        lista>
    <kiválasztó utasítás>    ::= "if" | "if else" | "switch"
    <iterációs utasítás>    ::= "while" | "do while" | "for"
    <vezérlésátadó utasítás> ::= "goto" | "continue" | "break" | ←
        "return"
```

Fentebb szerepel az a kódcsipet mely nem fordul le C89-es szabvánnyal, C99-el viszont igen, mivel változó deklarálása C89 es szabványba nem szerepelhetett a for ciklus argumentumai között.

3.4. Saját lexikális elemző

Írj olyan programot, ami számolja a bemenetén megjelenő valós számokat! Nem elfogadható olyan megoldás, amely maga olvassa betűnként a bemenetet, a feladat lényege, hogy lexert használjunk, azaz óriások vállán álljunk és ne kispályázzunk!

Megoldás videó: https://youtu.be/9KnMqrkj_kU

Megoldás forrása:

```
%{
#include <stdio.h>
int realnumbers = 0;
}%
digit [0-9]
%%
{digit}* (\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
    yylex ();
    printf("The number of real numbers is %d\n", realnumbers);
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A forráskód három fő részből áll, és a dupla % jelek választják őket külön.

- Az elsőben találjuk a definíciókat, ide olyan dolgokat tehetünk mint például a változók.

- A második részbe a szabályokat tesszük, megadjuk a regExp-t. Ebben a példában valós számokat keressünk ezért digit-et használunk a szabályba melyet megkülönböztet abban az esetben a pont van előtte, ilyen esetben tizedes szám lesz belőle.
- A harmadik részbe pedig felhasználói kód kerülhet, ilyen például a valós számok számának a kiírása.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

```
%{
#include <stdio.h>
%}
%%
[A-Za-z]+ {char betu[strlen(yytext)];
strcpy(betu,yytext);
int i;
for(i=0;i<strlen(yytext);i++){
if(betu[i]=='a' || betu[i]=='A')printf("4");
else if(betu[i]=='i' || betu[i]=='I' || betu[i]=='l' || betu[i]=='L')printf("1");
else if(betu[i]=='e' || betu[i]=='E')printf("3");
else if(betu[i]=='b' || betu[i]=='B')printf("8");
else if(betu[i]=='g' || betu[i]=='G')printf("9");
else if(betu[i]=='o' || betu[i]=='O')printf("0");
else if(betu[i]=='r' || betu[i]=='R' || betu[i]=='z' || betu[i]=='Z')printf("2");
else if(betu[i]=='s' || betu[i]=='S')printf("5");
else if(betu[i]=='t' || betu[i]=='T')printf("7");
else printf("%c",betu[i]);}
}
%%
int main()
{
yylex();
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Az előző feladat megoldása alapján lett létrehozva ez a l337 cipher, mely végig megy az inputon és kicserél minden egyes karaktert melynek létezik egy megfelelője a leet 48c-ből.

3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezelő)==SIG_IGN)
    signal(SIGINT, SIG_IGN);
```

Ha a SIGINT jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a jelkezelő függvény kezelje. (Miután a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)

**Bugok**

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

i.

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

ii.

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

Ismétlő utasítás mely ötször fog ismétlődni, nem jelent bug forrást.

iii.

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

Ismétlő utasítás mely ötször fog ismétlődni, nem jelent bug forrást.

iv.

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

**Bug**

Ismétlő utasítás mely ötször fog ismétlődni, viszont amennyiben a tömböt is szeretnénk használni ez egy jelentős bug forrást jelent.

v.

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

**Bug**

Tekintve, hogy a for ciklusban szereplő argumentum második fele nem egy logikai kifejezés, hanem egy pointer értékének a változtatása, ez egy bug forrást jelenthet.

vi.

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

**Bug**

A printf-ben szereplő f függvény visszatérési értékét írja ki a függvény kétszer, viszont nincs meghatározva C-ben, hogy az a értéke milyen sorrendben van növelve ezért ez egy jelentős bug forrás.

vii.

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

A programrészlet kiírja az f függvény által visszatérített értéket és az a változó értékét.

viii.

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

A `printf` kiírja az f függvény által visszatérített értéket megváltoztathatja az a értékét és kiírja az eredeti a értéket.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

1. Minden x -re létezik egy nála nagyobb y ami prím.
2. Minden x létezik egy annál nagyobb ikerprímszám.
3. Van olyan y amelynél minden x prím kisebb.
4. Van olyan y amelynél bármely nagyobb szám nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a=5;
```

- egészre mutató mutató

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

- egész referenciája

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

- egészek tömbje

```
int v[5]={1,2,3,4,5}
```

- egészek tömbjének referenciája (nem az első elemé)

```
int (&v)[3] = b;
```

- egészre mutató mutatók tömbje

```
int *v[5];
```

- egészre mutató mutatót visszaadó függvény

```
int *fuggveny();
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int *(*a)() = &f;
```

- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

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

- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

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

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó: <https://www.youtube.com/watch?v=1MRTuKwRsB0>

Megoldás forrása:

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int nr = 5;
    double **tm;
    printf("%p\n", &tm);
    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }
    printf("%p\n", tm);
    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL)
        {
            return -1;
        }
    }
    printf("%p\n", tm[0]);
    for (int i = 0; i < nr; ++i)
    for (int j = 0; j < i + 1; ++j)
        tm[i][j] = i * (i + 1) / 2 + j;
    for (int i = 0; i < nr; ++i)
    {
        for (int j = 0; j < i + 1; ++j)
            printf ("%f, ", tm[i][j]);
        printf ("\n");
    }
}
```

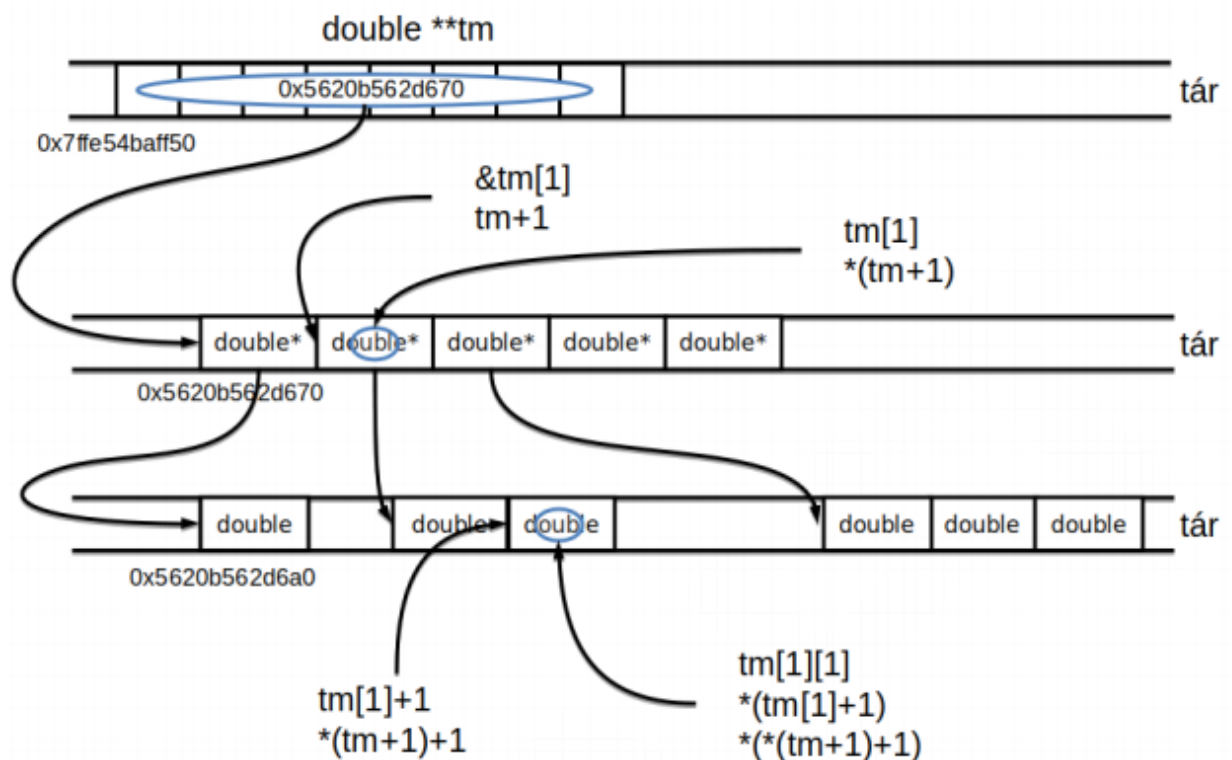
```

    for (int i = 0; i < nr; ++i)
    {
        for (int j = 0; j < i + 1; ++j)
            printf ("%f, ", tm[i][j]);
        printf ("\n");
    }
    for (int i = 0; i < nr; ++i)
        free (tm[i]);
    free (tm);
    return 0;
}

```

Tanulságok, tapasztalatok, magyarázat...

A program létrehoz egy `double **` típusú változót, amely egy tömb memóriacímére mutat, majd ennek a tömbnek minden eleme is egy-egy memóriacímre fog mutatni, még hozzá az adott elem sorszámanak megfelelő mennyiségű tömb elemre, ahogyan azt az alábbi ábra is reprezentálja. A `malloc` segítségével mindig annyi memóriacímet foglalhatunk le amennyire szükségünk van, jelen esetben amekkora a mérete az adott változó típusnak.



4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define MAX_KULCS 100
#define BUFFER_MERET 256

int main (int argc, char **argv)
{
    char kulcs[MAX_KULCS];
    char buffer[BUFFER_MERET];
    int kulcs_index = 0;
    int olvasott_bajtok = 0;
    int kulcs_meret = strlen (argv[1]);
    strncpy (kulcs, argv[1], MAX_KULCS);
    while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
    {
        for (int i = 0; i < olvasott_bajtok; ++i)
        {
            buffer[i] = buffer[i] ^ kulcs[kulcs_index];
            kulcs_index = (kulcs_index + 1) % kulcs_meret;
        }
        write (1, buffer, olvasott_bajtok);
    }
}
```

Tanulságok, tapasztalatok, magyarázat...

Az EXOR másnéven xor művelet bájtok lefedését, majd pedig a kizáró vagy művelet elvégzését az egymásra eső biteken jelenti, azaz, ha $a = 101$ és $b = 110$ akkor $a \oplus b = 011$. A fent leírt program az xor művelet segítségével titkosít egy szöveget, melyhez egy általunk megadott kulcsot használ. $\text{Eredeti szöveg} \oplus \text{kulcs} = \text{titkos szöveg}$ | $\text{Titkos szöveg} \oplus \text{kulcs} = \text{eredeti szöveg}$

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/ch01.html#exor_titkosito

Tanulságok, tapasztalatok, magyarázat...

A program lényege ugyanaz, mint az előző alcímnél C-ben, csak ezúttal Javában.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása: https://github.com/gergokinczel/unideb/blob/master/c_exor_toro.c

https://progater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia?fbclid=IwAR20XPCHTlyHUvM0GGA5adxYhVhw_dUNn7_8rb3_mQb4Seq4Kb4w

Tanulságok, tapasztalatok, magyarázat...

A C EXOR törő lényege, hogy megtalálja a kulcsot mely egy EXOR-ral titkosított magyar szöveget feltör és amely kulcsnak a mérete akár 8 karakter. Ez után pedig kiírja a kulcsot és a tiszta szöveget. A program úgy végzi a kulcs megtalálását, hogy minden lehetséges kulcs kombinációt kipróbál, majd a legmegfelelőbbet kiválasztja.

4.5. Neurális OR, AND és EXOR kapu

R

Megoldás videó: <https://youtu.be/Koyw6IH5ScQ>

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó: <https://www.youtube.com/watch?v=XpBnR31BRJY>

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

Tanulságok, tapasztalatok, magyarázat...

Azt a hálót, amelyben az összes bemenet közvetlenül a kimenetekre kapcsolódik egyrétegű neurális hálónak (single layer neural network) vagy perceptron (perceptron) hálónak nevezzük. Mivel mindegyik kimeneti egység független a többitől – mindegyik súly csak egyetlen kimenetre van hatással – vizsgálatainkat korlátozhatjuk az egykimenetű perceptronra.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

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

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezd0/elsocpp/mandelbrot/-mandelbrot.cpp#l1>

Tanulságok, tapasztalatok, magyarázat...

A Mandelbrot-halmaz egy síkbeli alakzat, amelyet egy alapvetően nagyon egyszerű algebrai összefüggés bonyolultabb (végtelennel kapcsolatos, analitikus fogalmakat, határértékszámítást igénylő) elemzése ad meg, rajzol ki. Ezeknek az összefüggéseknek a még legegyszerűbb (bár nem az egyetlen lehetséges) megközelítési módja a komplex számok felhasználásával történhet. A Mandelbrot-halmazt Benoît Mandelbrot fedezte fel, és Adrien Douady és John Hamal Hubbard nevezte el róla 1982-ben.

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

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

Megoldás forrása: https://github.com/gergokinczel/unideb/blob/Mandel/mandel_komplex.cpp

Tanulságok, tapasztalatok, magyarázat...

A program megközelítése ugyanaz, egyetlen különbség, hogy egy általunk készített struktúra helyett felhasználjuk a `complex` osztályt, melyben előre elkészített struktúrákat alkalmazunk.

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

A biomorfok biológikusan kinéző Pickover Szálak. Az 1980-as évek végén Clifford Pickover létrehozott úgynevezett biological feedback organizmusokat, hasonlókat a Júlia és Mandelbrot halmazokhoz. Pickover szerint egy biomorf egy olyan algoritmus, melyet változatos és összetett formák létrehozására lehet használni, melyek ugyanakkor gerinctelen életformákra hasonlítanak. Ezek az alakok bonyolultak és nehéz őket megjósolni mielőtt feltérképeznénk a kimenetelüket.

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

5.5. Mandelbrot nagyító és utazó C++ nyelven

Építs GUI-t a Mandelbrot algoritmusra, lehessen egérrel nagyítani egy területet, illetve egy pontot egérrel kiválasztva vizualizálja onnan a komplex iteráció bejárta z_n komplex számokat!

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

Megoldás forrása: https://github.com/gergokinczel/unideb/tree/Mandel/mandelbrot_rgb

Tanulságok, tapasztalatok, magyarázat...

A Mandelbrot nagyító a Mandelbrot halmaz létrehozását viszi egy lépéssel tovább. Minden egyes alkalommal, amikor ráközelítünk a kép egy részére, a program azt újrarajzolja "kinagyítja", de mivel a halmaz egy fraktál, ezt a végtelenségig meg tudjuk ismételni, elérve érdekesebbnél érdekesebb formákat.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: <https://youtu.be/Ui3B6IJnssY>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A lényeg ugyanaz mint C++ nyelven, ezúttal Javában megvalósítva.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

Megoldás videó: <https://bhaxor.blog.hu/2018/10/10/myrmecologist>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

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

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

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.

10. fejezet

Helló, Arroway!

10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

10.3. Általános

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

10.4. C

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

10.5. C++

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

10.6. Lisp

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