

# Univerzális programozás

---

**Írd meg a saját programozás tankönyvedet!**

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

Copyright © 2019 Dr. Bátfai Norbert

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

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

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

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

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

## COLLABORATORS

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Halász, Dávid	2019. március 11.	

## 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 <a href="https://gitlab.com/nbatfai/bhax">https://gitlab.com/nbatfai/bhax</a> 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

<b>I. Bevezetés</b>	<b>1</b>
<b>1. Vízió</b>	<b>2</b>
1.1. Mi a programozás?	2
1.2. Milyen doksikat olvassak el?	2
1.3. Milyen filmeket nézzek meg?	2
<b>II. Tematikus feladatok</b>	<b>3</b>
<b>2. Helló, Turing!</b>	<b>5</b>
2.1. Végtelen ciklus	5
2.2. Lefagyott, nem fagyott, akkor most mi van?	6
2.3. Változók értékének felcserélése	8
2.4. Labdapattogás	9
2.5. Szóhossz és a Linus Torvalds féle BogomIPS	11
2.6. Helló, Google!	13
2.7. 100 éves a Brun tétel	16
2.8. A Monty Hall probléma	17
<b>3. Helló, Chomsky!</b>	<b>20</b>
3.1. Decimálisból unárisba átváltó Turing gép	20
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	21
3.3. Hivatkozási nyelv	21
3.4. Saját lexikális elemző	22
3.5. l33t.1	23
3.6. A források olvasása	25
3.7. Logikus	27
3.8. Deklaráció	27

<b>4. Helló, Caesar!</b>	<b>30</b>
4.1. int *** háromszögmátrix	30
4.2. C EXOR titkosító	30
4.3. Java EXOR titkosító	30
4.4. C EXOR törő	30
4.5. Neurális OR, AND és EXOR kapu	31
4.6. Hiba-visszaterjesztéses perceptron	31
<b>5. Helló, Mandelbrot!</b>	<b>32</b>
5.1. A Mandelbrot halmaz	32
5.2. A Mandelbrot halmaz a std::complex osztállyal	32
5.3. Biomorfok	32
5.4. A Mandelbrot halmaz CUDA megvalósítása	32
5.5. Mandelbrot nagyító és utazó C++ nyelven	32
5.6. Mandelbrot nagyító és utazó Java nyelven	33
<b>6. Helló, Welch!</b>	<b>34</b>
6.1. Első osztályom	34
6.2. LZW	34
6.3. Fabejárás	34
6.4. Tag a gyökér	34
6.5. Mutató a gyökér	35
6.6. Mozgató szemantika	35
<b>7. Helló, Conway!</b>	<b>36</b>
7.1. Hangyaszimulációk	36
7.2. Java életjáték	36
7.3. Qt C++ életjáték	36
7.4. BrainB Benchmark	37
<b>8. Helló, Schwarzenegger!</b>	<b>38</b>
8.1. Szoftmax Py MNIST	38
8.2. Szoftmax R MNIST	38
8.3. Mély MNIST	38
8.4. Deep dream	38
8.5. Robotpszichológia	39

<b>9. Helló, Chaitin!</b>	<b>40</b>
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .	40
9.2. Weizenbaum Eliza programja . . . . .	40
9.3. Gimp Scheme Script-fu: króm effekt . . . . .	40
9.4. Gimp Scheme Script-fu: név mandala . . . . .	40
9.5. Lambda . . . . .	41
9.6. Omega . . . . .	41
 <b>III. Második felvonás</b>	 <b>42</b>
<b>10. Helló, Arroway!</b>	<b>44</b>
10.1. A BPP algoritmus Java megvalósítása . . . . .	44
10.2. Java osztályok a Pi-ben . . . . .	44
 <b>IV. Irodalomjegyzék</b>	 <b>45</b>
10.3. Általános . . . . .	46
10.4. C . . . . .	46
10.5. C++ . . . . .	46
10.6. Lisp . . . . .	46

## Ábrák jegyzéke

2.1. EXOR működése . . . . .	9
2.2. Monty Hall probléma . . . . .	17

DRAFT



# Előszó

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

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

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

## Hogyan forgasd

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

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

## Milyen nyelven nyomjuk?

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

## Hogyan nyomjuk?

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

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

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

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



#### A DocBook XML 5.1 új neked?

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

## **I. rész**

### **Bevezetés**

# 1. fejezet

## Vízió

### 1.1. Mi a programozás?

### 1.2. Milyen doksikat olvassak el?

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

### 1.3. Milyen filmeket nézzek meg?

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

## **II. rész**

### **Tematikus feladatok**

**Bátf41 Haxor Stream**

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

---

DRAFT

## 2. fejezet

# Helló, Turing!

### 2.1. Végtelen ciklus

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

Megoldás forrása: [https://gitlab.com/davidhalasz/bhax/tree/master/attention\\_raising/Source/vegtelen](https://gitlab.com/davidhalasz/bhax/tree/master/attention_raising/Source/vegtelen)

Végtelen ciklusnak nevezzük azt az utasítást, ami örökké futna, ha nem állítanánk le a programot. A végtelen ciklus nem minden esetben jelent rossz dolgot, csak akkor, ha olyan feladat során keletkezik, amelynek végrehajtása nem igényel ilyen ciklusos algoritmust. Ilyen ciklus lehet például az alábbi kódrészlet:

```
Program vegtelen.c {  
  
    int main() {  
        int i = 0;  
        while (i<=0) {  
            i--;  
            printf("\n vegtelen ciklus");  
        }  
    }  
}
```

A program a while utáni zárójelben lévő kifejezést értékeli ki. Mivel a az „i” változó értéke egyenlő vagy kisebb a nullával ezért ez a feltétel igaz lesz és a program lefut. Ennek eredményeképpen az i változó értéke eggyel kisebb lesz és újból indul az egész. Mivel az eredmény sose lesz nullánál nagyobb, ezért a feltétel mindig igaz lesz, azaz végtelen ciklusba kerül. Vannak esetek, amikor szükség van végtelen ciklusra, ekkor érdemes olyat programkódot használni, amiről lehet látni, hogy ezt direkt így akartuk és nem pedig szoftverhiba. Ilyen ciklus az alábbi példa, amikor nem töltjük ki a for ciklus fejlécét:

```
Program vegtelen_for {  
  
    int main() {  
        for(;;);  
    }  
}
```

```
}
```

Ez a végtelen ciklus 1 magot használ 100%-on. Ha több magot szeretnénk dolgoztatni, akkor hozzá kell adni a `#pragma omp parallel-t`, tehát a végleges kódrészlet így fog kinézni:

```
Program vegtelen_pragma.c {  
  
    int main() {  
        #pragma omp parallel  
        for(;;);  
    }  
  
}
```

Ezzel az elhelyezett OpenMP alapú parallel régiók segítségével olyan függvényt hívunk elő, amellyel könnyen kihasználhatjuk vele egy számítógép processzorainak teljes számítási kapacitását. Fontos megjegyezni, hogy fordítás során hozzá kell adni `-fopenmp` kódrészletet is, ezért végül így fog kinézni: `gcc vegtelen_pragma.c -o vegtelen_pragma -fopenmp` Ha olyan programra van szükség, ahol csak 0%-ban dolgoztatnak meg egy magot, akkor a `sleep(1)` függvényt kell használnunk:

```
Program vegtelen_sleep.c {  
  
    int main() {  
        for(;;) {  
            sleep(1);  
        }  
    }  
  
}
```

A `sleep()` függvény hatására a program a zárójelben megadott értékű másodpercre megáll, alszik. Ha a zárójelben nincs megadva argumentum, akkor meghatározatlan időre alszik. Ezzel érhető el, hogy a mag ne legyen 100%-ban kihasználva.

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

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

Megoldás videó:

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

```
Program T100  
{  
  
    boolean Lefagy(Program P)  
    {  
        if(P-ben van végtelen ciklus)
```



```
    return true;
  else
    return false;
}

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

## 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](https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk)

Megoldás forrása: [https://gitlab.com/davidhalasz/bhax/blob/master/attention\\_raising/Source/csere.c](https://gitlab.com/davidhalasz/bhax/blob/master/attention_raising/Source/csere.c)

Két változó értékének felcserélése segédváltozó nélkül a legegyszerűbben XOR-ral oldható meg:

```
Program csere.c {  
  
    int main(){  
  
        int x = 3;  
        int y = 5;  
  
        x ^= y; // x = 6  
        y ^= x; // y = 3  
        x ^= y; //x = 5  
  
        printf("x:%d y:%d\n", x, y);  
  
        return 0;  
    }  
}
```

Az XOR lényege, hogy a számok bináris alakjainak különbségét vizsgálja meg. Nézzük meg sorról sorra a kódot:

3 bináris alakja: 0011

5 bináris alakja: 0101

$x \oplus y$  esetén  $x$  értéke 0110 lesz, mert a 2 szám binárisan összehasonlítva, ha az adott biten a két szám értéke megegyezik, akkor 0-át kapunk, ha különbözőek, akkor pedig 1-et. Az eredményt az alábbi táblázat mutatja, ahol az eltérést pirossal, az egyezést zölddel jelöltem:

	BINÁRIS ALAK			
3	0	0	1	1
5	0	1	0	1
Összehasonlítás	0	1	1	0

2.1. ábra. EXOR működése

Ezután  $y \hat{=} x$  esetén (ahol  $y = 0101$  és  $x = 0110$ )  $y$  értéke  $0011$  lesz, ami a 3-as szám bináris alakjának felel meg. Ezután már csak a  $x$  értékét kell helyesen megadni, azaz  $x \hat{=} y$  esetén (ahol  $x = 0110$  és  $y = 0011$ )  $x$  értéke binárisan  $0101$  lesz, ezzel pedig meg is kapjuk az 5-ös számot, azaz  $y$  és  $x$  értékét sikeresen felcseréltük segédváltozó nélkül.

## 2.4. Labdapattogás

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

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

Megoldás forrása: [https://gitlab.com/davidhalasz/bhax/tree/master/attention\\_raising/Source/labdapattogas](https://gitlab.com/davidhalasz/bhax/tree/master/attention_raising/Source/labdapattogas)

Először nézzük meg a labdapattogatást if-ekkel. Az `initscr()` curses módba kapcsolja a terminált. Ez törli a képernyőt és fekete háttérrel ad, továbbá tárolja az ablak adataival kapcsolódó információkat. Először fontos, hogy az ablakkal és a lépésekkel kapcsolatos változókat meghatározása. Az `int x`-el és az `int y`-al adjuk meg az aktuális pozíciót, mely az  $x$  és az  $y$  tengelyen fog elhelyezkedni. Az `int xnov` és az `int ynov` lesznek a lépésközök, azaz hogy mennyit lépjen majd előre a labda az  $x$  és  $y$  tengelyen. Az `int mx`-ben és az `int my`-ben pedig az ablak méreteit fogjuk majd eltárolni.

Program `labdaif.c`

```
{  
  
    WINDOW *ablak;  
    ablak = initscr ();  
  
    int x = 0;  
    int y = 0;  
  
    int xnov = 1;  
    int ynov = 1;  
  
    int mx;  
    int my;  
    ...  
}
```

Mivel azt szeretnénk, hogy a program folyamatosan, megállás nélkül fusson, ezért végtelen ciklusra lesz szükségünk, amit a `for(;;)` ciklussal érhetünk el. Ezen belül kell program számára megadni, hogy pontosan mekkora lesz a képernyő. Ezt a `getmaxyx()` függvény segítségével érhetjük el, ahol az ablak-ban tárolt értékeket elmenti az `mx` és `my` változóba.

Megadhatjuk továbbá, hogy a képernyő ciklusonként törölje az előző "labda" nyomát a `clear()` függvénnyel. Ha ezt kihagyjuk, akkor a képernyőn folyamatosan látni fogjuk a labda eddigi helyzetét is, azaz folyamatosan "csíkot húz" maga után. Ennek a függvénynek mindig az `mvprintw()` előtt kell szerepelnie, ellenkező esetben csak üres ablakot fogunk látni. Ez utóbbi függvény segítségével adhatjuk meg, hogy az "O" karaktert, hol jelenítse meg a képernyőn. Majd az a következő pár sorban az `x` és `y` értékeket 1-el növeljük.

```
Program labdaif.c
{
    ...
    getmaxyx ( ablak, my , mx );

    clear ();
    mvprintw ( y, x, "O" );

    x = x + xnov;
    y = y + ynov;
    ...
}
```

Ezután következik annak meghatározása, hogy a labda elérte-e a képernyő szélét. Ha igen, akkor az értékeket meg kell szorozni -1-el és így a labda "visszapattog". Itt kell használnunk az `if()` függvényeket. Ha itt megnézzük az első sort, akkor láthatjuk, hogy ha az `x` értéke nagyobb mint az ablak `mx-1` értéke, akkor a labda útját, azaz az `xnov` értékét meg kell szorozni mínusz 1-gyel. Értelemszerűen ugyanígy kell meghatározni a többit is a hozzá tartozó változókkal.

```
Program labdaif.c
{
    ...
    if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
        xnov = xnov * -1;
    }
    if ( x<=0 ) { // elerte-e a bal oldalt?
        xnov = xnov * -1;
    }
    if ( y<=0 ) { // elerte-e a tetejet?
        ynov = ynov * -1;
    }
    if ( y>=my-1 ) { // elerte-e a aljat?
        ynov = ynov * -1;
    }
    ...
}
```

Végül az `usleep()` függvénnyel adjuk meg a labda sebességét. Jelenleg 1 másodperc van megadva, ami átszámolva 100000 mikroszekundum. Ha ez letelik, a ciklus újraindul.

Ha a **labdapattogást if nélkül** szeretnénk megoldani, akkor maradékos osztást kell használni, ami az alábbi kódrészletben látható:

```
Program labda.c
{
    ...

    for (;;)
    {
        getmaxyx (ablak, my, mx);
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;

        yj = (yj - 1) % my;
        yk = (yk + 1) % my;

        clear ();

        mvprintw (abs (yj + (my - yk)),
                  abs (xj + (mx - xk)), "o");

        refresh ();
        usleep (100000);

    }

    ...
}
```

## 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: [https://youtu.be/9KnMqrkj\\_kU](https://youtu.be/9KnMqrkj_kU), <https://youtu.be/KRZlt1ZJ3qk>, .

Megoldás forrása: [https://gitlab.com/davidhalasz/bhax/tree/master/attention\\_raising/Source/bogomips](https://gitlab.com/davidhalasz/bhax/tree/master/attention_raising/Source/bogomips)

A szóhosszt az alábbi programmal tudjuk kiírni:

```
Program szohossz.c
{
    int h = 0;
    int n = 0x01;
    do
        ++h;
```

```
while (n <= 1);  
printf ("A szóhossz ezen a gépen: %d bites\n", h);  
return 0;  
  
}
```

A program elindítása után a terminálban olvashatjuk, hogy a szóhoz ezen a gépen 32 bites.

A `0x01` kifejezés hexadecimális (16-os számrendszer) számot jelent, mely `0x` prefixszel kezdődik. A `do` és `while` páros pedig azt jelenti, hogy "csináld ezt, amíg a feltétel teljesül". Van még egy fontos eleme a programnak, ami a `while` utáni zárójelben szerepel, ez pedig a bittologató (vagy más néven shiftelő) operátor. Ennek a jele: `<<` vagy `>>` lehet. Jelen példában a `<<` operátort használjuk, mert a biteket balra szeretnénk tolni egyesével, egészen addig, amíg csupa nullát nem kapunk. A bitek léptetését így kell elképzelni:

Bitek léptetése

```
00000001  
00000010  
00000100  
00001000  
00010000  
00100000  
01000000  
10000000  
00000000
```

A program végül azért ír 32 bitet végeredményül, mert ennyiszer fut le a ciklus, amíg csupa nulla nem lesz.

A **Bogomips** egy Linux operációs rendszeren nyújtott mérési program, amely relatív módon jelzi, hogy a számítógép processzor milyen gyorsan fut, azaz a processzor hányszor halad át egy adott cikluson egy meghatározott idő alatt. A programot Linus Torvalds írta. Most ennek a programnak a működését fogom bemutatni.

A program elején látható, hogy először 2 változót deklaráltunk: az egyik a `loops_per_sec`, melynek értéke 1, a másik pedig a `ticks`, ami a `while` cikluson belül a `clock()` függvényt rendeljük hozzá. Ez a `clock()` függvény méri, hogy eddig mennyi processzor idő telt el. A `while` ciklus zárójelében a `loops_per_sec` értéke a `<=` shiftelő operátor miatt 2 hatványaival fog számolni. Ezután meghívjuk a `delay()` függvényt, aminek mindig átadom a ciklus változó értékét. Nagyon gyorsan növekedik, mindig hatványodik és ez a `delay()` függvény azt csinálja, hogy 0-tól egyesével növelgetve elszámol a `loops` értékig. Ezt követően újra lekérem a `ticks` értékét, mégpedig úgy, hogy a processzoridőből kivonjuk az előző időt. Lényegében ezzel kapjuk meg, hogy mennyi idő telt el.

Program `bogomips.c`

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

```
while (loops_per_sec <= 1)
{
    ticks = clock ();
    delay (loops_per_sec);
    ticks = clock () - ticks;

    ...
}
```

Az `if` függvénnyel megvizsgáljuk, hogy a `ticks` értéke nagyobb vagy egyenlő-e a `CLOCKS_PER_SEC` értékénél. Ez utóbbi értéke mindig 1.000.000. Ha ez a feltétel teljesül, akkor kiszámoljuk, `loops_per_sec` értékét. Itt arra vagyunk kíváncsiak, hogy milyen ciklusérték tartozna hozzá, ha nem 2 hatványaival mennénk. Tehát, hogy milyen hosszú ciklust képes végrehajtani a gép. Kiíratásnál még elosztjuk az adatokat, hogy jobban olvasható legyen a kapott eredmény.

Program `bogomips.c`

```
...
    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;
    }
}
...
```

## 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:

A PageRank a Google internetes keresőmotor alapja, amit a Google alapítói, Larry Page és Sergey Brin fejlesztettek ki 1998-ban a Stanford Egyetemen. A PageRank a Google bejegyzett védjegye, legfontosabb tulajdonsága, hogy képes elemezni az oldalak közötti kapcsolatokat és ezáltal relevánsabb találatokat tud visszaadni. A forrásban látható program egy 4 honlapból álló hálózatra számolja ki a négy lap PageRank-ét.

Először is szükségünk lesz egy 4x4-es mátrixra, mivel 4 honlapunk van. Ezt az `L` változóban deklaráljuk. Látható, hogy oszloponként vannak kiszámolva az eredmények. Minden oldalnak egy egységnyi szavazata van, amit szétoszt azok között az oldalak között, amikre hivatkozik. Tehát ha az `A` oszlopban csak 1

honlapra van hivatkozás, ezért oda 1-es kerül. A második oszlopban lévő honlap 2 oldalra is hivatkozik ezért ezt egyenlő arányban osztja szét, tehát mindkét hivatkozott oldal fél-fél szavazatot kap és ez így megy tovább a többi oszlopnál is. Ezután meghívjuk a `pagerank()` függvényt, aminek átadjuk az `L` változóban tárolt mátrixot.

Program `pagerank.c`

```
...

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}
    };

    pagerank(L);

    printf("\n");

    return 0;
}
```

A `pagerank()` függvényben a `PRv[]` tömbben lesznek eltárolva az `L` mátrixból kapott értékek. A `PR[]` tömbben pedig majd a számítás során kapott eredményeket. A `for` függvény négyszer fog lefutni, mert összesen ennyi honlapunk van, ezeket össze fogja adni, míg végül egy oszlopból álló 4 soros mátrixot kapunk, aminek eredménye bekerül a már említett `PR[]` tömbbe.

Program `pagerank.c`

```
...

void pagerank(double T[4][4]){
    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] = PR[i] + T[i][j]*PRv[j];
            }
        }

        if (tavolsag(PR,PRv,4) < 0.0000000001)
            break;

        for (i=0; i<4; i++){
```



```
        PRv[i]=PR[i];
    }
}
kiir (PR, 4);
}

...
```

Ezután if függvénnyel megvizsgáljuk, hogy a távolság() függvény által adott eredmény kisebb-e 0.0000000001-val. Az ezt a függvényt az alábbi kódrészletben lehet látni:

Program pagerank.c

```
...

double
tavolsag (double PR[], double PRv[], int n){

    int i;
    double osszeg=0;

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

...
```

Ezután már csak kiíratjuk a PR[] tömbben tárolt eredményeket egyesével, amit a kiir() függvény segítségével tesszük meg.

Program pagerank.c

```
...

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

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

...
```

## 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/davidhalasz/bhax/blob/master/attention\\_raising/Primek\\_R/stp.r](https://gitlab.com/davidhalasz/bhax/blob/master/attention_raising/Primek_R/stp.r)

A számelmélet egyik legfontosabb tétele a Brun-tétel, mely a prímszámokkal kapcsolatos elméleteket gondolja tovább. Prímszámoknak nevezzük azokat a számokat, amelyeknek trivális osztói vannak, azaz csak önmagával és 1-el oszthatók. A Brun-tétel szerint végtelen sok olyan  $p$  prím létezik, amire  $p+2$  is prím (pl.: 3,5; 5,7; 7,9; stb). Az ilyen prímpárokat, amiknek a különbsége 2, ikerprímeknek nevezzük. Az ilyen ikerprímsejtések bizonyítása vagy cáfolata jelenleg meghaladja a matematika eszközeit. Az ikerprímek, ha végtelen sokan vannak is, nagyon ríktán fordulnak elő a prímek között. Ugyanis míg a prímek reciprokaiból álló sor divergens, addig az ikerprímek reciprocai sora konvergens. Konvergensnek nevezzük azokat a sorokat, ha az elemek tartanak egy számhoz. Azt a számot pedig ahova eljut, a sor összegének nevezzük. A Brun tétel szerint ezt a sor összeget az ikerprímszámok reciprokaiknak összeadásával kapjuk meg  $[(1/3+1/5)+(1/5+1/7)+(1/7+1/9)+...]$  és megkapunk egy határértéket. Ezt a határértéket nevezzük Brun konstansnak. Most nézzük meg az alábbi programot, ami megpróbálja közelíteni a Brun konstans értékét:

```
Program stp.r
library(matlab)

stp <- 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 = stp)
plot(x,y,type="b")
```

A kód futtatásához Matlabra lesz szükség. A `primes()` függvény egy paramétert kér, az  $x$  helyére bármilyen számot írhatunk és az ott megadott számig fogja kiszámolni a prímeket.

A `diff` a `primes` vektorban lévő számok segítségével az egymást követő számok különbségét kapjuk meg egy vektorba rendezve.

Az `idx` változóban megkeressük azokat a számokat, ahol a `diff` egyenlő 2-vel. Ugyanis ezek lesznek majd az ikerprím párok.

A `t1primes` kiveszi az ikerprímpárok első tagját, majd a `t2primes` változóban az ikerprím pár első tagjához hozzáadunk 2-őt, és így megkapjuk az ikerprím pár második tagját is. Majd ezen ikerpárok reciprokösszegét a `rt1plust2` változóban tároljuk. Végül pedig a `sum()` függvénnyel ezeket a törteket összeadjuk.

Ezután már csak meg kell jelenítenünk a kapott eredményeket egy függvény segítségével. A jelen példában egy `seq()` függvényt fogjuk használni. Az `y` változóban az `sapply()` függvény használatával rendeljük hozzá az `x` érték adatait. Végül pedig a `plot()` függvénnyel rajzoljuk ki az eredményt.

## 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](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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/MontyHall_R)

A paradoxon egy az Egyesült Államokban az 1960-as években nagy sikerrel futott televíziós vetélkedő utolsó játékán alapszik. Nevét a show házigazdájáról kapta.

A probléma alaphelyzete a következő: A játékosnak mutatnak három zárt ajtót, melyek közül kettő mögött egy-egy kecske van, a harmadik pedig egy vadonatúj autót rejt. A hangulat fokozása érdekében a választást, illetve a felnyitást egy kicsit megbonyolították. A játékos kiválaszt egy ajtót, de mielőtt ezt kinyitná, a műsorvezető a másik két ajtó közül kinyit egyet, mely mögött biztos nem az autó van (természetesen a showman a kezdettől tisztában van azzal, hogy melyik ajtó rejt az autót). Ezt követően megkérdezi a már amúgy is ideges vendéget, hogy jól meggondolta-e a választását, vagyis nem akar-e váltani. A játékos dönt, hogy változtat, vagy sem, végül feltárul az így kiválasztott ajtó, mögötte a nyereménnyel. A paradoxon kérdése az, hogy érdemes-e változtatni, illetve van-e ennek egyáltalán jelentősége.

Vizsgáljuk meg a Monty Hall paradoxont: kezdetben tehát 1:3 az esélye, hogy bármelyik ajtó mögött ott lehet a főnyeremény, az autó. Ezután a műsorvezető kinyit egy olyan ajtót, ami mögött nincs ott az autó. Ekkor még mindig 1:3 lesz az esélye annak, hogy az általunk választott ajtó mögöttrejtőzik a kocs, és 2:3 hogy mégse ott. Így tehát 2:3 lesz annak is az esélye, hogy a másik ajtó mögött találjuk meg a főnyereményt, ezért jó, ha váltunk. Az alábbi táblázat bemutatja a nyerési esélyeinket:

1. ajtó	2. ajtó	3. ajtó	Monty kinyitja	Ha váltunk
kecske	kecske	autó	a 2. ajtót	nyerünk
kecske	autó	kecske	a 3. ajtót	nyerünk
autó	kecske	kecske	a 2. vagy a 3. ajtót	vesztünk

2.2. ábra. Monty Hall probléma

Láthatjuk, hogy ha mindig az első ajtót választjuk, és változtatunk a döntésünkön, akkor a három esetből kétszer nyerünk.

Most pedig nézzünk meg egy ezzel kapcsolatos R szimulációt. Először is megadjuk, hogy mennyi legyen a kísérletek száma (1000000). A kísérlet változóban deklaráljuk, hogy `sample()` függvénnyel, hogy

1-től 3-ig, generáljon random számokat, mégpedig annyiszor, ahány a `kiserletek_szama` változó értéke. A `replace=T` engedélyezi, hogy legyen ismétlődés a számok között. Ide kerül tehát, majd az, hogy hol lesz a nyeremény. ezután megint ugyanezt a kódrészletet adjuk meg a `jatekos` változónak is, mely a játékos választásait fogja tárolni. Majd a `musorvezeto` értékéhez azt adjuk meg, hogy mennyi a `kiserletek_szama` mérete.

Program mh.r

```
kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)
...
```

A következő `for` ciklus annyiszor fog lefutni, ahány a `kiserletek_szama`. Azon belül, ha a `kiserlet` adott eleme megegyezik a `jatekos` adott elemével, azaz eltalálta a helyes ajtót, akkor a `kiserlet` tömbből kivesszük az egyezést és így lesz elmentve az adat a `mibol` változóban. Ezt a `setdiff()` függvénnyel érhetjük el. Ellenkező esetben (`else`) mind a `kiserlet[]` és a `jatekos[]` tömbből ki kell venni az adott értéket. Ezután állítjuk össze a `musorvezeto[]` értékeit a `mibol` és a `sample()` függvény segítségével.

Program mh.r

```
...

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)]

}

...
```

A `nemvaltoztatasesnyer` tömbbe kerülnek azok az adatok, hogy a `kiserlet` és a `jatekos` értékeit összehasonlítva hol találhatók egyezések. Ehhez a `which()` függvényt használjuk. A `valtoztat` vektorban megadjuk, hogy a változó értéke azonos legyen a `kiserletek_szama`-val.

Program mh.r

```
...

nemvaltoztatasesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)
```

...

Ezután megint for ciklus jön, megint ugyanannyiszor, ahány a `kiserletek_szama`. Dekráljuk a `holvalt` tömböt, mégpedig úgy, hogy a `setdiff()` függvénnyel kivesszük azokat az értékeket, amik egyeznek az adott indexben a `musorvezeto` és a `jatekos` értékeivel.

Program `mh.r`

...

```
valtoztatesnyer = which(kiserlet==valtoztat)
```

```
sprintf("Kiserletek szama: %i", kiserletek_szama)
```

```
length(nemvaltoztatesnyer)
```

```
length(valtoztatesnyer)
```

```
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
```

```
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Ha ezzel megvagyunk, akkor a `valtoztatesnyer` változót ugyanúgy adjuk meg, ahogy eddig, majd kiíratjuk az adatokat.

## 3. fejezet

# Helló, Chomsky!

### 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfiájával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása: [https://gitlab.com/davidhalasz/bhax/blob/master/attention\\_raising/Source/Chomsky/convert.c](https://gitlab.com/davidhalasz/bhax/blob/master/attention_raising/Source/Chomsky/convert.c)

A decimális számrendszer, vagy más néven tízes számrendszer egy 10 elemből álló halmaz, melynek első tagja a 0, az utolsó tagja pedig a 9. Ha ezt át szeretnénk váltani unáris, azaz egyes számrendszerbe, akkor a végeredményként egy csupa 1-esekből álló értéket kapunk. Az 1 csupán szimbólum, lehet helyettesíteni bármilyen más szimbólummal, a lényeg, hogy az N számot az általunk választott szimbólum N-szeri ismétlésével ábrázoljuk.

Ha egy decimálisból unárisba átváltó programot szeretnénk írni, akkor egy olyan ciklust kell létrehozni, ami mindig annyiszor írja ki az egyest, ahány számot adtunk meg. Ha az adott szám nulla, akkor eredményként nem fogunk kapni semmit, mivel a nullát nem képes egyes számrendszerben ábrázolni.

```
#include <stdio.h>

int main()
{
    int a;
    int count = 0;

    printf("Adj meg egy természetes számot\n");
    scanf("%d", &a);
    printf("A beírt szám unáris alakban:");

    for (int i = 0; i < a; ++i) {
        printf("1");
        count++;
    }

    printf("\n");
    return 0;
}
```

```
}
```

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

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

Megoldás videó:

Megoldás forrása:

A Noam Chomsky nevéhez fűződő úgynevezett Chomsky-féle hierarchia 1-es típusa definiálja a környezetfüggő nyelvtanokat. Kétféle szabály létezik:

- $\alpha A \gamma \rightarrow \alpha \beta \gamma$  alakú,  
ahol  $A \in N$ ,  $\alpha, \gamma \in (NUT)^*$ ,  $\beta \in (NUT)^+$ . Tehát itt az  $A$  nemterminális,  $\alpha$  pedig egy nemterminálisokból és terminálisokból álló üres vagy nem üres szó. A jobb oldalon álló  $\beta$  és  $\gamma$  viszont már nemterminális és terminálisokból álló szavak.
- $S \rightarrow \epsilon$  alakú,  
ahol az  $S$  kezdőszimbólum nem szerepel egyetlen szabály jobb oldalán sem.

Egy példa: Legyen  $G = (\{S, A, B, C\}, \{a, b, c\}, S, \{S \rightarrow \lambda, S \rightarrow abc, S \rightarrow aABC, A \rightarrow aABC, A \rightarrow aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\})$ .

Ekkor bizonyítható, hogy  $G$  az  $L(G) = \{a^n b^n c^n \mid n \geq 0\}$  nyelvet generálja.

### 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://gitlab.com/davidhalasz/bhax/blob/master/attention\\_raising/Source/Chomsky/szabvany.](https://gitlab.com/davidhalasz/bhax/blob/master/attention_raising/Source/Chomsky/szabvany.)

A BNF (Backus-Naur Form) egy olyan metanyelv, melyek segítségével szabályok alkothatók meg. Legfőbb célja a programozási nyelvek szintaxisának leírása. Főbb részei:

- $\langle \text{név} \rangle$

Olyan fogalmak tartoznak ide, mint pl.: azonosító, betű, törtszám, prízszám, stb. Tehát bármi lehet. Nem terminális elem.

- $::=$

Ez fogja elválasztani a szabály bal- és jobb oldalát.

- A definiálandó nyelv karakterkészlete, azaz a terminálisok. Az elválasztó jel jobb oldalán helyezkedik el.

Ezek alapján néhány C utasítások BNF-ben:

```
//feltételes utasítás
if (<kifejezés>)
    <utasítás>
else if (<kifejezés>)
    <utasítás>
else (<kifejezés>)
    <utasítás>

//while utasítás
while (<kifejezés>)
    do <utasítás>
```

Most pedig jöjjön egy olyan kódcsipet, amely c89-cel nem, de c99-el működik:

```
int main()
{
    //comment
    return 0;
}
```

A fenti kódot ha c89-cel fordítjuk le, akkor az alábbi hibaüzenetet fogunk kapni:

```
gcc -std=c89 szabvany.c -o szabvany

szabvany.c: In function 'main':
szabvany.c:3:5: error: C++ style comments are not allowed in ISO C90  ↵
    //comment
    ^
szabvany.c:3:5: error: (this will be reported only once per input file)  ↵
```

Ha pedig c99-cel, akkor gond nélkül működik:

```
gcc -std=c99 szabvany.c -o szabvany
```

### 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://gitlab.com/davidhalasz/bhax/blob/master/attention\\_raising/Source/Chomsky/lexer.l](https://gitlab.com/davidhalasz/bhax/blob/master/attention_raising/Source/Chomsky/lexer.l)

Az alábbi kódot .l végződésű fájlba kell menteni, mert a lex programmal fogunk dolgozni. A program segítségével lexikális szabályokból lexikális elemző programkódot fogunk generálni. Ehhez a következő parancsot kell megadnunk: `lex -o lexer.c lexer.l`, ami elkészíti számunka a c forráskódot. Ezután már csak a szokásos módon gcc-vel fordítjuk le, azzal a különbséggel, hogy a végéhez beírjuk az `-lfl` is: `gcc lexer.c -o lexer -lfl`.

```
%{
#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;
}
```

A fenti program 3 fő részből áll, amelyeket a `%%` szimbólum választja szét őket. Az első rész tartalmazza a headert és a változót. Itt bármilyen C kódot megadhatunk a `%{` és `%}` jelek között. Ezt a Flex módosítás nélkül fogja átmásolni a generálandó kódba.

A második rész 2 részből álló szabályokat tartalmaz: Először a reguláris kifejezésekkel kezdjük, utána jön a kapcsos zárójelbe zárva a C kód. A reguláris kifejezéseknél határozzuk meg a keresési szabályokat. Jelen esetben számmal kezdődött keresünk (`digit`), ami akár többször is előfordulhat (`*`). Ezt követően van egy zárójeles rész is, ami azt jelenti, hogy `????`. Az ezt követő C kódban adjuk meg, hogy majd a terminálba bevitt szöveget vizsgálja meg. Az `atof()` függvény segítségével a `string`-ből `double` lesz. Tehát ezekből a szabályokból létrejön majd a `yylex` nevű függvény és `return` esetén újra ide fog visszakerülni.

A program harmadik részében látható egy `yylex()` függvény, ami a fenti szabályokat hívja elő, továbbá kiírjuk az eredményt.

### 3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása: [https://gitlab.com/davidhalasz/bhax/blob/master/attention\\_raising/Source/Chomsky/leet.l](https://gitlab.com/davidhalasz/bhax/blob/master/attention_raising/Source/Chomsky/leet.l)

A leet nyelv lényege, hogy bizonyos betűket számokkal vagy ASCII karakterekkel szokták helyettesíteni. Innen jön az, hogy leet helyett l33t írunk. A mostani példában olyan programot mutatok be, ami az adott szövegben ezeket a betűket kicseréli a megfelelő számokkal, vagy karakterekkel. A megadott forráskódot

is lex programmal kell C-be fordítani. Vizsgáljuk meg a programot. Ahogy az előző feladatnál, úgy itt is 3 részből áll.

Az első részben definiáljuk a L33SIZE konstansszerű elemet. Ezt a #define segítségével oldjuk meg és a nevet mindig csupa nagybetűvel kell megadni. Ha a programban valahol beírjuk a megadott konstans nevet (L33SIZE), akkor a helyére fogja majd behelyettesíteni a hozzá tartozó szöveget. Jelen esetben ehhez a változóhoz meg van adva egy char c és egy hozzá tartozó char \*leet[4] tömb, ebből a kettőből fog összeállni a l337d1c7[] tömb, ahol az előbbi lesz majd a kicserélendő betű, utóbbi pedig az, hogy mire lehet majd kicserélni (4 választási lehetőség van).

```
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))

struct cipher {
    char c;
    char *leet[4];
} l337d1c7 [] = {

    {'a', {"4", "4", "@", "/-\\\"}},
    ...
}
```

Ha az egyes betűkhöz hozzárendeltük a megfelelő számokat vagy karaktereket. Akkor a második részben fogjuk megadni C-ben, hogy hogyan történjen a karaktercsere:

```
...
. {

    int found = 0;
    for(int i=0; i<L337SIZE; ++i)
    {

        if(l337d1c7[i].c == tolower(*yytext)) //Megegyezik-e a két karakter?
        {

            int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0)); //random szám ←
                generálása 1 és 100 között

            if(r<91)
                printf("%s", l337d1c7[i].leet[0]);
            else if(r<95)
                printf("%s", l337d1c7[i].leet[1]);
            else if(r<98)
                printf("%s", l337d1c7[i].leet[2]);
            else
                printf("%s", l337d1c7[i].leet[3]);

            found = 1;
            break;
        }
    }
```

```
    }

    if(!found)
        printf("%c", *yytext);

    }
    ...
```

A kapcsos zárójel elején egy pont van, ami azt jelenti, hogy minden egyes karaktert vizsgáljon meg a szövegben. Ez lesz tehát a reguláris kifejezés. A C részben látható, hogy a `for` ciklusban előhívjuk a `L33SIZE` nevű konstanst, tehát ide fogja behellyettesíteni azt, amit már az első részben megadtunk.

Szükségünk lesz a `tolower()` függvényre is, mert ha nagybetű van a szövegben, akkor azt nem fogja számításba venni a program, ezért ezzel a függvénnyel minden betűt kisbetűvé alakítunk át. Ezután a kapott random szám alapján cseréljük ki az adott betűket. Tehát például az "a" betűt kell kicserélni, és `r` értéke 97, akkor ez azt jelenti, hogy az "a" betű "@"-ra lesz kicserélve, mivel az if-else utasításban itt fog teljesülni:

```
...
    else if(r<98)
        printf("%s", l337d1c7[i].leet[2]);
    ...
```

Ez azt jelenti, hogy `l337d1c7[i]` tömbön belül a `leet[2]` tömb 3. eleme legyen kiválasztva, ami a "@". Előfordul azonban, hogy a két karakter nem egyezik meg a vizsgálat során (`if(!found)`), ezért ilyenkor az eredeti karaktert írjuk ki.

### 3.6. A források olvasása

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

```
if(signal(SIGINT, jelkezeslo)==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 `jelkezeslo` 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 meggyránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

Mielőtt átnéznénk a kódcsipeteket, előbb fontosnak tartom megemlíteni, hogy a kódokat úgy vizsgáljuk meg, hogy feltételezzük, az adott változók, függvények értéke már előzetesen deklarálva vannak.

i.

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

Ez a kódcsipet az előző példa ellentettje. Azt jelenti, hogy ha a SIGINT jel kezelése nem volt figyelmen kívül hagyva, akkor ezután se legyen figyelmen kívül hagyva.

ii.

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

A for ciklus 5-ször hajtódik végre. Az i változó értéke 0-tól 4-ig fog változni. A ++i azt jelenti, hogy i értékét előbb növeljük és ez lesz az új érték.

iii.

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

Ez a ciklus ugyanúgy fog végrehatjtódni, mint az előző. Itt is ugyanúgy 0-tól 4-ig fog számolni. Az i++ pedig azt jelenti, hogy i értéke marad a régi, majd utána növeli 1-el.

iv.

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

A for ciklus kicseréli a tömb elemeit. A tömb első eleme marad a régi, az azt követő elemek értéke azonban már rendre 0, 1, 2, 3 lesz. Ha a tömb mérete 5-nél nagyobb, akkor a többi értéket figyelmen kívül hagyja, mivel a for ciklus csak 5-ször fut le. Ezért tehát a program bár gond nélkül le fog futni, de bugos.

v.

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

A for ciklusnak itt 2 felétele van, ugyanis látható, hogy van egy && (ÉS) logikai operátor is. A ciklus addig fog futni, amíg i értéke kisebb mint n. A logikai operátor jobb oldalán nincs relációs operátor megadva, ezért ez a program bugos. A változók előtti \* jel azt jelenti, hogy egy adott tárterületre mutatnak a változók értékei.

vi.

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

Az printf() függvény segítségével írjuk ki a zárójelben megadott paraméterek alapján. Jelen esetben a cél az lenne, hogy kiíratjuk az f() függvények visszatérési értékét, de ez a kód bugos, ugyanis az f() függvény két paramétert kér, de ezek kiértékelési sorrendje nincs meghatározva.

vii.

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

Ez a kód már helyes. Az f() függvény egy paramétert kér, ami jelen esetben az "a" lesz, így ennek az értéke kerül kiíratásra.

viii.

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

Ez a kód is le fog fordulni, csak itt az f() függvénynél &a argumentum van megadva, ami azt jelenti, hogy egy terület memóriacímére mutató értéket ad paraméterül.

### 3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```

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

```

```

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

```

```

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

```

```

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

```

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/MatLog\\_LaTeX](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/MatLog_LaTeX)

Megoldás videó: <https://youtu.be/ZexiPy3ZxsA>, [https://youtu.be/AJSXOQFF\\_wk](https://youtu.be/AJSXOQFF_wk)

Végtelen sok prímszám van.

```

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

```

Végtelen sok ikerprímszám van.

```

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

```

Véges sok prímszám van.

```

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

```

Véges sok prímszám van.

```

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

```

### 3.8. Deklaráció

Megoldás videó:

Megoldás forrása:

- egész

```
int egesz = 5;
```

- egészre mutató mutató

```
*mutato = &eges;
```

- egész referenciája

```
int &r_egesz = egesz;
```

- egészek tömbje

```
int tomb[3] = {1, 2, 3};
```

- egészek tömbjének referenciája (nem az első elemé)

```
int (&r_tomb)[3] = tomb;
```

- egészre mutató mutatók tömbje

```
int *m_tomb[3];
```

- egészre mutató mutatót visszaadó függvény

```
int* e_mutato = f(mutato);
```

- egészre mutató mutatót visszaadó függvényre mutató mutató

```
int* (*f_mutato)(int*) = f;
```

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

```
int (*getInt (int a)) (int b, int c);
```

- 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;
```

Integer, azaz egész típusú változót, melynek változóneve: a

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

Az első sorban bevezetett a változóra mutató mutató lesz a \*b.

- ```
int &r = a;
```

Az a változó referenciája lesz az &r

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

Integer típusú, c változónevű tömb deklarálása, melynek mérete 5.

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

A c tömb referenciája.

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

Ez egy integer típusú tömbre mutató mutató.

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

Integer típusú h függvényre mutató mutató.

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

Egy egészre mutató mutatóval visszatérő függvényre mutató mutató.

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

Egy egészszel visszatérő, 2 egész paramétert váró függvényre mutató mutató.

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

függénymutató, mely egy egészet visszaadó, két egész paramétert váró függvényre mutató mutatót visszaadó, egészet kapó függvény.

## 4. fejezet

# Helló, Caesar!

### 4.1. int \*\*\* háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 4.4. C EXOR törő

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

Megoldás videó:



Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 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](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R)

Tanulságok, tapasztalatok, magyarázat...

## 4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 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](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 iteráció bejárta  $z_n$  komplex számokat!

---

Megoldás forrása:

Megoldás videó:

Megoldás forrása:

## 5.6. Mandelbrot nagyító és utazó Java nyelven

DRAFT

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása:

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

### 6.2. LZW

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

Megoldás videó:

Megoldás forrása:

### 6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

### 6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

## 6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

## 6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

## 7. fejezet

# Helló, Conway!

### 7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

## 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...

DRAFT

## 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](https://youtu.be/OKdAkI_c7Sc)

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/GIMP\\_Lisp/Chrome](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](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](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:

DRAFT

## **III. rész**

### **Második felvonás**

DRAFT

**Bátf41 Haxor Stream**

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

---

DRAFT

---

## 10. fejezet

# Helló, Arroway!

### 10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

### 10.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## **IV. rész**

### **Irodalomjegyzék**

### 10.3. Általános

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

### 10.4. C

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

### 10.5. C++

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

### 10.6. Lisp

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