Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS			
	TITLE : Univerzális programozás		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Petrus, József Tamás	March 17, 2019	

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]

Contents

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 találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

Part I

Bevezetés

Chapter 1

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.

Part II

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.

Chapter 2

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:

Megoldás forrása: Ez a program egy magon végtelen ciklust futtat, viszont a top parancs alapbeállításaival 0.0%-os processzorhasználatot mutat.

```
#include <stdio.h>

int main()
{
    for(;;)
    {
        sleep(1);
        printf("Ez egy végtelen ciklus!\n");
    }
    return 0;
}
```

Ez a program egy magot terhel 100%-osan.

```
int main()
{
    while(1);
    return 0;
}
```

Ez a program pedig négy magot (a számítógépem négy magos processzorral rendelkezik) terhel 100%-osan.

```
#include <unistd.h>
```

```
int main()
{
    int t1,t2,t3;
    if(!(t1=fork()))
    {
        for(;;);
    }
    if(!(t2=fork()))
    {
        for(;;);
    }
    if(!(t3=fork()))
    {
        for(;;);
    }
    for(;;);
}
```

Tanulságok, tapasztalatok, magyarázat... Az egy magon 0%-os processzorhasználathoz az volt az elgondolás, hogy ha a ciklusmagban "elaltatjuk" a programot, akkor nem lesz kimutatható processzorhasználat. Az egy mag 100%-os leterhelését egy egyszerű "üres" végtelen ciklussal próbáltam először, ami sikeresnek bizonyult. A processzor összes magjának terhelését viszont már egyértelműen nem lehet egy egyetlen szálon futó programmal terhelni, ezért forkoltam három gyerek szálat, mind a háromban "üres" végtelen ciklussal, ami sikeresen terhelte a processzorom összes magját.

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 vlgtelen 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 épőlő Lefagy2 már nem tartalmaz feltételezett, csak 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.

Ez a feladat a megállási probléma bemutatására szolgál. A megállási probléma abból áll, hogy el lehet-e dönteni egy porgramról adott bemenet esetén, hogy végtelen ciklusba kerül-e. Alan Turing 1936-ban bizonyította be, hogy nem lehetséges olyan általános algoritmust írni, amely minden program-bemenet párról megmondja, hogy végtelen ciklusba kerül-e.

2.3 Változók értékének felcserélése

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

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

Megoldás forrása:

```
#include <stdio.h>

int main()
{
    int a=5,b=4;

    printf("a=%d, b=%d\n",a,b);
    int tmp = a;
    a=b;
    b=tmp;
    printf("a=%d,b=%d\n",a,b);
    return 0;
}
```

A megoldásomban azt a módszert alkalmaztam, amit először ismertem erre a feladatra. Ezt egykori informatika tanárom "bögrés cserének" hívta. Ezt úgy kell elképzelni, mintha a két változó egy bögre tej és egy bögre tea lenne, és úgy kell kicserélni a tartalmukat, hogy azok ne keveredjenek. A megoldás, hogy venni kell egy harmadik (segéd) bögrét, esetünkben változót, és abba beletölteni az első bögre tartalmát. Az első bögrébe beletölteni a második bögre tartalmát, és a második bögrébe beletölteni a harmadik (segéd) bögre tartalmát.

2.4 Labdapattogás

Először if-ekkel, majd bármiféle logikai utasítás vagy kifejezés nasználata nélkül írj 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:

https://github.com/Ignissen/pjt_bevprog/blob/master/pattogas_c.c

https://github.com/Ignissen/pjt_prog1/blob/master/pattogas_if.c

Ezt a feladatot először C++ nyelven készítettem el a Bevezetés a programozásba nevű kurzuson. Amikor atírtam a programot C-re, egyből szembetűntek a legalapvetőbb különbségek a nyelvek között. Például a standard kimenetre való írás módjának különbözősége. A program tartalmaz egy n*m-es karaktermátrixot, aminek a "szélső" elemei '#' karakterrel jelöltek, hogy láthatóak legyenek a "pálya" szélei. A labda '@' karakterrel van jelölve a képernyőn. Tartalmaz egy i int típusú számlálót, amely a program működéséhez szükséges. A program main függvényében található egy végtelen while ciklus, amelynek legelején a labda koordinátáinak kiszámítátása áll. Ezután a program lemásolja a program legelején létrehozott tömböt egy másik tömbbe. A másolás után meghívja a draw függvényt, amely egy kétdimenziós karaktertömböt, 2 int-et (labda x és y koordinátája), illetve egy karaktert, amely a labdát jelölő karakter. A karaktertömb másolására azért van szükség, mert a megoldásomban a `draw` függvényben a tömb azon elemét fölülírom a labda karakterével, amely koordinátája megegyezik a labdáéval. Ennek eredményeként, ha az eredeti tömböt adnám át a függvénynek, idővel az egész "pálya" betelne labdával. A draw függvény hívása után a számlálót növelem eggyel, illetve várakoztatom a program végrehajtását 50 ms-al, hogy a labda mozgása szemmel követhető legyen.

2.5 Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása: https://github.com/Ignissen/pjt_prog1/blob/master/szohossz.c

A legtöbb személyi számítógép esetében az int mérete 32 bit. Ez azt jelenti, hogy egy int típusú változó -2,147,483,648 és +2,147,483,647 közötti számokat képes tárolni. Az általam tesztelt számítógépen 32 bites az int mérete. Ez valószínűleg a legtöbb személyi számítógépre igaz.

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/Ignissen/pjt_prog1/blob/master/pagerank.c

A PageRank egy olyan algoritmus, amellyel weboldalak relatív fontosságát lehet megállapítani. Azt adja meg, hogy véletlenszerű böngészés esetén mekkora az esélye annak, hogy az adott oldalra találunk. Alapja, hogy egy oldalon minden hivatkozás egy-egy "szavazat" a hivatkozott oldalra. Az alapján meg lehet állapítani egy oldal relatív fontosságát, hogy hány az oldalra mutató hivatkozás van a többi oldalon, illetve, hogy hány oldalra hivatkozik az adott oldal. Az algoritmusban egy jobb minőségű oldal "szavazata" erősebbnek számít, mint egy kis relatív fontosságúé. Mivel a felhasználó általában nem fogja végignézni az összes linket a weboldalon, ezért bevezettek a képletbe egy csillapító faktort. A megoldásomban Lovász Botond segített.

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

A Brun-tétel szerint az ikerprímek reciprokösszege egy meghatározható szám felé tart, amelyet a B_2 konstanssal jelölnek, amely értéke $B_2 \approx 1{,}902160583104$. Az ikerprím olyan két egymást követő prímszámot jelent, amelyek különbsége 2.

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

A Monty Hall probléma egy az Amerikai Egyesült Államokban sugárzott televíziós vetélkedő játékszabályai alapján jött létre. A szabály az, hogy a versenyző kiválaszt 3 ajtó közül egyet, amelyek mögött vagy kecske van (2 ajtó esetében) vagy egy autó van az ajtó mögött (1 ajtó mögött van csak autó). A játékvezető, aki tudja, melyik ajtó mögött mi található, kinyit egy ajtót, amelyet a versenyző nem választott, majd megkérdezi, hogy szándékszik-e ajtót váltani. Ezután, ha a versenyző nem váltott ajtót, akkor a játékvezető kinyitja a második nem választott ajtót. Ha a versenyző ajtót váltott, akkor az eredetileg választott ajtót nyitja ki a játékvezető.

Az a probléma azért paradoxon, mert a válasz arra, hogy megéri-e váltani az ajtók között a játékvezető kérdése után, az, hogy igen, érdemes változtatni az ajtón, viszont ez a józan észnek annyira ellentmond, hogy ezt a problémát paradoxonnak minősítik.

A probléma megoldása azon alapszik, hogy, maikor választunk a három ajtó közül, akkor 1/3 az esélyünk arra, hogy a választott ajtó mögött található az autó. A játékvezető először mindenképp kecskét rejtő ajtót nyit ki a játékosnak. Az eslő ajtó választásánál 2/3 az esélyünk arra, hogy kecskét választunk, ez alapján a játékvezető kénytelen a másik kecskét rejtő ajtót kinyitni. Ez alapján látható, hogy ha váltunk az első ajtó kinyitása előtt, javul az esélye annak, hogy a játékos autót nyer.

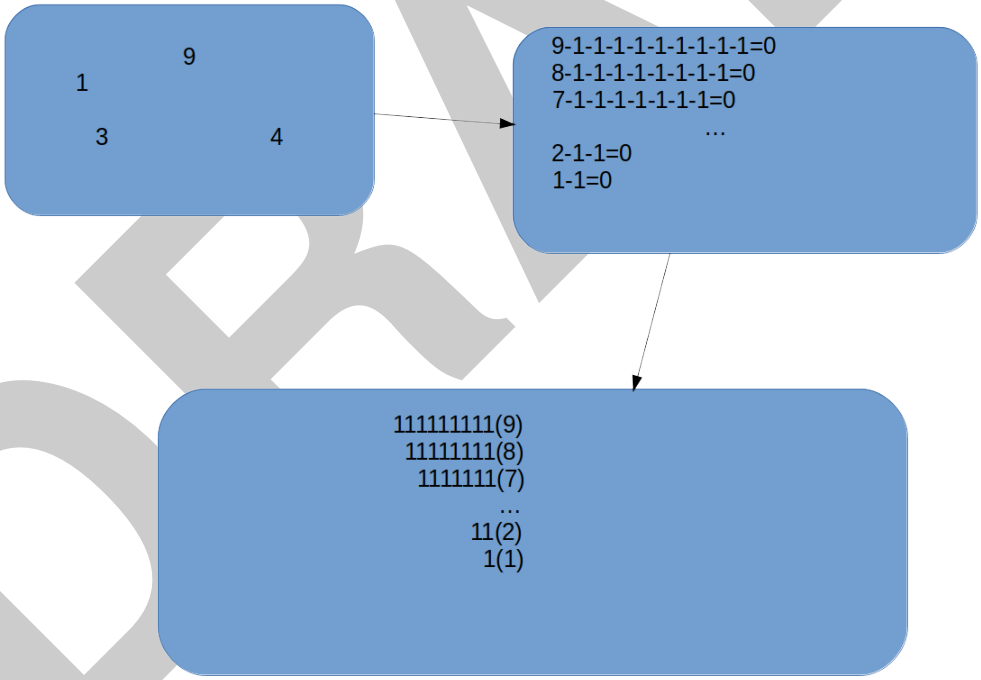
Chapter 3

Helló, Chomsky!

3.1 Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:



Az unáris számrendszerben való ábrázolás n darab (n a decimális szám) egyforma jel, karakter egymás utáni leírásával történik.

A decimálisból unárisba átváltás úgy történik, hogy folyamatosan 1-eket vonunk ki a számból, és tároljuk a levont egyeseket.

3.2 Az $a^n b^n c^n$ nyelv nem környezetfüggetlen

Mutass be legalább két környezetfüggő generatív grammatikát, amely ezt a nyelvet generálja!

Megoldás videó:

Megoldás:

Első környezetfüggő generatív grammatika:

```
Szabályok:
S -> aBSc
S -> abc
Ba -> aB
Bb -> bb

Példa levezetés:
S -> aBSc -> aBaBScc -> aBaBabccc -> aaBBabccc -> aaBaBbccc -> aaaBBbccc -> ↩
    aaaBbbccc -> aaabbbccc
```

Második környezetfüggő generatív grammatika:

```
Szabályok:
S -> abc
S -> aXbc
Xb -> bX
Xc -> Ybcc
bY -> Yb
aY -> aaX
aY -> aa

Példa levezetés:
S -> aXbc -> abXc -> abYbcc -> aYbbcc -> aaXbbcc -> aabXbcc -> aabbXcc ->  ↩
   aabbYbccc -> aabYbbccc -> aaYbbbccc -> aaabbbccc
```

A generatív nyelvtan elméletét Noam Chomsky alkotta meg, és ő dolgozta ki a Chomsky-hierarchiát. A formális grammatikáknak három típusa van, a környezetfüggetlen nyelvtan, a szabályos nyelvtan és a generatív nyelvtan. A környezetfüggetlen nyelvtanban a szabályok megadása esetén a szabály bal oldalán csak nem terminális változó állhat, illetve a jobb oldalán csak terminális változók állhatnak.

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: https://github.com/Ignissen/pjt_prog1/blob/master/utasitasok.txt

```c
#include <stdio.h>

int main()
{
    for(int i=0;i<5;i++)
    {
        printf("%d\n",i);
    }
    return 0;
}
```

A Backus-Naur-forma a különböző nyelvek egyik lehetséges leírási módszere. Ezt a leírási módszert John Backus hozta létre, eredetileg az ALGOL programozási nyelvhez. Azóta már a legtöbb programozási nyelv szintaxisát BNF-ben adják meg, illetve természetes leírásához is használják alkalmanként. Peter Naur egyszerűsítette le a leírási módszert, ezért Donald Knuth javaslatára Naur neve is belekerült a leírási módszer megnevezésébe.

Feladat volt még olyan C programot írni, amely egyes nyelvi szabvánnyal, jelen esetben a C89-es szabvánnyal, nem fordul le, míg például a C99-es szabvánnyal már igen. A C89-es szabvány például még nem engedte a for ciklusban a ciklusfejben történő ciklusváltozó deklarálását. Ezt szemlélteti a fenti kód is, ugyanis a -std=c89 kapcsolót használva hibát jelez a fordító.

3.4 Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

```
%{
#include <stdio.h>
int szamok=0;
%}
%option noyywrap
%%
[[:digit:]]+  {
            szamok++;
        }

[[:alpha:]][[:print:]]*    {   /* Nem csinálunk semmit. */   }
%%
int main(void)
{
    yylex();
    printf("%d számot talált a lexer: \n",szamok);
```

```
    return 0;
}
```

Az lex fájl első részében, amit a %{...%} jelöl, jelzem a fordítónak, hogy az stdio.h függvénykönyvtárat szeretném haszálni az stdoutra való íráshoz, majd létrehozok egy számlálót, ami a lexer által észlelt számok darabszámát fogja tárolni. A második részben, amit a %%...%% jelöl, megadtam a lexernek, hogy bizonyos mintákra hogyan reagáljon. A [[:digit:]]+ azt jelenti, hogy legalább egy számjegy egymás után. Ha legalább egy számjegyet talál egymás után, abban az esetben eggyel növeli a számlálót. Ezután megadtam, hogy bármilyen más minta eseté ben, effektíve ne csináljon semmit. Majd az utolsó részben, amely már nincs külön jelölve, a `main` függvényben meghívjuk a `yylex` függvényt, amely meghívja magát a lexert, amely végigfutja bájtonként a bemenetet. Ha a lexer futása véget írt, kiíratom a számláló értékét. Majd a return 0-val jelzi az operációs rendszer felé, hogy a program futása véget ért.

3.5 l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása: https://github.com/Ignissen/pjt_prog1/blob/master/leet.c

A lex első részében a program által látható függvénykönyvtárak include-jai láthatók. Ezután egy int típusú változó létrehozása áll, amely a random számok generálásának beállításához szükséges. A program működésének alapja a cipher típusú tömb létrehozása, ami a különböző betűkhöz és számokhoz tartozó lehetséges leet kódokat tartalmazza, többnyire három kódolt betűt és 4 "eredeti" betűt, hogy kisebb eséllyel legyen minden betű átalakítva.

A kód következő részében minden a lexer által beolvasott karakterre megnézi a program, hogy benne van-e a cipher típusú tömb kódolandó karakterei között. Ha megtalálja, akkor ahhoz a karakterhez tartozó egyik kódolást véletlenszerűen kiválasztja, majd a standard kimenetre kiírja. Ha nem találta meg, akkor az eredeti karaktert kiírja a standard kimenetre.

A `main` függvényben a program meghívja a `yylex` függvényt, azaz magát a lexert. Ha a lexer futása véget ér, akkor a program 0-val tér vissza, amely azt jelzi az operációs rendszernek, hogy a program futása sikeresen véget ért.

3.6 A források olvasása

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

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

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

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

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

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

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

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

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

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

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

Megoldás forrása:

```c
#include <stdio.h>
#include <signal.h>

void jelkez()
{
    printf("\nNem nyert.\n");
}

int main()
{
    while(1)
    {
        if(signal(SIGINT, jelkez)==SIG_IGN)
            signal(SIGINT, SIG_IGN);
    }
    return 0;
}
```

A feladatban leírt kódcsipetet természetes nyelven valahogy így lehet elképzelni: Ha a program elkap egy megszakítás jelet, akkor meghívja a `jelkezelo` függvényt/eljárást, amely valamit csinál a megszakításra, ezután pedig visszajelzi az operációs rendszernek, hogy a program kezelte a jelet.

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

Ez a csipet nem fog a célnak megfeleően működni, mert a manual signal lapja alapján a signal függvény a második argumentumával fog visszatérni, amelynek az if-ben saját magával kéne nem egyenlőnek lennie, ezért a jelkezelő függvény soha nem lesz meghívva.

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

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

Ez a két csipet szintaktikailag különbözik abba, hogy az elsőnél először növelnénk az i értéket, majd használnánk az értékét, de eppen a példában nem használjuk az értékét egyből, tehát nem számít. A két csipet szemantikailag azonos, az előbb említett ok miatt, minden esetben hibátlanul fog lefutni.

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

Ez a csipet egy léptetős ciklus, amely ötször fog lefutni, és minden iterációban a tomb nevű tömb i-edig elemét egyenlővé teszi i-vel. Ez a csipet nem minden esetben fog hibátlanul lefutni, mert ez a csipet feltételezi, hogy a tomb nevű tömb legalább 5 elemű, illetve a ciklus előtt deklarálva és inicializálva van egy i nevű ciklusváltozó. Egyéb esetben a program olyan memóriaterületre fog hivatkozni, ami számára nem megengedett, ez pedig az (modern) operációs rendszer általi azonnali leállítást eredményez.

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

Ez a csipet minden estben le fog fordulni, azonban ha az n változó nagyobb, mint a d vagy s tömb hossza, abban az esetben az előző csipetnél leírt probléma fog előfordulni.

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

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

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

Ezek a csipetek uyganazt csinálják, csak kicsit másképp. A csipetek kiírnak standard inputra két egészet. Ezen csipeteknek közös lehetséges bugforrása, hogy az argumentumok sorrendjétől nem független a kiértékelés.

3.7 Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Az első formula természetes nyelven: Minden x-re létezik olyan y, hogy x kisebb, mint y, és y prím. Tehát: Minden számnál létezik nagyobb prímszám.

A második formula természetes nyelven: Minden x-re létezik olyan y, hogy x kisebb, mint y, y prím és y rákövetkezőjének rákövetkezője is prím. Tehát: Minden számnál léteznek nagyobb ikerprímek.

A harmadik formula természetes nyelven: Létezik olyan y, hogy minden x-re igaz, hogy ha x prím, akkor x kisebb, mint y. Tehát: Minden prímszámra igaz, hogy létezik tőle nagyobb szám.

A negyedik formula természetes nyelven: Létezik olyan y, hogy minden x-re igaz, hogy ha y kisebb, mint x, akkor x nem prím. Tehát: Létezik olyan szám, amelytől nem létezik kisebb prímszám.

3.8 Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
  int a;  //egész típusú változó létrehozása
  ```
- ```
  int *b = &a;  //egészre mutató mutató
  ```
- ```
  int &r = a;  //egész referenciája
  ```
  ```

- ```
  int c[5];  //egészek tömbje
  ```
- ```
  int (&tr)[5] = c;  //egészek tömbjének referenciája
  ```
- ```
  int *d[5];  //egészre mutató mutatók tömbje
  ```
- ```
  int *h ();  //egészre mutató mutatót visszaadó függvény
  ```
- ```
  int *(*l) ();  //egészre mutató mutatót visszaadó függvényre mutató ↩
     mutató
  ```
- ```
  int (*v (int c)) (int a, int b)  //egészet visszaadó és két egészet kapó ↩
     függvényre mutató mutatót visszaadó, egészet kapó függvény
  ```
- ```
  int (*(*z) (int)) (int, int);  //függvénymutató egy egészet visszaadó és ↩
     két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó ↩
     függvényre
  ```

Megoldás videó:

Megoldás forrása:

```
                #include <stdio.h>
#include <stdlib.h>

int main()
{
    //Első csipet
    int a=5;
    //Második csipet
    int *b = &a;
    *b=2;
    printf("a=%d\n",a);
    //Harmadik csipet
    int &r=a;
    r=4;
    printf("a=%d\n\n",a);
    return 0;
}
```

Ez a program gcc-vel nem fordul, mert a c-ben nincs referencia típus. Az első csipetben létrehoztam egy int típusú változót. A második csipetben létrehoztam egy egészre mutató mutatót, amely az első egészre mutat. A harmadik csipetben létrehoztam egy egész referenciáját, amely értéke az a-val egyenlő. A referencia típus azt valahogy úgy kell elképzelni, mint linuxon a hardlinkeket.
```

```c
#include <stdio.h>

int main()
    //Negyedik csipet
    int c[5]={1,2,3,4,5};
    //Ötödik csipet
    int (&tr)[5] = c;
    for(int i=0;i<5;i++)
    {
        printf("%d\n",tr[i]);
    }
    //Hatodik csipet
    int *d[5];
    for(int i=0;i<5;i++)
    {
        printf("%p\n",d[i]);
    }
    return 0;
}
```

Ez a program gcc-vel nem fordul, mert a c-ben nincs referencia típus. A negyedik csipetben létrehoztam egy egész típusú tömböt, amely elemei rendre 1, 2, 3, 4, 5. Az ötödik csipetben létrehoztam egy egészek tömbjének referenciáját. A hatodik csipetben létrehoztam egy egészekre mutató mutatók tömbjét, amely 5 elemű.

```c
#include <stdio.h>
#include <stdlib.h>
//Hetedik csipet
int *h()
{
    return (int*) malloc(sizeof(int));
}

int main()
{
    //Nyolcadik csipet
    int *(*l)() = h;

    printf("%p\n",l());
    return 0;
}
```

A hetedik csipetben létrehoztam egy egészre mutató mutatót visszaadó függvényt. A nyolcadik csipetben létrehoztam egy, az előző csipetben létrehozott függvényre mutató mutatót. Ezután egy `printf` függvénnyel standard outpura kiírattam a függvény pointer segítségével a `h` függvény által lefoglalt egész memóriacímét.

```c
#include <stdio.h>
```

```c
#include <stdlib.h>

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

int (*asd(int c))()
{
    if(c) return sum;
    else return mul;
}
///Kilencedik csipet
int (*v(int c))(int a,int b)
{
    return asd(c);
}



int main()
{
    //Tizedik csipet
    int *(*z(int))(int,int) = v;
    printf("%d\n",z(0)(5,5));
    return 0;
}
```

A kilencedi csipetben létrehoztam egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényt. A tizedik csipetben létrehoztam egy, az előző feladatben létrehozott függvényre mutató mutatót.

Chapter 4

Helló, Caesar!

4.1 double ** háromszögmátrix

Megoldás videó:

Megoldás forrása: https://github.com/Ignissen/pjt_prog1/blob/master/haromszog_matrix.c

A programban helyet foglalok a memóriában egy double-öket tartalmazó alsó háromszögmátrixnak. Majd k=0-tól a mátrix minden elemének 1.1-es lépésközzel értékül adtam k-t. Ezután a mátrix standard uotputra való kiíratása történik. Következőnek felszabadítom a pointerek által lefoglalt memóriacímeket, majd a "return 0"-val jelzem az operációs rendszer felé, hogy a program futása hiba nélkül befejeződött.

4.2 C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: https://github.com/Ignissen/pjt_prog1/blob/master/exor.c

Ez a program a legelején megnézi, hogy a kapott argumentumok száma több, mint kettő. Erre azért van szükség, mert a kódolandó szövegfájl nevét és a titkosításhoz használt kulcsot parancssori argumentumként kapja meg a program. Ha a feltétel teljesül, tovább fut a program, ha nem, akkor kiírja standard outputra a program helyes használatának módját. Ezután az igaz ágon belül megnyitjuk olvasásra a kódolandó szöveget tartalmazó fájlt. Ha a fájl megnyitása nem sikerült, hibaüzenettel kilép a program. Ha sikerült, akkor karakterenként beolvassuk, a kulcs megfelelő karakterével "össze-exorozzuk", majd kiíratjuk standard outputra. Ha végzett a beolvasással, akkor a tiszta szöveget tartalmazó fájlt bezárjuk, majd kilép és jelzi az operációs rendszer felé, hogy a program futása hiba nélkül véget ér.

4.3 Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása: https://github.com/Ignissen/pjt_prog1/blob/master/exor.java

Ez a program az előző feladat megoldásának java átirata, amely annyival különbözik az előzőtől, hogy itt a tiszta szöveget tartalmazó fájl neve tiszta.txt, amely egy mappában van a programmal, iletve a kulcsot a standard inputról kéri be, nem parancssori argumentumokként.

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/Ignissen/pjt_prog1/blob/master/t.c

Ez a program több függvényből is áll. A `atlagos_szohossz` függvény a program által éppen generált tesztkulcscsal visszafejtett szöveg átlagos szóhosszát adja vissza.

A `tiszta_lehet` függvény azt adja meg, hogy a tesztkulcscsal visszafejtett szöveg lehet-e a tényleges visszafejtett szöveg. Ennek az a feltétele, hogy az átlagos szóhossz az hat és kilenc között legyen, illetve tartalmazza a következő négy szót: "hogy", "az", "nem", "ha".

Az `exor` függvény hajtja végre a titkos szöveg a program által generált kulcsokkal történő visszafejtését, amely vagy a tényleges visszafejtés, vagy nem.

Az `exor_tores` függvény meghívja az `exor` függvényt, majd a `tiszta_lehet` függvényt, amely, ha igaz értékkel tér vissza, akkor kiírja az adott kulcsot, illetve a kulcscsal visszafejtett szöveget.

Végül a `main` függvényben beolvassuk a titkosított szöveget, majd nyolc egymásbaágyazott for ciklussal végig vizsgáljuk az összes kulcsot. Ezt a lépést Bátfai Norbert Párhuzamos programozás GNU/Linux környezetben c. könyvének linkelt fejezete alapján OpenMP használatával párhuzamossá tettem.

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

Ebben a feladatban a cél egy olyan neurális háló létrehozása és tanítása, amely az egyszerű logikai műveletek elvégzésére képes.

Ez a kód igazából négy kis eltéréssel ismétlődő részből áll. Minden részben lényegében ugyanaz történik, egyedül a logikai művelet változik, amelyre feltanítjuk a neurális hálót. Ezalól kivétel az utolsó, amiről lentebb szó lesz.

Első rész:

```
a1     <- c(0,1,0,1)
a2     <- c(0,0,1,1)
```

```
OR    <- c(0,1,1,1)

or.data <- data.frame(a1, a2, OR)

nn.or <- neuralnet(OR~a1+a2, or.data, hidden=0, linear.output=FALSE,  ↩
   stepmax = 1e+07, threshold = 0.000001)

plot(nn.or)

compute(nn.or, or.data[,1:2])
```

Az elején megadjuk, hogy milyen bementi adatokból milyen eredményt kell megközelítenie a threshold-dal jelölt hibahatáron belül. Ezután ezt megadjuk a neurális hálónak is, majd a neurális hálót feltanítjuk a feladatra. Itt meghívjuk a `neuralnet` függvényt, amely megkapja a bementi adatokat és az elvárt kimeneteket, 0 rejtett réteggel, 0.000001-es hibahatárral. Ezután a `plot` függvénnyel kirajzoljuk a neurális háló sematikus képét egy gráf segítségével.

Majd a `compute` függvénnyel meghívjuk a már feltanított neurális hálót az elején megadott adatokkal, hogy kiszámolja a logikai műveletek eredményét.

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ↩
   output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

compute(nn.exor, exor.data[,1:2])
```

Itt annyi különbség van, hogy míg a harmadik részben feltanított neurális háló kb. 50%-os pontossággal dolgozott, ami annyit jelent, mintha véletlenszeűen találgatott volna, itt már több neuron van a hálóban, növelve a pontosságot. Ebben az esetben három rejtett neuronréteg van, amelyek rendre 6, 4, 6 neuronból állnak. Ezeket a "hidden=c(6,4,6)" argumentum jelöli. Ezzel már a hibahatáron belülre kerül többnyire az EXOR logikai művelet értékének kiszámítása.

4.6 Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása: https://github.com/Ignissen/pjt_prog1/blob/master/perc.cpp

A program a kód elején vizsgálja, hogy a parancssori argumentumok száma kettő-e. Erre azért van szükség, mert az bemenetként szolgáló png fájl nevét parancssori argumentumként kapja meg a program. Ezután a png++ függvénykönyvtár segítségével beolvassa a bementi fájlt. Ezután létrehoz egy perceptront 4 rétegű neurális hálóval, amelynek neuron száma sorra 3, a kép pixeleinek száma, 256, 1. Majd létrehoz egy dinamikusan foglalt a kép pixeleinek megfelelő számosságú double tömböt, amelybe belemásolja a kép minden pixelének vörös értékét. Ezután meghívja a perceptront, hogy dolgozza fel a képet. A progvam végén felszabadítja a pointerek által foglalt memóriát, majd kilép.

Chapter 5

Helló, Mandelbrot!

5.1 A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

5.3 Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

5.4 A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

5.5 Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

5.6 Mandelbrot nagyító és utazó Java nyelven

Chapter 6

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 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:

Chapter 7

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...

Chapter 8

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...

Chapter 9

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:

Chapter 10

Helló, Gutenberg!

10.1 Programozási alapfogalmak

[?]

10.2 Programozás bevezetés

[KERNIGHANRITCHIE]

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

10.3 Programozás

[BMECPP]

Part III

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.

Chapter 11

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...

Part IV

Irodalomjegyzék

11.3 Általános

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

11.4 C

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

11.5 C++

[BMECPP] Benedek, Zoltán Ã©s 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.