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	Bátfai, Norbert	2019. március 16.	

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? . . . 7
2.3. Változók értékének felcserélése . . . 9
2.4. Labdapattogás . . . 10
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 11
2.6. Helló, Google! . . . 14
2.7. 100 éves a Brun tétel . . . 15
2.8. A Monty Hall probléma . . . 17

3. Helló, Chomsky! **20**
3.1. Decimálisból unárisba átváltó Turing gép . . . 20
3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 20
3.3. Hivatkozási nyelv . . . 21
3.4. Saját lexikális elemző . . . 22
3.5. l33t.l . . . 23
3.6. A források olvasása . . . 23
3.7. Logikus . . . 26
3.8. Deklaráció . . . 28

4. Helló, Caesar! 31
4.1. double ** háromszögmátrix . . . 31
4.2. C EXOR titkosító . . . 33
4.3. Java EXOR titkosító . . . 35
4.4. C EXOR törő . . . 36
4.5. Neurális OR, AND és EXOR kapu . . . 38
4.6. Hiba-visszaterjesztéses perceptron . . . 42

5. Helló, Mandelbrot! 44
5.1. A Mandelbrot halmaz . . . 44
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 44
5.3. Biomorfok . . . 44
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 44
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 44
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 45

6. Helló, Welch! 46
6.1. Első osztályom . . . 46
6.2. LZW . . . 46
6.3. Fabejárás . . . 46
6.4. Tag a gyökér . . . 46
6.5. Mutató a gyökér . . . 47
6.6. Mozgató szemantika . . . 47

7. Helló, Conway! 48
7.1. Hangyaszimulációk . . . 48
7.2. Java életjáték . . . 48
7.3. Qt C++ életjáték . . . 48
7.4. BrainB Benchmark . . . 49

8. Helló, Schwarzenegger! 50
8.1. Szoftmax Py MNIST . . . 50
8.2. Szoftmax R MNIST . . . 50
8.3. Mély MNIST . . . 50
8.4. Deep dream . . . 50
8.5. Robotpszichológia . . . 51

9. Helló, Chaitin! 52
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 52
9.2. Weizenbaum Eliza programja . . . 52
9.3. Gimp Scheme Script-fu: króm effekt . . . 52
9.4. Gimp Scheme Script-fu: név mandala . . . 52
9.5. Lambda . . . 53
9.6. Omega . . . 53

10. Helló, Gutenberg! 54
10.1. Programozási alapfogalmak . . . 54
10.2. Programozás bevezetés . . . 55
10.3. Programozás . . . 55

III. Második felvonás 56

11. Helló, Arroway! 58
11.1. A BPP algoritmus Java megvalósítása . . . 58
11.2. Java osztályok a Pi-ben . . . 58

IV. Irodalomjegyzék 59
11.3. Általános . . . 60
11.4. C . . . 60
11.5. C++ . . . 60
11.6. Lisp . . . 60

Ábrák jegyzéke

2.1. A B_2 konstans közelítése 17

4.1. A double ** háromszögmátrix a memóriában 31

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.

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:

Megoldás forrása: https://github.com/p-adrian05/BHAX/tree/master/textbook_programs/Turing/vegtelen_ciklus

Ebben az esetben a következő végtelen ciklus 100%-ban dolgoztat egy magot:

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

A program fordítása és indítása után a **top -p `pgrep -u adrian loop1`** parancs beírásakor láthatjuk a várt eredményt.

```
Tasks: 1 total, 1 running, 0 sleeping,   0 stopped,   0 zombie
%Cpu0: 1,3 us, 0,3 sy, 0,0 ni, 98,0 id, 0,0 wa, 0,0 hi, 0,3 si, 0,0 st
%Cpu1: 0,3 us, 0,3 sy, 0,0 ni, 99,3 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2: 100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3: 1,4 us, 0,0 sy, 0,0 ni, 98,6 id, 0,0 wa,  0,0 hi, 0,0 si, 0,0 st
KiB Mem: 8025272 total, 4308952 free, 1653004 used, 2063316 buff/cache
KiB Swap: 3897340 total, 3897340 free, 0 used. 5653664 avail Mem

PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
4134 adrian    20   0    4376    756    692 R 100,0  0,0   1:14.23 loop1
```

0% terhelés eléréséhez meg kell hívnunk a `sleep(n)` metódust ami lelassítja az iterálás sebességét.

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

```
 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4515 adrian    20   0    4376    816    752 S   0,0  0,0   0:00.00 loop0
```

Minden mag 100% terhelését szálasítással lehet megoldani az OpenMP segítségével.

```
#include <omp.h>

    int main()
    {
        #pragma omp parallel
        {
            while(1){}
        }
        return 0;

    }
```

gcc loop_all_cores.c -o loop -fopenmp parancsal fordítjuk le ebben az esetben.

```
  Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie
%Cpu0:100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1:100,0 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2:100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3:100,0 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st


 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4914 adrian    20   0   35576   1024    928 R 388,7  0,0   8:00.85 loop
```

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(;;);
  }

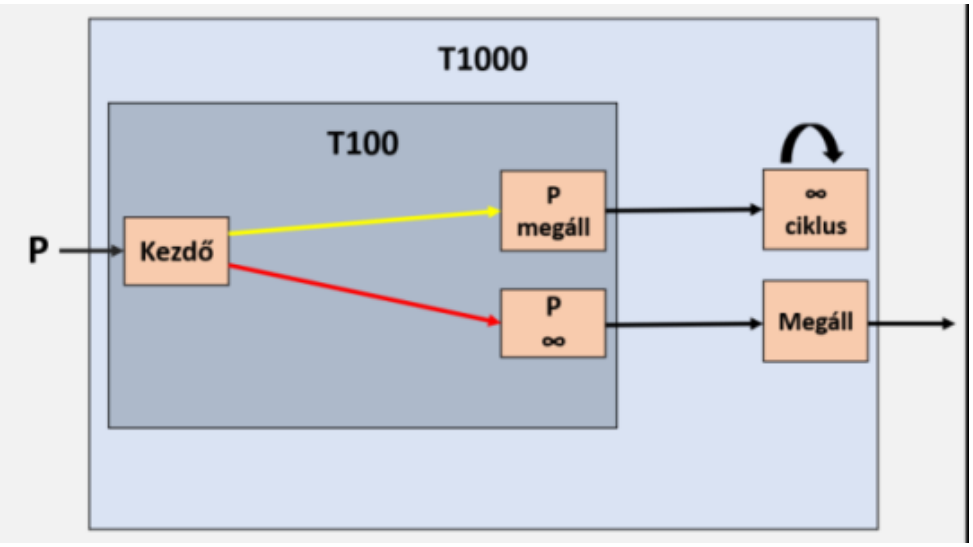
  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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.



Tegyük fel, hogy létezik olyan program (T100), ami el tudja dönteni egy másik programról, hogy van-e benne végtelen ciklus. Ha van, akkor megáll a program, ha nincs, akkor pedig végtelen ciklusba kezd. Létrehozunk egy új programot (T1000) az előzőt (T100) felhasználva és ha a T100 megállt, akkor végtelen ciklusba kezd, ha pedig a T100 kezdett végtelen ciklusba, akkor megáll.

Mi történik, ha magát etetjük meg a programunkkal?

A megállás csak akkor lehetséges, ha a T100 nem áll meg, de ez pedig csak akkor lehet, ha a második argumentumként kapott saját programunk megáll.

Ebből ellentmondásra jutottunk, tehát nem lehet ilyen programot írni.

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: https://github.com/p-adrian05/BHAX/blob/master/textbook_programs/Turing/val_csere.c

Két változó értékének felcserélésének legegyszerűbb módja, amikor segédváltozót használunk, amiben ideiglenesen eltároljuk az első változó értékét, majd az elsőt egyenlővé tesszük a másodikkal és a másodikat a segédváltozóval. Azonban léteznek módszerek segédváltozó nélküli cserére is.

Ebben az esetben segédváltozó nélkül cseréljük meg két változó értékét a különbségük kihasználásával.

```
#include<stdio.h>

int main()
{
    int a = 3;
    int b = 8;

    //segedvaltozo nelkül
    b = b-a;
    a = a+b;
    b = a-b;
    printf("a=%d b=%d\n",a,b);
```

A következő módszer pedig az EXOR-os csere.

```
#include<stdio.h>

int main()
{
    int a = 3;
    int b = 8;

    //exorra
    //kettes számrendszerben:
    //a = 8-> 0001
    //b = 3-> 1100

    a = a^b; //1101
    b = a^b; //0001
    a = a^b; //1100

    printf("a=%d b=%d\n",a,b);
```

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/p-adrian05/BHAX/tree/master/textbook_programs/Turing/labda

A labdapattogás nevű feladat klasszikus megoldása feltételvizsgálatokkal történik, ahol nyilvántartjuk a terminál méretét (fordításkor **-lncurses** könyvtár használatával), a labda koordinátáinak kezdőértékét és lépések mértékét. Ezután if feltételvizsgálattal nézzük, hogy ha elérte az egyik oldalt a labda, akkor változtatjuk a lépések irányát. Használjuk még a **usleep(n)** metódust aminek segítségével lassítjuk a labda mozgását és a **clear()** metódussal pedig csak 1 labdát fogunk látni,mivel minden egyes iterációnál törli a ablakot. Azonban if nélkül is megoldható a feladat:

```
#include <iostream>
#include <vector>
#include <unistd.h>
using namespace std;

const int szelesseg = 40;//ablak szelesseg
const int magassag = 20;//ablak magassag
int x=1, y=1; //labda kezdoertek
int deltax=1,deltay=1; // hanyasaval lepkedjen
vector<int> ablakx;
vector<int> ablaky;

void kirajzol();
void mozgatas(){
  x=x+deltax;
  y=y+deltay;
  deltax=deltax*ablakx[x];
  deltay=deltay*ablaky[y];
}

int main(){

    for (int i=0; i<szelesseg; i++)
    {
         ablakx.push_back(1);
    }

    for (int i=0; i<magassag; i++)
    {
        ablaky.push_back(1);
    }

    ablakx[0]=-1;//bal oldal
```

```
    ablakx[szelesseg-1]=-1; // jobb oldal
    ablaky[0]=-1; //teteje
    ablaky[magassag]=-1; // alja

    for(;;)
    {
        kirajzol();
        mozgatas();
        usleep(100000);
    }

}
```

Hasonlóan az if-es megoldáshoz itt is rögzítjük a kezdőértékeket. Ebben az esetben nem saját ablakot rajzolunk ki, hogy lássunk ilyet is ,aminek előre meghatározzuk a magasságát és szélességét. A **kirajzol()** metódus keretet rajzol ki, amiben majd pattog a labda, más dolga nincs. A **main()** metódusban az ablakunk x és y szélességével töltjük fel a két vektort. Ezután beállítjuk -1 értékre a vektorok két elemét, hogy a **mozgatas()** metódusban megszorozzuk a labda lépéseinek nagyságát (ami egyben az irányért is felelős) az oldalak értékeivel. Ha -1-el szorzunk, akkor ez az jelenti, hogy elérte az egyik oldalt és változtatjuk az irányt, hasonlóan az if-es megoldásnál, ahol ugyanez ifekkel történt.

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/p-adrian05/BHAX/tree/master/textbook_programs/Turing/bogomips

```
#include <stdio.h>

int main()
{
  int word = 1;
  int length = 0;

  do
  {
    length++;
  }
  while (word<<=1);


  printf("A szó %d bites\n", length);

    return 0;
```

```
}
```

A típus méretét, illetve hogy hány bitet foglal, bitshifteléssel könnyen meghatározhatjuk. A while ciklusunkban mindig shiftelünk egyet balra a biteken és addig növeljük a length változót, amíg csupa 0 bitet nem fog tartalmazni a word változónk. Mivel az int típusú változók 4 byte-on tárolódnak, ezért 32 bites lesz a szavunk.

```
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs$ gcc wordLength.c -o ↩
    w
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs$ ./w
A szó 32 bites
```

BogoMips:

Ez az algoritmus a Linux kernelben a CPU sebességét méri fel induláskor a busy-loop beállításához.A busy-loop azt jelenti, hogy egy folyamat folyamatosan vizsgál egy feltételt, amíg az igaz nem lesz és a BogoMips mutatja meg, hogy hány másodpercig nem csinál semmit a CPU.

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

void delay (unsigned long long loops)
{
  for (unsigned long long i = 0; i < loops; i++);
}

int main (void)
{
  unsigned long long loops_per_sec = 1;
  unsigned long long ticks;

  printf ("Calibrating delay loop..");
  fflush (stdout);

  while ((loops_per_sec <<= 1))
    {
      ticks = clock ();
      delay (loops_per_sec);
      ticks = clock () - ticks;
      printf ("%llu %llu\n", ticks, loops_per_sec);

      if (ticks >= CLOCKS_PER_SEC)
  {
    loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

    printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
```

```
      (loops_per_sec / 5000) % 100);

    return 0;
  }
    }

  printf ("failed\n");
  return -1;
}
```

A while ciklusban bitshifteléssel megyünk végig a 2 hatványain. A ticks-ben tároljuk mennyi processzoridőt használt a CPU eddig, majd a delay() függvénynek átadjuk loops_per_sec változót (aminek a bitjei mindig odébb vannak eggyel tolva), ahol elszámolunk 0-tól a változó végéig. Ezután megint lekérjük a processzoridőt kivonva az előző ticks-ben tárolt CPU időt, így megkapjuk, mennyi ideig tartott a elszámolni a loops_per_sec változó végéig. Majd megnézzük if-el, hogy nagyobb vagy egyenlő a kapott ticks, mint a CLOCKS_PER_SEC aminek az értéke 1 millió és ha ez igaz, akkor kiszámoljuk, hogy milyen érték kell ahhoz,hogy a ciklusértékéket megkapjuk, ezzel meghatározva a CPU sebességét.

```
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs/Turing/bogomips$ ./ ↩
   bogo
Calibrating delay loop..2 2
1 4
1 8
1 16
1 32
1 64
2 128
3 256
5 512
9 1024
16 2048
31 4096
58 8192
115 16384
266 32768
489 65536
919 131072
1844 262144
3678 524288
3216 1048576
4878 2097152
9305 4194304
19356 8388608
37773 16777216
76300 33554432
148557 67108864
292380 134217728
583864 268435456
1167209 536870912
```

```
ok - 918.00 BogoMIPS
```

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/p-adrian05/BHAX/blob/master/textbook_programs/Turing/pagerank.c

A Page Rank az interneten található oldalakat rangsorolja. Kezdetben minden oldalnak van egy szavazati pontja és ha az egyik linkeli a másikat, akkor a linkelt oldal megkapja a linkelő pontját. Tehát egy oldal akkor lesz előkelőbb helyen egy google kereséskor, ha minél több másik oldal linkel rá, illetve ezen oldalakra is minél többen linkelnek, annál jobb minőségűnek fog számítani egy linkelése vagy szavazata. Az alábbi algoritmusunk 4 honlapot rangsorol:

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

void kiir (double tomb[], int db)
{
  int i;

  for (i = 0; i < db; ++i)
    printf ("%f\n", tomb[i]);
}

double tavolsag (double PR[], double PRv[], int n)
{
  double osszeg = 0.0;
  int i;

  for (i = 0; i < n; ++i)
    osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

  return sqrt(osszeg);
}

int main (void)
{

  double L[4][4] = {
    {0.0, 0.0, 1.0 / 3.0, 0.0},
    {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
    {0.0, 1.0 / 2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0 / 3.0, 0.0}
  };
```

```
  double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
  double PRv[4] = { 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0 };

  int i, j;

  for (;;)
  {

    for (i = 0; i < 4; ++i)
    {
        PR[i] = 0.0;
        for (j = 0; j < 4; ++j)
            PR[i] += (L[i][j] * PRv[j]);
    }
        if (tavolsag (PR, PRv, 4) < 0.00000001)
            break;

      for (i = 0; i < 4; ++i)
        PRv[i] = PR[i];

  }

  kiir (PR, 4);

  return 0;
}
```

Az L nevű többdimenziós tömbben vannak rögzítve mátrix formájában az adatok a linkelésekről, melyik oldalra melyik oldal linkel és mennyit. A végtelen ciklusban nullázzuk PR összes elemét, majd rögtön hozzáadjuk az L mátrix és PRv vektor szorzatainak értékét. Ezután a távolság metódusunkban végigmegyünk a PR és PRv vektorokon és egy változóban eltároljuk ezek különbségének a négyzetét (hogy ne legyen negatív) és gyököt vonva visszaadjuk az értéket, amely ha kisebb mint 0.00000001, akkor kilépünk a végtelen ciklusból, ellenkező esetben pedig PRv tömböt feltöltjük PR elemeivel. Végül kiiratjuk az értékeket.

```
0.090909
0.545455
0.272727
0.090909
```

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

Megoldás forrása: https://github.com/p-adrian05/BHAX/blob/master/attention_raising/Primek_R/stp.r

A Brun tétel azt mondja, hogy az ikerprímszámok reciprokából képzett összege konvergál egy számhoz. Ezt határt Brun konstansnak nevezzük. Ezzel ellentétben a prímszámok a végtelen felé tartanak.

Mik azok a prímszámok?

A prímszám olyan természetes szám, ami csak önmagával és eggyel osztható és minden természetes szám előállítható prímszámok szorzataként. Ikerprímek pedig azok a prímszámok, amelyek egymás után következnek és különbségük 2.

A Brun tétel szimulációja a matematikai R nyelvben lesz megírva, használva a matlab csomagot a prímszámok számítására.

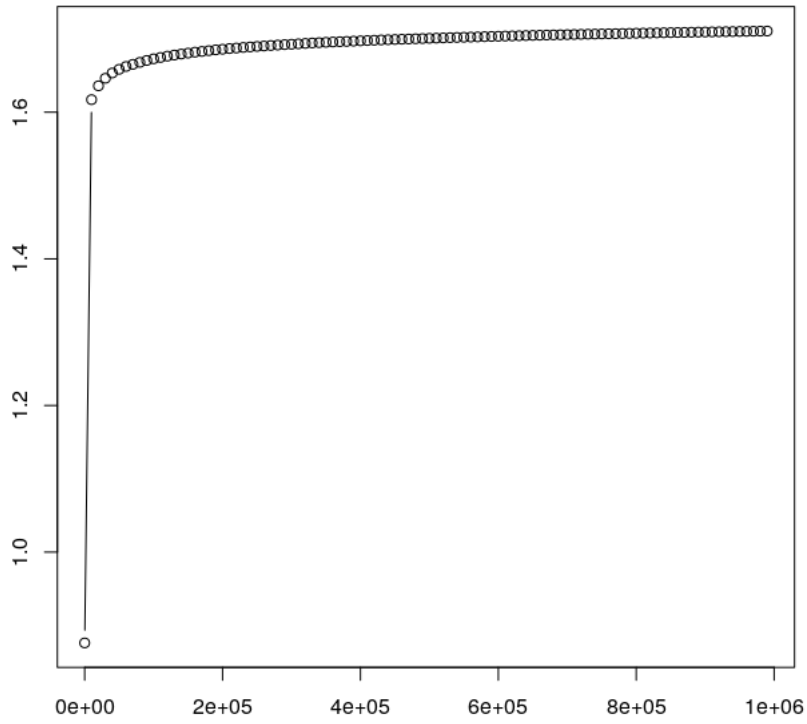
```
sumTwinPrimes <- function(x){

    primes = primes(x)
    diff = primes[2:length(primes)]-primes[1:length(primes)-1]
    idx = which(diff==2)
    t1primes = primes[idx]
    t2primes = primes[idx]+2
    rt1plust2 = 1/t1primes+1/t2primes
    return(sum(rt1plust2))
}

x=seq(13, 1000000, by=10000)
y=sapply(x, FUN = sumTwinPrimes)
plot(x,y,type="b")
```

Létrehozunk egy függvényt aminek paraméterként átadjuk, hogy meddig számolja függvényünk az ikerprímszámokat. A primes vektorba kiszámoljuk a **primes(x)** fügvénnyel a prímszámokat. A diff vektorban eltároljuk az egymásután következő prímszámok különbségét. Az idx vektorban, pedig azokat a helyeket tároljuk, el ahol a különség 2, tehát ikerprímek találhatóak ott. A t1primes vektorban elátároljuk az ikerprímek első párját, a t2primes-ban pedig hozzáadva minden első párhoz kettőt, az ikerprímek második párját is. Az rt1plust2 vektorban tároljuk minden párnak a reciprokainak az összegét, majd ezeket az összegeket **sum()** függvénnyel összeadjuk és visszaadjuk ezt az értéket.

Ezután kirajzoltatjuk és láthatjuk, hogy valóban egy felső határhoz konvergálnak az összegek.



2.1. ábra. A B_2 konstans közelítése

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://github.com/p-adrian05/BHAX/blob/master/attention_raising/MontyHall_R/mh.r

A Monty Hall probléma egy amerikai televíziós vetélekedőben jelent meg, ahol a műsor végén a játékosnak 3 ajtó közül kellett választania. A nyeremény csak az egyik ajtó mögött volt. A játékos választása után a műsorvezető kinyitott egy üres ajtót és feltette a kérdést, hogy fenntartja-e a választását a játékos vagy egy másik ajtót választ. A Monty Hall probléma kérdése, hogy számít-e, hogy a játékos megváltoztatja-e a választását. Józan ésszel gondolkodva nem számít, mivel a maradék két ajtó közül az egyik mögött van a nyeremény, így 50-50% az esélye annak, hogy nyerünk. A feladvány bizonyítása több matematikai professzoron is kifogott, köztük a világhírű Erdős Pálon is, akit csak a számítógépes szimuláció győzött meg, ami alapján számít, hogy másik ajtót választunk, ugyanis ekkor megduplázódik az esélyünk a nyerésre.

Amikor először választunk ajtót, akkor 1/3 az esélye annak, hogy eltaláljuk a nyertes ajtót és 2/3, hogy nem. Ezután a játékvezető kinyit egy ajtót, amelyik üres és ha nem változtatunk a döntésünkön, továbbra is 1/3 lesz annak az esélye, hogy nyertünk. Viszont mivel már csak 2 ajtó van a játékban ezért ha változtatunk, akkor 2/3 lesz az esélyünk a nyerésre.

Ennek bizonyítását láthatjuk R nyelven megfogalmazva:

```
kiserletek_szama=1000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

    if(kiserlet[i]==jatekos[i]){

        mibol=setdiff(c(1,2,3), kiserlet[i])

    }else{

        mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))

    }

    musorvezeto[i] = mibol[sample(1:length(mibol),1)]

}

nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]

}

valtoztatesnyer = which(kiserlet==valtoztat)


sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Először eltároljuk, hogy hány kisérletet végezzünk el, majd a kiserlet és jatekos vektorokban kisorsolunk 1 és 3 között véletlenszerűen számokat. A műsorvezető vektorát beállítjuk a kísérletek számával. Ezután egy for ciklussal végigmegyünk minden kísérleten és ha kiserlet i-edig értéke megyegyezik a a játékos i-edig találatával, az jelenti, hogy eltalálta a játékos a nyereményt és a mibol vektorba az a két érték kerül be amelyeket a játékos nem választott. (Ez a két érték az üres ajtókat jelenti ebben az esetben.)

Ha a játékos nem találta el elsőre a kiserlet vektorban található számot, akkor a mibol vektorba már csak 1 egy érték kerülhet, az amelyik nem a nyeremény és nem a játékos által kiválasztott érték. Ezután a musorvezeto vektorba berakjuk a mibol vektorban található számot, illetve ha két érték van benne akkor a

kettőből az egyiket véletlenszerűen.

Ezután értkezik a kiértékelés. A nemvaltoztatesnyer vektorba kerülnek azok az esetek, amikor elsőre eltalálja a játékos a megfelelő ajtót. Megint végigmegyünk a kísérleteken és a holvalt vektorba azok vagy az az érték kerül az 1, 2 és 3 közül amely nem egyenlő a műsorvezető és a játékos által választottal vagyis ekkor ha váltana a játékos akkor nyerne. A változtat vektorba pedig a holvalt vektor elemei közül az egyiket rakjuk át.

A valtoztatesnyer vektorba pedig azok az értékek kerülnek, amelyek a kiserlet vektorba és a változtat vektorba találhatóak, vagyis ekkor az az ajtó a nyertes ,amelyiket másodjára választanánk. Eztuán pedig kiiratjuk az esetek számait:

```
[1] "Kiserletek szama: 1000000"
> length(nemvaltoztatesnyer)
[1] 333590
> length(valtoztatesnyer)
[1] 666410
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.5005777
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 1000000
```

3. fejezet

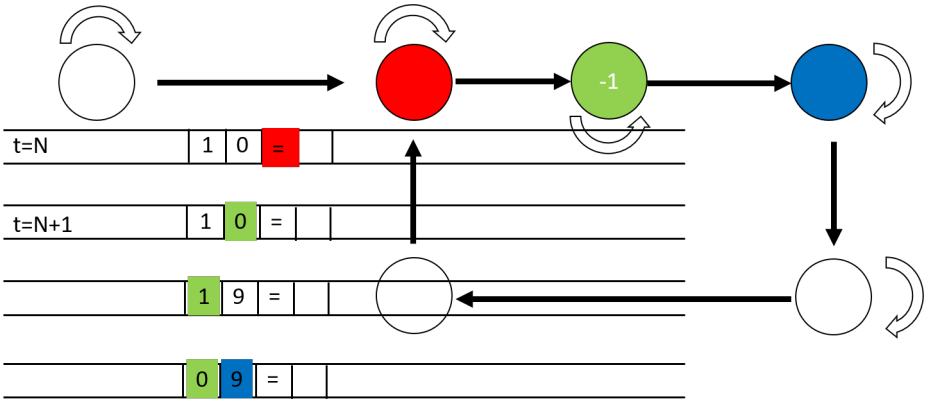
Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:



A decimális számok N darab egyessel vannak ábrázolva unáris számrendszerben, nincs más számjegy. A turing gép az átváltást úgy hajtja végre, hogy addig von ki a megadott decimális számból egyet, amíg az nulla nem lesz és a tárba pakolja az egyeseket. A feldolgozás mindig a szám utolsó számjegyével kezdődik. Ha ez 0, akkor 9-el a kék állapotba kerül és addig ismétlődik, amíg 0 nem lesz. Ezután ez folytatódik a többi számjeggyel ugyanez.

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

```
<utasítás>::=
    <összetett_utasítás> | <feltételes_utasítás> | <iterációs_utasítás> |
    <vezérlés_átadó_utasítás> |  <címkézett_utasítás> | <kifejezés_utasítás ↩
       >| <nulla_utasítás>
<összetett_utasítás>::= <deklaráció_lista> | <utasításlista>
<deklaráció_lista>::= <deklaráció>
<utasításlista>::= <utasítás>
<feltételes_utasítás>::= if | if else | switch
<iterációs_utasítás>::= while | do while | for
<vezérlés_átadó_utasítás>::= break | return | goto | continue
<címkézett_utasítás>::= <azonosító>
<kifejezés_utasítás>::= <kifejezés>
<nulla_utasítás>::= ;
```

C99-el lefordul:

```
    #include <stdio.h>

    int main ()
    {
      // Printing to screen.
      printf ("Hello World\n");
    }
```

```
adrian@adrian-MS-7817:~/Desktop/programs$ gcc -o a -std=c99 a.c
adrian@adrian-MS-7817:~/Desktop/programs$
```

Azonban C89-el nem fordul le:

```
adrian@adrian-MS-7817:~/Desktop/programs$ gcc -o a -std=c89 a.c
a.c: In function ‘main’:
a.c:5:7: error: C++ style comments are not allowed in ISO C90
       // Printing to screen.
```

```
       ^
a.c:5:7: error: (this will be reported only once per input file)
```

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: https://github.com/p-adrian05/BHAX/blob/master/textbook_programs/Chomsky/realnumber.l

A lexer minta illesztésen alapulva legenerálja a lexikális elemzőnket, nekünk csak a mintát kell megadnunk, ami alapján figyeli a begépelt szöveget. Ebben a feladatban olyan programot írunk, ami a begépelt szövegből felismeri és megszámolja a valós számokat.

```
%{
#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;
}
```

Az első részben deklarálunk egy int változót, amit növelve kapjuk meg a számok számát és végén definiáljuk digit néven 0-9-ig a számjegyeket.

```
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
```

A második részben adjuk a mintát. A * előtt a digit, azt jelenti, hogy digitből(számból) bármennyi lehet. A . önmagában azt jelenti, hogy bármilyen karakterre rá lehet illeszteni, azonban nekünk le kell védeni \ jellel

és így a valós számoknál lévő pontot fogja értelmezni. Utána digit, vagyis megint jönnek a számjegyek, a + pedig azt jelenti, hogy legalább 1 számnak kell lennie a pont után. És ha van találat a mintára, akkor növeljük a realnumbers változót. Ezután kiiratjuk a felismert számot, illetve az atof-al átkonvertált double verzióját is.

```
%%
{digit}*(\.{digit}+)? {++realnumbers;
printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
```

A harmadik rész maga a program, ahol meghívjuk a lexert, ami összerakja a fenti minta alapján a programunkat és a program befejezte után kiírja a talált számokat.

Fordításkor először a lexnek megmondjuk milyen kimenetet készítsen a fent megírt realnumber.l forrásból. Ezután szokott módon fordítjuk a kapott c forráskódot hozzálinkelve a flex könyvtárat.

```
adrian@adrian-MS-7817:~/Desktop/programs$ clear
adrian@adrian-MS-7817:~/Desktop/programs$ lex -o realnumber.c realnumber.l
adrian@adrian-MS-7817:~/Desktop/programs$ gcc realnumber.c -o realnumber - ↩
   lfl
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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);
```

Ha a SIGINT jel kezelése nem volt figyelmen kívűl hagyva, akkor a jelkezelo függvény kezelje, máskülönben figyelmen kívűl legyen hagyva.

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

for ciklussal 0-tól 4-ig megyünk, az i-t növelve. ++i azt jelenti, hogy megnöveli az i étékét majd visszaadja az 1-el megnövelt értéket.

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

for ciklussal 0-tól 4-ig megyünk, az i-t növelve. i++ azt jelenti, hogy megnöveli az i étékét, de először visszaadja az eredeti értéket és azután növeli. For ciklusban nincs jelentősége, hogy ++i vagy i++;

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

Indulunk for ciklussal 0-tól 4-ig, majd tomb ielemét megváltoztatjuk arra az i-re ami megnövelés előtt volt.

```
adrian@adrian-MS-7817:~/Desktop/programs$ splint a.c
Splint 3.1.2 --- 20 Feb 2018

a.c: (in function main)
a.c:9:26: Expression has undefined behavior (left operand uses i,  ↩
   modified by
            right operand): tomb[i] = i++
  Code has unspecified behavior. Order of evaluation of function  ↩
     parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining  ↩
     evaluation
  order, then the result of the expression is unspecified. (Use - ↩
     evalorder to
  inhibit warning)
a.c:15:2: Path with no return in function declared to return int
  There is a path through a function declared to return a value on  ↩
     which there
  is no return statement. This means the execution may fall through  ↩
     without
  returning a meaningful result to the caller. (Use -noret to inhibit  ↩
     warning)

Finished checking --- 2 code warnings
```

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

0-tól megyünk for ciklussal, addig amíg i kisebb mint n és ha d tömbre muntató mutató következő eleme egyenlő az s mutató által mutatatott tömb következő elemével. Hiba: összehasonlításkor 2db egyelnőségjelet használunk.

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

Kiiratunk két egész számot. Mindkettőt az f függvény adja vissza. Első esetben az f függvénynek átadjuk az a változót és az a változó 1-el megnövelt értékét. Másodszor pedig az a változó 1-el megnövelt értékét és a-t adjuk átadjuk az f függvénynek.

```
adrian@adrian-MS-7817:~/Desktop/programs$ splint a.c
Splint 3.1.2 --- 20 Feb 2018

a.c: (in function main)
a.c:16:22: Argument 2 modifies a, used by argument 1 (order of  ↩
   evaluation of
              actual parameters is undefined): f(a, ++a)
  Code has unspecified behavior. Order of evaluation of function  ↩
     parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining  ↩
     evaluation
  order, then the result of the expression is unspecified. (Use - ↩
     evalorder to
  inhibit warning)
a.c:16:30: Argument 1 modifies a, used by argument 2 (order of  ↩
   evaluation of
              actual parameters is undefined): f(++a, a)
a.c:16:17: Argument 2 modifies a, used by argument 3 (order of  ↩
   evaluation of
    actual parameters is undefined): printf("%d %d", f(a, ++a), f(++a, ↩
        a))
a.c:16:28: Argument 3 modifies a, used by argument 2 (order of  ↩
   evaluation of
    actual parameters is undefined): printf("%d %d", f(a, ++a), f(++a, ↩
        a))
a.c:2:5: Function exported but not used outside a: f
  A declaration is exported, but not used outside this module.  ↩
     Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   a.c:6:1: Definition of f

Finished checking --- 5 code warnings
```

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

Két egész számot iratunk ki, az egyik az f függvény által visszaadott szám, az a változót átadjuk az f-nek, a másik pedig az a változó.

```
adrian@adrian-MS-7817:~/Desktop/programs$ splint a.c
Splint 3.1.2 --- 20 Feb 2018

a.c:2:5: Function exported but not used outside a: f
  A declaration is exported, but not used outside this module.  ↩
     Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   a.c:6:1: Definition of f

Finished checking --- 1 code warning
```

viii.

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

Két egész számot iratunk ki, az egyik az f függvény által visszaadott szám, az a változó memóriacímét átadjuk az f-nek, a másik pedig az a változó.

```
adrian@adrian-MS-7817:~/Desktop/programs$ splint a.c
Splint 3.1.2 --- 20 Feb 2018

a.c: (in function main)
a.c:16:19: Function f expects arg 1 to be int gets int *: &a
  Types are incompatible. (Use -type to inhibit warning)
a.c:2:5: Function exported but not used outside a: f
  A declaration is exported, but not used outside this module.  ↩
     Declaration can
  use static qualifier. (Use -exportlocal to inhibit warning)
   a.c:6:1: Definition of f

Finished checking --- 2 code warnings
```

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

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://github.com/p-adrian05/BHAX/blob/master/textbook_programs/Chomsky/Matlog/-matlog.tex

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

Ahhoz, hogy a fenti mondatatokat megadjuk, definiálni kell következőket:

```
$ (x \text{ páros}) \leftrightharpoons \exists y(y+y=x)$
$ (x < y) \leftrightharpoons \exists z(z+x=y) \wedge \neg (x=y)$
$(x \vert y) \leftrightharpoons \exists z ( z \cdot x=y )\wedge ( x \neq 0) ↩
   $
$(x \text{ prím}) \leftrightharpoons  (\forall z( z \vert x )\supset ( z =  ↩
   x \vee z=S0))\wedge ( x \neq 0)\wedge ( x \neq S0)$
```

Majd ezt lefordítva pdf formátumba ezt kapjuk:

- $(x \text{ páros}) \leftrightharpoons \exists y(y + y = x)$
- $(x < y) \leftrightharpoons \exists z(z + x = y) \wedge \neg(x = y)$
- $(x|y) \leftrightharpoons \exists z(z \cdot x = y) \wedge (x \neq 0)$
- $(x \text{ prím}) \leftrightharpoons (\forall z(z|x) \supset (z = x \vee z = S0)) \wedge (x \neq 0) \wedge (x \neq S0)$
- $(\forall x \exists y((x < y) \wedge (y \text{ prím}))) \leftrightharpoons (\infty \text{ sok prímszám van})$
- $(\forall x \exists y((x < y) \wedge (y \text{ prím}) \wedge (SSy \text{ prím}))) \leftrightharpoons (\infty \text{ sok iker-prímszám van})$
- $(\exists y \forall x(x \text{ prím}) \supset (x < y)) \leftrightharpoons (\text{véges sok prímszám van})$
- $(\exists y \forall x(y < x) \supset \neg(x \text{ prím})) \leftrightharpoons (\text{véges sok prímszám van})$

x akkor páros, ha van olyan y változó, amihez önmagát hozzáadva x-et kapjuk.

x kisebb mint y, ha van olyan z változó, amihez x-et hozzáadva y-t kapjuk és x nem egyenlő y-nal.

x osztója y-nak, ha létezik olyan z változó, hogy ha z-t megszorozzuk x-el, y-t kapjuk és x nem egyenlő 0-val.

x prímszám, ha minden olyan z változó osztója x-nek ami egyenlő x-el vagy 1-gyel és x nem egyenlő 0-val és 1-gyel.

Végtelen sok prímszám van: Minden x változó esetén van olyan y változó, amely nagyobb mint x és y prímszám.

Végtelen sok iker-prímszám van: Minden x változó esetén van olyan y változó, amely nagyobb mint x, prímszám és hozzáadva kettőt is prímszám.

Véges sok prímszám van: Van olyan y változó, amely minden x változó esetén, ha x prím, akkor nagyobb mint x.

(2. megfogalmazás) Véges sok prímszám van: Van olyan y változó, amely ha minden x változó esetén, kisebb mint x, akkor x nem 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

```
int *pointerFg(int *c)
{
    return c;
}

int fuggveny(int a, int b)
{
    return a;
}
```

```
typedef int(*A) (int,int);;
A fuggveny2(int);


int main()
{
    //egesz
    int a = 1;

    //egeszre mutato mutato
    int *b;

    //egesz referenciaja
    b = &a;

    //egeszek tombje
    int tomb[5] = {1,2,3,4,5};

    //egeszek tombjenek referenciaja
    int (&tombref)[5] = tomb;

    //egeszre mutato mutatok tombje
    int *tombMutato[5];

    //egeszre mutato mutatot visszaado fuggveny
    int *c = pointerFg(&a);

    //egeszre mutato mutatot visszaado fuggvenyre mutato mutato
    int *(*mutato)(int*) = pointerFg;

    //egeszet visszaade es ket egészet kapo fuggvenyre mutato mutatot ↩
       visszaado, egeszet kapo fuggveny
    A st = fuggveny2(10);


}
```

Mit vezetnek be a programba a következő nevek?

- `int a;`

  Egy int típusú a nevű változó.

- `int *b = &a;`

  b egészre mutató mutató a-ra mutat.

- `int &r = a;`

r egeszre mutató mutató, ami a címet tartalmazza.

- ```
  int c[5];
  ```

  Öt elemű egészekből álló tömb.

- ```
  int (&tr)[5] = c;
  ```

  Öt elemű egészekből álló tombre mutató mutató, ami a c tömbre mutat.

- ```
  int *d[5];
  ```

  5 elemű egészekre mutató mutatókból álló tömb.

- ```
  int *h ();
  ```

  Egy egésszel visszatérő paraméter nélküli függvényre mutató mutató.

- ```
  int *(*l) ();
  ```

  Egy egészre mutató mutatóval visszatérő, paraméter nélküli függvényre mutató mutató.

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

  Egy egésszel visszatérő 2 egészet váró függvényre mutató mutatóval visszatérő 2 egészet váró függvény.

- ```
  int (*(*z) (int)) (int, int);
  ```

  Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre mutató mutató.

Megoldás videó:

Megoldás forrása: https://github.com/p-adrian05/BHAX/blob/master/textbook_programs/Chomsky/deklaracio.c

Tanulságok, tapasztalatok, magyarázat...
  ```

4. fejezet

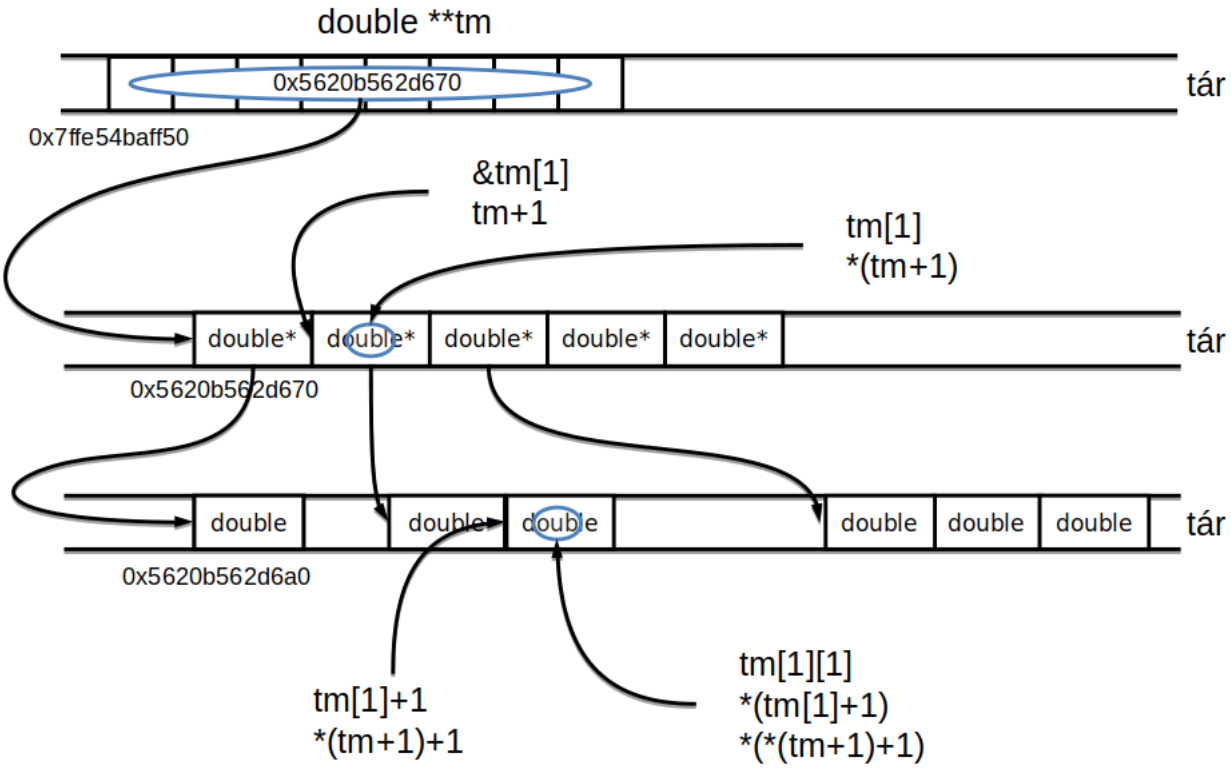
Helló, Caesar!

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 videó: https://youtu.be/1MRTuKwRsB0, https://youtu.be/RKbX5-EWpzA.

Megoldás forrása: https://github.com/p-adrian05/BHAX/blob/master/textbook_programs/Caesar/tm.c



4.1. ábra. A double ** háromszögmátrix a memóriában

```
#include <stdio.h>

#include <stdlib.h>
int
main ()
{
    int nr = 5;
    double **tm;
    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }
    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL ↩
           )
        {
            return -1;
        }
    }
    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;
}
```

A képen a double ** háromszögmátrix memóriafoglalását bemutató ábrát láthatjuk. Az **nr** változóval megadjuk a háromszögmátrix magasságát. Jelen esetben ez 5 lesz. A **double ** tm** egy mutatóra mutató mutató. A **malloc**-al foglalunk memóriát ami egy mutatót ad vissza a lefoglalt területre (double * mutatót), amit double ** mutatóvá típuskényszerítjük, így a tm mutatónkat rá tudjuk állítani erre a tárterületre. A **malloc**-nak átadjuk mekkora területet foglaljon le. Ebben az esetben a double * mutató mérete, ami 8, szorozva az nr változóval, így 40 -et kapva ennyi bájtot foglalunk a memóriába.

Ezután egy for ciklussal végigmegyünk ezen a területen 1-5-ig és minden double * mutatót is ráállítunk

egy double típusú területre, aminek a darabszámát mindig egyel növeljük,így az első mutató 1db double-re mutat, a második 2-re és így tovább haladva 5-ig, így kapjuk meg a háromszögmátrixot. Ezután értékeket adunk a változóinknak és kiiratjuk őket, majd felszabadítjuk a lefoglalt területeket.

```
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs/Caesar$ ./tm
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
0.000000,
1.000000, 2.000000,
3.000000, 4.000000, 5.000000,
6.000000, 7.000000, 8.000000, 9.000000,
10.000000, 11.000000, 12.000000, 13.000000, 14.000000,
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása: https://github.com/p-adrian05/BHAX/blob/master/textbook_programs/Caesar/exor/exor.c

Ebben a feladatban a kizáró vagyos (XOR) titkosítás van bemutatva. A kizáró vagyos titkosításkor a titkosítandó szöveg bájtjait lefedjük a titkosító kulcs bájtjaival és egy kizáró vagy műveletet végzünk el rajtuk. A kizáró vagy művelet 1 értéket ad, ha a két bit különböző és 0-t ha megegyező. Például:

```
            Kódolás:
            A tiszta szöveg bájtja:                          10011
            A kulcs bájtja:                                  00110
            A titkosított szöveg bájtjai XOR művelet után:   10101
            Dekódolás:
            A kulcs bájtja:                                      00110
            A titkosított szöveg bájtjai:                        10101
            XOR művelet után visszakapjuk az eredeti szöveget:   10011
```

```
#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);
    }
}
```

A program elején definiáljuk a kulcs maximum méretét és a beolvasáshoz szükséges buffer méretét. Ezután a main függvényben létrehozzuk az ezekkel a méretekkel rendelkező char típusú tömböket. A kulcs_index változó mutatja az aktuális elemét a kulcsnak, amivel majd végrehajtjuk a műveletet, az olvasott_bajtok pedig a beolvasott bájtok számát fogja tárolni. Az **strlen()** függvény segítségével rögzítjuk a kulcs méretét amit a parancssorban adunk meg argumentumként. A while ciklus addig fut, amíg tudunk olvasni a bemenetről és azt tároljuk a bufferben, ha beolvasott szöveg végéhez értünk, akkor a **read()** függvény 0 értéket ad vissza és a ciklus véget ér. A while ciklusban végigmegyünk a beolvasott bájtokon és végrehajtjuk a tikosítást alkalmazva a kizáró vagy műveletet, majd az eredményt kiírjuk egy dokumentumba.

Fordításkor a c99 szabványt kell használnunk:

```
gcc exor.c -o e -std=c99
```

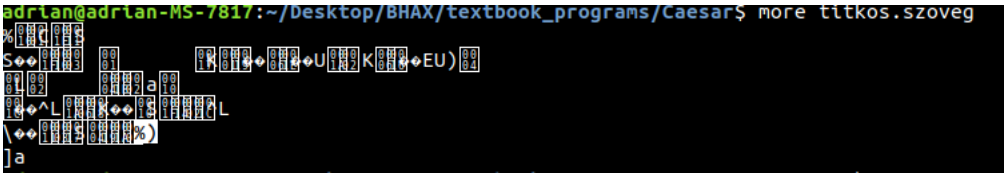
Eztuán létrehozunk egy szövegfájlt, amiben tároljuk a titkosítandó szövegünket

```
adrian@adrian-MS-7817:~/Desktop$ more tiszta.szoveg
Nem tudok kimerítő leírást adni arról, hogy hogyan tudsz megtanulni
programozni -- nagyon összetett tudásról van szó. Egyet azonban
elárulhatok: a könyvek és tanfolyamok nem érnek túl sokat (sok,
valószínűleg a legtöbb hacker autodidakta). Aminek van értelme:
 (a) kódot olvasni és kódot írni.
```

Majd futtatjuk a programot megadva paraméterként a kulcsot és a tiszta.szoveg fájlunk tartalmát a programba irányítjuk (ezt olvassa be). A kimenő adatokat pedig a titkos.szoveg nevű fájlba irányítjuk. Ez lesz a titkosított szövegünk.

```
adrian@adrian-MS-7817:~/Desktop$ ./e kulcs <tiszta.szoveg >titkos.szoveg
```

Kiírva a tikos.szoveg tartalmát láthatjuk a kódolt szövegünket:



A dekódolás hasonlóan zajlik. Megadjuk azt a kulcsot amivel a kódolás történt és beolvassuk a tikos.szoveg tartalmat.

```
adrian@adrian-MS-7817:~/Desktop$ ./e kulcs <titkos.szoveg
Nem tudok kimerítő leírást adni arról, hogy hogyan tudsz megtanulni
programozni -- nagyon összetett tudásról van szó. Egyet azonban
elárulhatok: a könyvek és tanfolyamok nem érnek túl sokat (sok,
valószínűleg a legtöbb hacker autodidakta). Aminek van értelme:
(a) kódot olvasni és kódot írni.
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:https://github.com/p-adrian05/BHAX/blob/master/textbook_programs/Caesar/exor/ExorTitkos

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcsSzöveg,
            java.io.InputStream bejövőCsatorna,
            java.io.OutputStream kimenőCsatorna)
            throws java.io.IOException {

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;
        while((olvasottBájtok =
                bejövőCsatorna.read(buffer)) != -1) {

            for(int i=0; i<olvasottBájtok; ++i) {

                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);
```

```
        }

    }

    public static void main(String[] args) {

        try {

            new ExorTitkosító(args[0], System.in, System.out);

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Javaban egy ExorTitkosító publikus osztályt vagy objektumot hozunk létre amiben a munkát végezzük. Először létrehozzuk a konstruktort, amivel majd átadjuk az objektumnak a kapott értékeket. Jelen esetben a paraméterként átadott kulcsot tároljuk egy stringbe, majd a bejövő és kijövő csatornát hozzuk létre. Egy byte tömben tároljuk a megadott kulcs bájtjait, illetve a buffer méretet. A while ciklusba addig olvasunk az int típusú beolvasottBájtok nevű változónkba, amíg -1-et nem kapunk, és egy for ciklussal iterálunk az olvasott bájtokon egyesével végezve a kizáró vagy műveletet a kulcsunk bájtjaival, amit tárolunk buffer tömbben.Végül a kimenő csatornán kiiratjuk a titkosított szöveget. A main() metódusban példányosítjuk az ExorTitkosító objektumunkat és átadjuk neki a paraméterként kapott kulcsot és a be és kimenő csatornát Mindezt try és catch ágba téve, az esetleges hibák elkapásához.

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/p-adrian05/BHAX/blob/master/textbook_programs/Caesar/exor/t.c

```
int
main (void)
{
  char kulcs[KULCS_MERET];
  char titkos[MAX_TITKOS];
  char *p = titkos;
  int olvasott_bajtok;
```

```
  while ((olvasott_bajtok =
    read (0, (void *) p,
    (p - titkos + OLVASAS_BUFFER <
     MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
    p += olvasott_bajtok;
  // maradek hely nullazasa a titkos bufferben
  for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';
  // osszes kulcs eloallitasa
 for (int ii = '0'; ii <= '9'; ++ii)
    for (int ji = '0'; ji <= '9'; ++ji)
      for (int ki = '0'; ki <= '9'; ++ki)
  for (int li = '0'; li <= '9'; ++li)
    for (int mi = '0'; mi <= '9'; ++mi)
      for (int ni = '0'; ni <= '9'; ++ni)
        for (int oi = '0'; oi <= '9'; ++oi)
    for (int pi = '0'; pi <= '9'; ++pi)
      {
        kulcs[0] = ii;
        kulcs[1] = ji;
        kulcs[2] = ki;
        kulcs[3] = li;
        kulcs[4] = mi;
        kulcs[5] = ni;
        kulcs[6] = oi;
        kulcs[7] = pi;
    if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
          printf
      ("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
       ii, ji, ki, li, mi, ni, oi, pi, titkos);
        // ujra EXOR-ozunk, igy nem kell egy masodik buffer
        exor (kulcs, KULCS_MERET, titkos, p - titkos);
  return 0;
}
```

Ez az exor törő program 0-9-ig számokat tartalmazó 8 számjegyű számmal kódolt szöveget töri fel. A program teljes egészében a forrásban látható. Hosszúsága miatt csak a main() függvényt szúrtam be. Hasonlóan az exor titkosítónál, itt is meg kell adni a max kulcs méretet, jelen esetben ez 8, illetve a titkos szövegünk max méretét. A while ciklusban beolvassuk a titkos szöveget, majd a titkos bufferben nullázzuk a maradék helyet. Ezután az összes lehetséges kulcsot előállítjuk for ciklusokkal 0-9-ig, közben alkalmazva a kulcsokon a kizáró vagy műveletét a tikos szöveggel és ha van találat akkor kiiratjuk a kulcsot és a már feltört szöveget, majd újra exorozunk, így nem kell új buffer.

Az exor titkosító programunkkal titkosítottunk egy szöveget a megfelelő kulcsot megadva, majd az exor törővel sikerült ezt feltörni, második találatra. Minél több különböző számot tartalmaz a kucsunk, annál tovább fog tartani a törés. Jelen esetben egy egyszerűbb kulccsal lett titkosítva a sz9veg, a törés gyorsabbá tétele miatt.

```
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs/Caesar$ ./e 10000001 ↩
    <tiszta.szoveg > titkos.szoveg
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs/Caesar$ ./t <titkos. ↩
   szoveg
Kulcs: [10000000]
Tiszta szoveg: [Nem tudnk kimertő lerást 'dni arr$^3$l, hogx hogyan!tudsz ↩
    mdgtanulnh
progralozni --!nagyon ¶sszeteut tudárról vao szó. Dgyet aznnban
el¡rulhatnk: a k$\div$nyvek ès tanfomyamok ndm érnej túl snkat (soj,
valórzínűldg a leguöbb habker autndidakta(. Aminej van éstelme:
)a) kódnt olvasoi és k$^3$dot ísni.
]
Kulcs: [10000001]
Tiszta szoveg: [Nem tudok kimerítő leírást adni arról, hogy hogyan tudsz  ↩
   megtanulni
programozni -- nagyon összetett tudásról van szó. Egyet azonban
elárulhatok: a könyvek és tanfolyamok nem érnek túl sokat (sok,
valószínűleg a legtöbb hacker autodidakta). Aminek van értelme:
(a) kódot olvasni és kódot írni.
```

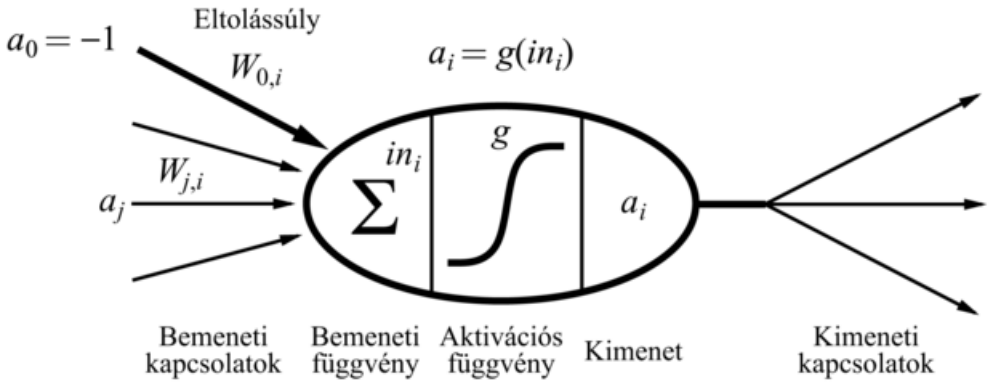
4.5. Neurális OR, AND és EXOR kapu

R

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

Megoldás forrása: https://github.com/p-adrian05/BHAX/blob/master/attention_raising/NN_R/nn.r

A neurális háló vagy számítógépes idegháló az emberi agy által fogadott elektromos jelek feldolgozásán, összegyűjtésén és szétterjesztésén alapszik. Az ábrán látható az idegsejt matematikai modellje:



A bemeneten számok érkeznek amelyeknek van egy súlya, amit összeadunk és a neuron akkor fog "tüzelni", ha a bemeneti értékek súlyozott összege meghalad egy bizonyos köszöböt, ami a W0 lesz -1 értékkel, ez az eltolássúly. Ezután egy g aktivációs függvényt alkalmaz a kapott értékre, majd tovább adja a függvény értékét.

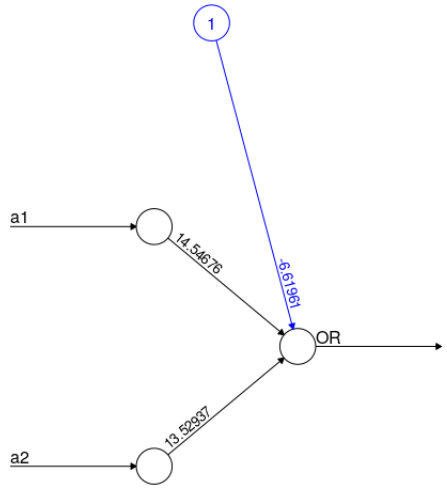
```
library(neuralnet)
```

```
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])
```

R nyelvben dolgozunk, amihez telepíteni kell először a neuralnet csomagot. A a1 és a2-be berakjuk az értékeket és az OR-ban megadjuk mit kellene kapnia ha logikai vagy művelettet laklamzaunk a1 és a2-re, tehát megtanítjuk a szabályokat. Ez alapján elkezni magát tanítani a program, a súlyokat beállítja, majd a compute parancssal leelenőrizzük, hogy valóban a megadott értékeinkre helyes eredmnényt ad a program, amit megtanult, hogyan állítson elő. Láthatjuk, hogy jó eredményeket kaptunk, illetve a jó erdeményhez nagyon közelitő értékeket.

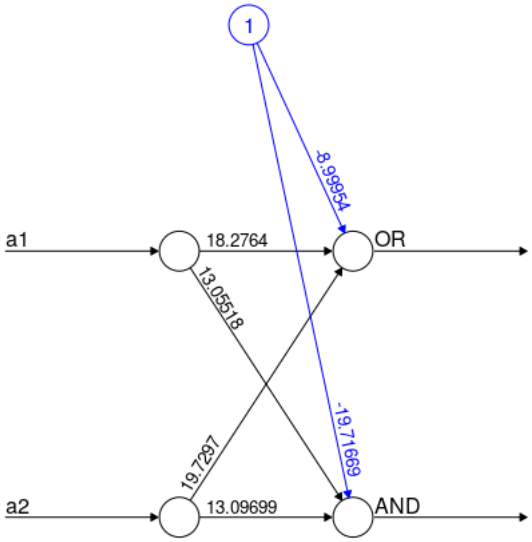
```
 compute(nn.or, or.data[,1:2])
$neurons
$neurons[[1]]
     a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1
$net.result
            [,1]
[1,] 0.001332171
[2,] 0.999639318
[3,] 0.999002995
[4,] 1.000000000
```

Illetve vizuálisan is láthatjuk a hálónkat a súlyokkal:




```
a1    <- c(0,1,0,1)
a2    <- c(0,0,1,1)
OR    <- c(0,1,1,1)
AND   <- c(0,0,0,1)
orand.data <- data.frame(a1, a2, OR, AND)
nn.orand <- neuralnet(OR+AND~a1+a2, orand.data, hidden=0, linear.output= ↩
   FALSE, stepmax = 1e+07, threshold = 0.000001)
plot(nn.orand)
compute(nn.orand, orand.data[,1:2])
```

Ebben az esetben már az és logikai műveletet is megtanítjuk a programnak, az előzővel megegyező módon.



Error: 3e-06 Steps: 188

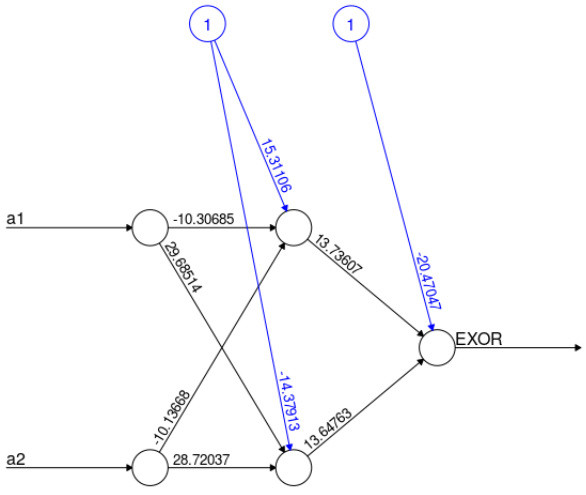
Az exor művelet megtanítása ugyanezzel az algoritmussal azonban már nem működik. Nem kapunk helyes eredményt, ezért ide egy más algoritmus fog kelleni.

```
 compute(nn.or, or.data[,1:2])
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1
$net.result
          [,1]
[1,] 0.4999969
[2,] 0.5000000
[3,] 0.4999995
[4,] 0.5000025
```

```
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=2, linear.output=FALSE,  ↩
   stepmax = 1e+07, threshold = 0.000001)
plot(nn.exor)
compute(nn.exor, exor.data[,1:2])
```

A különbség ez előző algoritmusokhoz képest itt annyi, hogy több rejtett neuronnal dolgozunk, jelen esetben 2-vel, tehát több rétegű neuronokkal lehetséges a tanítás. A hidden értékét kell 2-re állítani. Láthatjuk az eredmény így már helyes lesz:

```
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1
$neurons[[2]]
     [,1]       [,2]          [,3]
[1,]    1 0.99999978 5.691445e-07
[2,]    1 0.99333503 9.999998e-01
[3,]    1 0.99437209 9.999994e-01
[4,]    1 0.00586727 1.000000e+00
$net.result
             [,1]
[1,] 0.001187882
[2,] 0.998911294
[3,] 0.998926671
[4,] 0.001178602
```



4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:https://youtu.be/XpBnR31BRJY

Megoldás forrása: https://github.com/p-adrian05/BHAX/tree/master/textbook_programs/Caesar/perceptron

Ebben a feladatban a mandelbrot halmaz által generált kép rgb kódjait áttesszük a neurális háló inputjába, egy három rétegű hálót csinálunk és végül különböző számítások alapján kapunk a 3. rétegben egy számot.

```
#include <iostream>
#include "mlp.hpp"
#include "png++/png.hpp"
int main (int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width()*png_image.get_height();

    Perceptron* p = new Perceptron(3, size, 256, 1);
    double* image = new double[size];

    for(int i {0}; i<png_image.get_width(); ++i)
         for(int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;
    double value = (*p) (image);
    std::cout << value << std::endl;
    delete p;
    delete [] image;
}
```

Használjuk az mlp.hpp-t amiben a perceptron osztály van. Beimportáljuk az előállított képünket. A perceptron objektumnak foglalunk szabad területet. Megadjuk , hogy hány réteget használunk, a kép méretét, és hogy 1 szám legyen az eredmény. Lefoglalunk a képnek helyet, majd feltöltjük for ckulussal a paraméterben megadott képpel, végül meghívjuk a függvény operátort aminek átadjuk a képet. Végül kiírjuk a függvény által visszaadott értéket.

Az alábbi módon fordítjuk:

```
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs/Caesar$ g++ mlp.hpp  ↩
   main.cpp -o perc -lpng -std=c++11
```

Majd lefuttatva, átadva a képet megkapjuk az értéket:

```
adrian@adrian-MS-7817:~/Desktop/BHAX/textbook_programs/Caesar$ ./perc  ↩
   mandel.png
0.585456
```

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

5.3. Biomorfok

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

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

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

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 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:

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

Juhász István - Magas szintű programozási nyelvek 1

A programnyelvek három szintjét különböztetjük meg: gépi nyelv, assembly szintű nyelv, magas szintű nyelv. A magas szintű nyelven nyelven megírt programot forrásprogramnak nevezzük. Összeállítását szintaktikai és szemantikai szabályok alapján vegzi a fordító program ami gépi nyelvre fordítja a programot, hogy a processzor végre tudja hajtani a programot. A fordítóprogram a következő lépéseket hajtja végre: lexikális elemzés, szintaktikai elemzés, szemantikai elemzés, kódgenerálás. A másik technika egy forrásprogram végrehajtására az interpreteres technika, ami nem készít tárgyprogramot, hanem az utasításokat értelmezi és azonnal végre is hajtja. Minden programnyelvnek megvan a saját szabványa, amit hivatkozási nyelvnek hívunk.

Programnyelvek osztályozása

Imperatív nyelvek: a programozó egy programszöveget ír, algoritmust kódol ami működteti a processzort. Alcsoportjai: eljárásorientált és objektumorientált nyelvek.

Deklaratív nyelvek: Nem algoritmikus nyelvek, a proramozó csak a problémát oldja meg, a nyelvi implementációkban van beépítve a megoldás megkeresésének módja, a programozónak nincs lehetősége memóriaműveletekre. Alcsoportjai: Funkcionális és logikai nyelvek.

Karakterkészlet

Minden program forrászövegének legkisebb alkotórészei a karakterek, amit minden nyelv definiál 3 kategóriába: betűk, számjegyek egyéb karakterek. A lexikális egységek a program azon elemei, melyeket a fordító a lexikális elemzés során felsimer és tokenizál. Fajtái: többkarateres szimbólum, szimbolikus név, címke, megjegyzés, literál. A többkarateres szimbolumok olyan karaktersorozatok, amelyeknek csak a nyelv tulajdonít jelentést.Pl.: ++, --, /*,*/. Szimbolikus nevek közül az azonosító olyan karaktersorozat, ami betűvel kezdődik és betűvel vagy számjeggyel folytatódhat. A kulcsszavak vagy védett szavak olyan szavak amelyeknek a nyelv jelentést tulajdonít. Pl.: if, for, case, break. A stantard azonosítónak a nyelv tulajdonít jelentést, de a programozó által megváltoztatjató, például a NULL. A megjegyzés olyan programozási eszköz, melynek segítségével a programban olyan karaktersorozatok írása megengedett, amelyek nem a fordítónak szól, hanem a programot olvasónak. Általában a ezek a program működésével kapcsolatos magyarázó szövegek. A literál pedig olyan eszköz aminek segítségével fix értékek építhetők be a program szövegébe.

Adattípusok

Az adattípusnak van egy neve, ami egy azonosító. Minden adattípus mögött van egy belső ábrázolális mód. A reprezentáció az egyes típusok tartományába tartozó értékek tárban való megjenelését határozza meg, tehát, hogy az egyes elemek hány bájtra képződnek le. Saját típust úgy tudunk létrehozni, hogy megadjuk a tartományát, a műveleteit és a reprezentációját. Két nagy csoportjuk van: az egyszerű adattípus, tartománya atomi értékeket tartalmaz és összetett adattípus, aminek tartományának elemei is valamilyen típussal rendelkeznek. Az egyszerá adattípusba tartozik az egész típus, belső ábrázolásuk fix pontos.A valós típusok belső ábrázoláluk lebegőpontos. A karakteres típus elemei karakterek, karakterlánc típusé pedig karaktersorozatok. A logikai típus, igaz vagy hamis értéket tárol. Az összetett típusok közül a két legfontosabb a tomb és a rekord. A tömb statikus és homogén összetett típus, amelyben az elemek azonos típusúak. A tömböt mint típust meghatározza a dimenzióinak száma, hány sor, hány oszlopból áll, az elemek indexei, elemek sorszáma, amely egész típusú és az elemek típusa. A mutató típus egyszerű típus, amely tárcímeket tárolhat. Egyik legfontosabb művelete a megcímzett tárterületen elhelyezkedő érték elérése.

A nevesített konstansnak három része van: név, típus, érték és mindig deklarálni kell, ennek értéke ekkor eldől és nem változtatható meg. A változónak négy része van? név, attribútumok, cím, érték.A név egy azonosító. Az attribútum a típusa és deklarációval kap értéket, amely változtatható a program futása során. Mindaddig amíg nincs értéke, addig határozatlan, tehát nem használható fel. Explicit vagy automatikus deklaráció lehetséges. Előbbi esetén a programozó végzi a deklarációt, utóbbi esetben pedig a fordítóprogram rendel attribútumot azokhoz a változókhoz amelyek nincsenek explicit módon megadva, deklarálva. A változóhoz cím rendelhető két féle módon: dinamukus tárkiosztás: a futás előtt eldől a változó címe és futás alatt ez nem változik. Dinamikus tárkiosztás esetén a rendszer végzi a cím hozzásrendelést.

A C nyelvnek vannak aritmetikai és származtatott típusai. Az aritmetikai típusokhoz tartoznak az integrális típusok: egész(int,short,long), karakter(char),felsorolásos és valós(float, double, long double). A származtatott típusokhoz tartozik a tömb, függény, mutató, struktúra, union és vannak a void típusok. Az aritmetikai típusok az egyszerű, a származtatottak pedig az összetett típusok. Nincs logikai típus. A hamis az int 0, minde más inthez rendelt érték igaznak minősül. Az unsigned típusminősítő nem előjeles ábrázolást, a signed pedig előjeles ábrázolást jelöl. A struktúra egy fix szerkezetú rekord. A void típus tartománya üres,nincsenek műveletei. A const megagásával nevesített konstanst deklarálunk. Saját típus definiáláshoz TYPEDEF-el lehetséges, de ez csak a típus nevét adja meg nem hoz létre új típust. Struktúra deklarásása STRUCT-al lehetésges, union pedig UNION-nal. A C csak egydimenziós tömböket kezel. Deklaráláshoz az indexek darabszámát kell megadni, ami 0-tól darabszám-1-ig fut. A C a tömböt mutató típusként kezeli. A tömb neve a tömb első elemét címzi. Van automatikus deklaráció, int egész típus lesz ha egy névhez nem adunk visszatérési típust.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

10.3. Programozás

[BMECPP]

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.

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. 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.