

# Univerzális programozás

---

**Írd meg a saját programozás tankönyvedet!**

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

Copyright © 2019 Kinczel Gergő-János

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

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

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

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

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

**COLLABORATORS**

	<i>TITLE :</i> Univerzális programozás		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Bátfai, Norbert Ács Kinczel, Gergő	2019. május 9.	

**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
0.0.5	2019-04-02	Folyamatban.	trigarosa
0.0.6	2019-05-09	Befejezés.	trigarosa

## 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	7
2.4. Labdapattogás	8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS	8
2.6. Helló, Google!	9
2.7. 100 éves a Brun tétel	9
2.8. A Monty Hall probléma	9
<b>3. Helló, Chomsky!</b>	<b>10</b>
3.1. Decimálisból unárisba átváltó Turing gép	10
3.2. Az $a^n b^n c^n$ nyelv nem környezetfüggetlen	11
3.3. Hivatkozási nyelv	12
3.4. Saját lexikális elemző	13
3.5. l33t.1	14
3.6. A források olvasása	14
3.7. Logikus	16
3.8. Deklaráció	16

<b>4. Helló, Caesar!</b>	<b>18</b>
4.1. double ** háromszögmátrix	18
4.2. C EXOR titkosító	19
4.3. Java EXOR titkosító	20
4.4. C EXOR törő	21
4.5. Neurális OR, AND és EXOR kapu	22
4.6. Hiba-visszaterjesztéses perceptron	22
<b>5. Helló, Mandelbrot!</b>	<b>24</b>
5.1. A Mandelbrot halmaz	24
5.2. A Mandelbrot halmaz a <code>std::complex</code> osztállyal	24
5.3. Biomorfok	24
5.4. A Mandelbrot halmaz CUDA megvalósítása	25
5.5. Mandelbrot nagyító és utazó C++ nyelven	25
5.6. Mandelbrot nagyító és utazó Java nyelven	25
<b>6. Helló, Welch!</b>	<b>26</b>
6.1. Első osztályom	26
6.2. LZW	26
6.3. Fabejárás	26
6.4. Tag a gyökér	28
6.5. Mutató a gyökér	28
6.6. Mozgató szemantika	29
<b>7. Helló, Conway!</b>	<b>30</b>
7.1. Hangyaszimulációk	30
7.2. Java életjáték	34
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. Mély MNIST	39
8.3. Minecraft-MALMÖ	40

---

<b>9. Helló, Chaitin!</b>	<b>41</b>
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . . .	41
9.2. Gimp Scheme Script-fu: króm effekt . . . . .	41
9.3. Gimp Scheme Script-fu: név mandala . . . . .	44
<b>10. Helló, Gutenberg!</b>	<b>45</b>
10.1. Programozási alapfogalmak . . . . .	45
10.2. Programozás bevezetés . . . . .	46
10.3. Programozás . . . . .	47
<b>III. Második felvonás</b>	<b>48</b>
<b>11. Helló, Arroway!</b>	<b>50</b>
11.1. A BPP algoritmus Java megvalósítása . . . . .	50
11.2. Java osztályok a Pi-ben . . . . .	50
<b>IV. Irodalomjegyzék</b>	<b>51</b>
11.3. Általános . . . . .	52
11.4. C . . . . .	52
11.5. C++ . . . . .	52
11.6. Lisp . . . . .	52

---

# 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?

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



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

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

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



#### A DocBook XML 5.1 új neked?

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

---

# **I. rész**

## **Bevezetés**

# 1. fejezet

## Vízió

### 1.1. Mi a programozás?

### 1.2. Milyen doksikat olvassak el?

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

### 1.3. Milyen filmeket nézzek meg?

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

## **II. rész**

### **Tematikus feladatok**

**Bátf41 Haxor Stream**

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

---

## 2. fejezet

# Helló, Turing!

### 2.1. Végtelen ciklus

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

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

Megoldás forrása:

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

Megoldás videó:

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

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

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

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

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

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

Mit for kiírni erre a T1000 (T1000) futtatásra?

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

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

Tanulságok, tapasztalatok, magyarázat...

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

## 2.3. Változók értékének felcserélése

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

Megoldás videó: [https://bhaxor.blog.hu/2018/08/28/10\\_begin\\_goto\\_20\\_avagy\\_elindulunk](https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk)

Megoldás forrása:

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



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

Tanulságok, tapasztalatok, magyarázat...

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

## 2.4. Labdapattogás

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

## 2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: [https://youtu.be/9KnMqrkj\\_kU](https://youtu.be/9KnMqrkj_kU)

Megoldás forrása:

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

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

Tanulságok, tapasztalatok, magyarázat...

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

## 2.6. Helló, Google!

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

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat...

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

## 2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/blob/master/attention\\_raising/Primek\\_R](https://gitlab.com/nbatfai/bhax/blob/master/attention_raising/Primek_R)

## 2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás videó: [https://bhaxor.blog.hu/2019/01/03/erdos\\_pal\\_mit\\_keresett\\_a\\_nagykonyvben\\_a\\_monty\\_hall-paradoxon\\_kapcsan](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)

Tanulságok, tapasztalatok, magyarázat...

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

## 3. fejezet

# Helló, Chomsky!

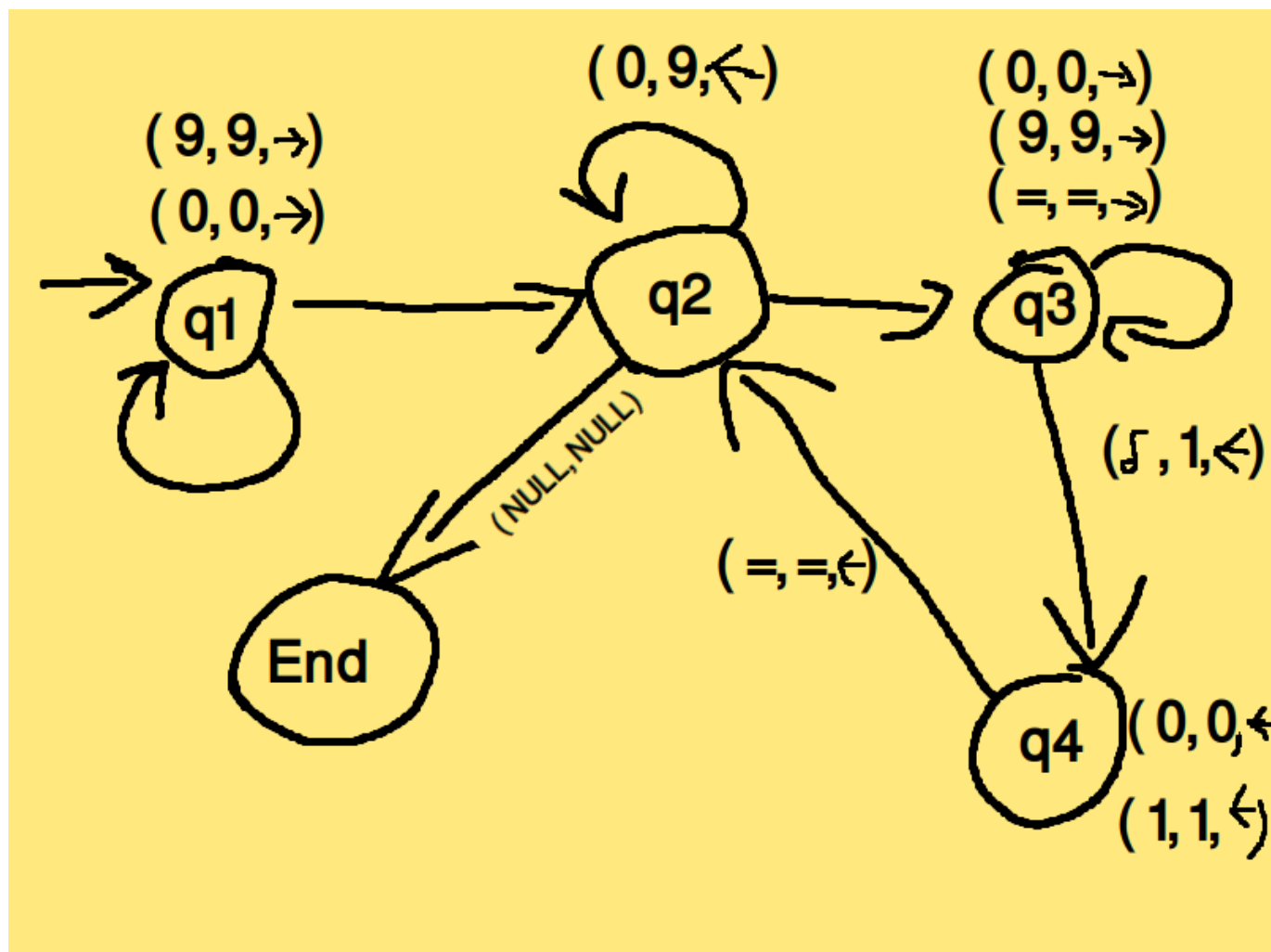
### 3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

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



Tanulságok, tapasztalatok, magyarázat...

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

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

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

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

Megoldás forrása:

S, X, Y legyenek változók

$a, b, c$  legyenek konstansok

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

### 3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

### 3.4. Saját lexikális elemző

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

Megoldás videó: [https://youtu.be/9KnMqrkj\\_kU](https://youtu.be/9KnMqrkj_kU)

Megoldás forrása:

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

### 3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

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

Tanulságok, tapasztalatok, magyarázat...

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

### 3.6. A források olvasása

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

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

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

**Bugok**

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

i.

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

ii.

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

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

iii.

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

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

iv.

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

**Bug**

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

v.

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

**Bug**

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

vi.

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

**Bug**

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



vii.

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

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

viii.

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

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

### 3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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)

Tanulságok, tapasztalatok, magyarázat...

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

### 3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész

```
int a=5;
```

- egészre mutató mutató

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

- egész referenciája

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

- egészek tömbje

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

- egészek tömbjének referenciája (nem az első elemé)

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

- egészre mutató mutatók tömbje

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

- egészre mutató mutatót visszaadó függvény

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

- egészre mutató mutatót visszaadó függvényre mutató mutató

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

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

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

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

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

## 4. fejezet

# Helló, Caesar!

### 4.1. double \*\* háromszögmátrix

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

Megoldás forrása:

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

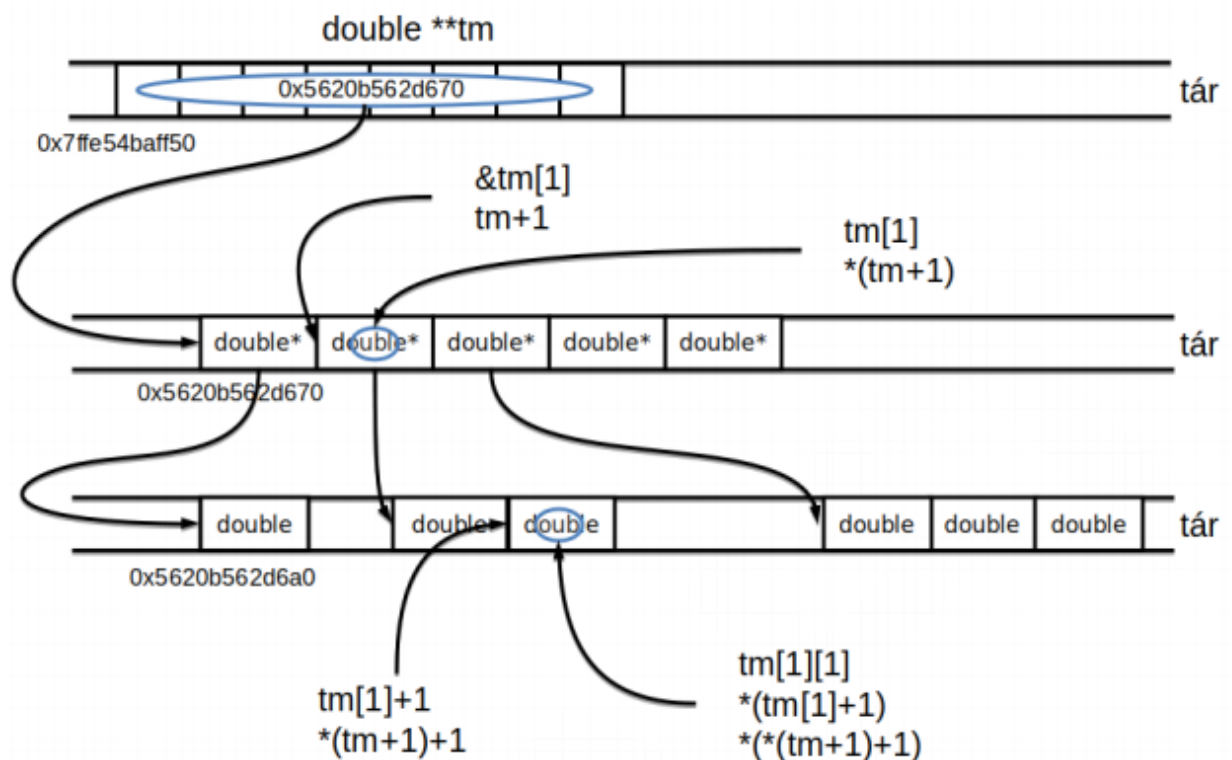
```

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

```

Tanulságok, tapasztalatok, magyarázat...

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



## 4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

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

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

Tanulságok, tapasztalatok, magyarázat...

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

## 4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

```
import java.io.InputStream;
import java.io.OutputStream;

public class ExorTitkosító {

    public static void Exor(String kulcsSzöveg, java.io.InputStream bejöv ←
        őCsatorna, 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
        {
            ExorTitkosító(args[0], System.in, System.out);
        }
        catch(java.io.IOException e) { e.printStackTrace(); }
    }
}
```

Tanulságok, tapasztalatok, magyarázat...

Lényeges különbséget nem tapasztalhatunk a C verziós és a Java verziós EXOR titkosító között. Itt az algoritmus az ExorTitkosító osztályban lett megvalósítva, de ugyanazt az elvet használja, mint az előző feladatban leírt verzió. A külső while ciklus buffer tömbönként addig olvassa a bemenetet, amíg csak tudja. A belső for ciklusban helyezzük rá a kulcsot a beolvasott bájtokra a kulcsIndex változó segítségével, majd végrehajtjuk a kizáró vagy műveletet, az eredmény a buffer tömbben keletkezik, amit végül a kimenetre írunk.

## 4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:[https://github.com/gergokinczel/unideb/blob/master/c\\_exor\\_toro.c](https://github.com/gergokinczel/unideb/blob/master/c_exor_toro.c)

[https://progpater.blog.hu/2011/02/15/felvetelt\\_hirdet\\_a\\_cia?fbclid=IwAR20XPCHTlyHUvM0GGA5adxYhVhw\\_dUNn7\\_8rb3\\_mQb4Seq4Kb4w](https://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia?fbclid=IwAR20XPCHTlyHUvM0GGA5adxYhVhw_dUNn7_8rb3_mQb4Seq4Kb4w)

Tanulságok, tapasztalatok, magyarázat...

A C EXOR törő lényege, hogy megtalálja a kulcsot mely egy EXOR-al titkosított magyar szöveget feltör és amely kulcsnak a mérete akár 8 karakter. Ez után pedig kiírja a kulcsot és a tiszta szöveget. A program úgy végzi a kulcs megtalálását, hogy minden lehetséges kulcs kombinációt kipróbál, majd a legmegfelelőbbet kiválasztja (legmegfelelőbb itt egy olyan szövegre céloz mely gyakori magyar szavakat tartalmaz, és ugyanakkor az átlagos szóhossznak is eleget tesz).

## 4.5. Neurális OR, AND és EXOR kapu

R

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

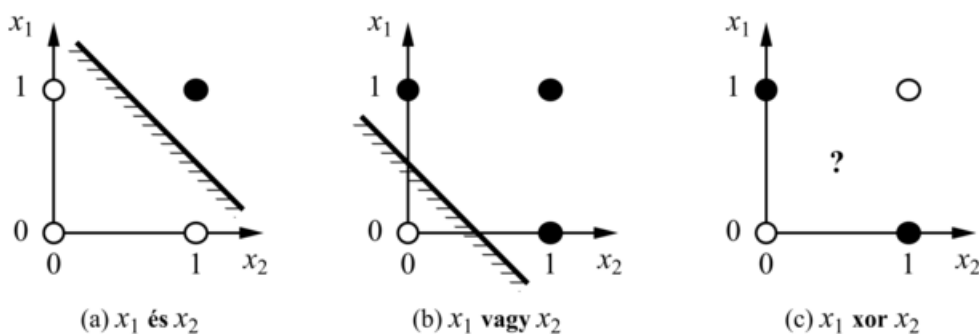
Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/NN\\_R](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R)

Tanulságok, tapasztalatok, magyarázat...

A neurális háló egy alapot képez a mesterséges intelligencia megértéséhez és elkészítéséhez. Ennek minimum három fő rétege van. A bementeti réteg, amely a konkrét összehasonlítandó elemeket adja át. A köztes rétegben történik meg a tényleges művelet elvégzése, ilyen például a *vagy* és az *és* művelet. A kimeneti rétegben pedig egy neuron/függvény melybe bekerül az információ.

A háló fő feladata, hogy az egyes neuronok egy értéket továbbítsanak, amennyiben a köztes rétegben szereplő érték elért egy úgynevezett küszöbértéket, majd pedig egy véges lépésszám elvégzése után megközelítsék a valós eredményt. Minden kapcsolat rendelkezik egy súlyal, amely az adott kapcsolat előjelét és erősségét határozza meg.

A forrásban szereplő példák közül a *vagy* és az *és* művelet bármi addíció nélkül is kivételesen működik, az XOR művelethez viszont rejtett rétegekre is szükségünk van, hogy a kívánt pontosságot el tudjuk érni. Ennek az az oka, hogy az XOR műveletet nem lehet lineárisan szeparálni, ellentétben a *vagy* és az *és* művelettel.



## 4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

Tanulságok, tapasztalatok, magyarázat...

Azt a hálót, amelyben az összes bemenet közvetlenül a kimenetekre kapcsolódik egyrétegű neurális hálónak (single layer neural network) vagy perceptron (perceptron) hálónak nevezzük. Mivel mindegyik kimeneti egység független a többtől – mindegyik súly csak egyetlen kimenetre van hatással – vizsgálatainkat korlátozhatjuk az egykimenetű perceptronra. A perceptronok két fajta csomópontból tevődnek össze: bemeneti és kimenetiből. A két csomópont között helyezkedik el az a súlyozott kapcsolat, mely meghatározza mennyire erős a ki és bemenet közötti viszony. A háló aképpen tanul, hogy akár több száz ilyen súlyozást elvégez, majd egy bizonyos kimeneti érték felé konvergál ami valószínűleg a megoldás.



## 5. fejezet

# Helló, Mandelbrot!

### 5.1. A Mandelbrot halmaz

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

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

Megoldás forrása: [https://github.com/gergokinczel/unideb/blob/Mandel/mandel\\_komplex.cpp](https://github.com/gergokinczel/unideb/blob/Mandel/mandel_komplex.cpp)

Tanulságok, tapasztalatok, magyarázat...

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

### 5.3. Biomorfok

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

Megoldás forrása: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/Biomorf](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf)

Tanulságok, tapasztalatok, magyarázat...

---

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

## 5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás videó: [https://bhaxor.blog.hu/2018/09/02/ismerkedes\\_a\\_mandelbrot\\_halmazal](https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazal)

Megoldás forrása: [https://github.com/gergokinczel/unideb/tree/Mandel/mandelbrot\\_rgb](https://github.com/gergokinczel/unideb/tree/Mandel/mandelbrot_rgb)

Tanulságok, tapasztalatok, magyarázat...

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

## 5.6. Mandelbrot nagyító és utazó Java nyelven

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

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apbs02.html#id570518>

Tanulságok, tapasztalatok, magyarázat...

Látványában nem különbözik sokat a nagyító a C++-os verzióhoz képest, mégis kivitelezésében egy sokkal átláthatóbb kódot kapunk. Forráskódunk két osztályra van felosztva, az egyikben találjuk meg a halmaz kiszámolását egy megadott pontban, ezt felhasználván a másik osztályban szerepel a halmaz tényleges lerajzolása, egy pillanatkép készítése, e mellett pedig a jobb egérgombbal való klikkeléskor elküldi a másik osztálynak az egér x,y koordinátája szerint rámutatott csomópontot, amivel újraszámolja a halmazt így "nagyítva" a képet. Ezt a folyamatot annyiszor tudjuk megismételni ahányszor csak szeretnénk.

## 6. fejezet

# Helló, Welch!

### 6.1. Első osztályom

Valósítsd meg C++-ban és Java-ban a 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 eltérve kiszámolt szám.

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/polargen/> <https://www.tankonyvtar.hu/tartalom/tkt/javat-tanitok-javat/ch01.html>

Tanulságok, tapasztalatok, magyarázat...

A program megvalósítja a megvalósíthatatlant. Random számokból még randomabb számokat készít, mindent szorzás, összeadás kivonás négyzetre emelés, gyökvonás és logaritmusok segítségével. A C++ és Java verzió nem sokban tér el egymástól, Javában az osztályok elkészítése valamivel kézenfekvőbb, sokra viszont nem használjuk, csupán egy logikai változót reprezentál, amely jelöli, hogy van e éppen tárolt szám.

### 6.2. LZW

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

Megoldás forrása: [https://github.com/gergokinczel/unideb/blob/master/binfa\\_c.c](https://github.com/gergokinczel/unideb/blob/master/binfa_c.c)

Tanulságok, tapasztalatok, magyarázat...

A binfa egy bináris gráf, melyben a 0-ás elemek balra az 1-es elemek jobbra kerülnek egy gyökértől kiindulva. Bemeneti adatként egy 0 és 1-esekből álló számláncot kap az algoritmus, melyből sorban haladva a fába építi az elemeket, és minden olyan alkalommal, amikor beépít egy elemet a fába visszaugrik a gyökérre, egészen addig míg el nem fogynak a számok.

### 6.3. Fabejárás

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

Megoldás forrása:

```
//posztorder
void kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        kiir (elem->jobb_egy);
        kiir (elem->bal_nulla);
        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
            , melyseg-1);

        --melyseg;
    }
}
```

```
//preorder
void kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > max_melyseg)
            max_melyseg = melyseg;

        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ←
            , melyseg-1);
        kiir (elem->jobb_egy);
        kiir (elem->bal_nulla);

        --melyseg;
    }
}
```

Tanulságok, tapasztalatok, magyarázat...

A fabejárás változtatása egy nagyon egyszerű művelet, csupán a kiírató algoritmusban kell a kiíratás helyét felcserélni: a preorder(gyökerkezdő) bejáráshoz a kiíratást a függvény újrahívása elé kell tenni, a posztorder(gyökérvégző) bejáráshoz pedig a kiíratást a függvény újrahívása után kell elhelyezni.

## 6.4. Tag a gyökér

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

Megoldás forrása: <https://github.com/gergokinczel/unideb/blob/master/binfa2.cpp>

Tanulságok, tapasztalatok, magyarázat...

A fa lényegében ugyanaz, mint C-ben viszont át van ültetve egy osztályba. Ennek az osztálynak van egy további Csomópont és Fa osztálya melyeket a fa építésénél és bejárásánál is egyaránt használnunk kell, és amelyek átláthatóbbá teszik a forráskódunkat. Így tehát rendelkezünk egy konstruktorral, mely megadja a fánk kezdetleges helyzetét, ugyanakkor meghadja a lehetőséget, hogy a jelenlegi csomópont változóval építhessük a fánkat. Ugyanakkor dekonstruktorunk is van, melyre szükség van mivel helyet foglalunk a memóriában a csomópontoknak.

## 6.5. Mutató a gyökér

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

Megoldás forrása: [https://github.com/gergokinczel/unideb/blob/master/binfa\\_mutat.cpp](https://github.com/gergokinczel/unideb/blob/master/binfa_mutat.cpp)

Tanulságok, tapasztalatok, magyarázat...

Ebben a megoldásban az előzőhöz képest megváltozik a gyökér, mely a jobb és baloldali elemekhez hasonlóan már aggregációban lesz a fával, azaz mutatóként hozzuk létre. Lefoglaljuk a memóriacímét nem csak a jobb és bal elemeknek, hanem a gyökérnek is, ezzel konzisztensé téve a fa részeit. Kódrészletben alig történik változás:

```
//előtte
protected
    Csomopont gyoker
-----
//utána
protected
    Csomopont *gyoker
```

```
//előtte
public:

    LZWBinFa ():fa (&gyoker)
    {
    }
-----
//utána
public:

    LZWBinFa ()
    {
    gyoker = new Csomopont();
```

```
fa = gyoker;
}
```

```
        //az összes előfordulásnál
        //előtte
&gyoker
-----
        //utána
gyoker
```

```
        //előtte
~LZWBinFa ()
{
    szabadit (gyoker.egyenesGyermek ());
    szabadit (gyoker.nullasGyermek ());
}
-----
        //utána
~LZWBinFa ()
{
    szabadit (gyoker->egyenesGyermek ());
    szabadit (gyoker->>nullasGyermek ());
}
```

## 6.6. Mozgató szemantika

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

Megoldás forrása: <https://gist.github.com/gergokinczel/7ded0f06588bbee48117c53a596aa2e9>

A mozgató szemantika lehetőséget nyújt arra, hogy egy t objektumot egy másik objektum felé "mozgassunk", a forrásait átvigyük. A binfánkban ezt arra használhatjuk és használjuk, hogy egy másolatot készítsünk a már felépített binfáról, majd azt újra kiírjuk. Erre a mozgatásra az `std::move` parancsot használjuk fel, mely nemcsak binfa, hanem bármilyen más objektum "mozgatását" is megengedi. Tulajdonképpen egy új memóriacímet kapcsol az objektumhoz ezzel egy sokkalta gyorsabb, és effektívebb másolást eredményezve, minthogy a fa minden elemét átmásoljuk.

```
LZWBinFa binFa_uj = std::move ( binFa );
```

## 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: [https://gitlab.com/nbatfai/bhax/tree/master/attention\\_raising/Myrmecologist](https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Myrmecologist)

Tanulságok, tapasztalatok, magyarázat...

A hangyaszimuláció egy nagyon alapvető sejtautomata, melynek az a célja, hogy egy hangyabolyt imitáljon.

Az elkészített hangyaszimulációs programunk több részre van osztályokkal felbontva, melyeknek több különböző szerepe van. Először is a legalapabb része a hangya kolóniánknak egy individuális hangya, mely az `Ant` osztályba lett létrehozva és amely mindössze csak három adattal rendelkezik: a hangya x,y koordinátája és az iránya.

```
#ifndef ANT_H
#define ANT_H

class Ant
{
public:
    int x;
    int y;
    int dir;
    Ant(int x, int y): x(x), y(y) {

        dir = qrand() % 8;

    }
};

typedef std::vector<Ant> Ants;
```

Ezt az osztályt használja fel a programunk a továbbiakban, hogy a hangyákat elkészítse. Rendelkezőnk még egy `AntWin` osztállyal, amely tartalmazza konstruktort, dekonstruktort és a rajzolási eventet.

```
#ifndef ANTWIN_H
#define ANTWIN_H

#include <QMainWindow>
#include <QPainter>
#include <QString>
#include <QCloseEvent>
#include "antthread.h"
#include "ant.h"

class AntWin : public QMainWindow
{
    Q_OBJECT

public:
    AntWin(int width = 100, int height = 75,
           int delay = 120, int numAnts = 100,
           int pheromone = 10, int nbhPheromon = 3,
           int evaporation = 2, int cellDef = 1,
           int min = 2, int max = 50,
           int cellAntMax = 4, QWidget *parent = 0);

    AntThread* antThread;

    void closeEvent ( QCloseEvent *event ) {

        antThread->finish();
        antThread->wait();
        event->accept();
    }

    void keyPressEvent ( QKeyEvent *event )
    {

        if ( event->key() == Qt::Key_P ) {
            antThread->pause();
        } else if ( event->key() == Qt::Key_Q
                    || event->key() == Qt::Key_Escape ) {
            close();
        }

    }

    virtual ~AntWin();
    void paintEvent(QPaintEvent*);

private:
```



```

    int ***grids;
    int **grid;
    int gridIdx;
    int cellWidth;
    int cellHeight;
    int width;
    int height;
    int max;
    int min;
    Ants* ants;

public slots :
    void step ( const int &);
};

```

Az utolsó osztályunk az AntThread, melyet a hangyák mozgatására használunk, ebbe bele tartozik az, hogy új irányt adunk nekik, konkrétan elmozgatjuk őket és a feromon kibocsátásukat is meghatározzuk.

```

#ifndef ANTTHREAD_H
#define ANTTHREAD_H
#include <QThread>
#include "ant.h"

class AntThread : public QThread
{
    Q_OBJECT

public:
    AntThread(Ants * ants, int ***grids, int width, int height,
              int delay, int numAnts, int pheromone, int nbrPheromone,
              int evaporation, int min, int max, int cellAntMax);

    ~AntThread();

    void run();
    void finish()
    {
        running = false;
    }

    void pause()
    {
        paused = !paused;
    }

    bool isRunnung()
    {
        return running;
    }
}

```

```

private:
    bool running {true};
    bool paused {false};
    Ants* ants;
    int** numAntsinCells;
    int min, max;
    int cellAntMax;
    int pheromone;
    int evaporation;
    int nbrPheromone;
    int ***grids;
    int width;
    int height;
    int gridIdx;
    int delay;

    void timeDevel();

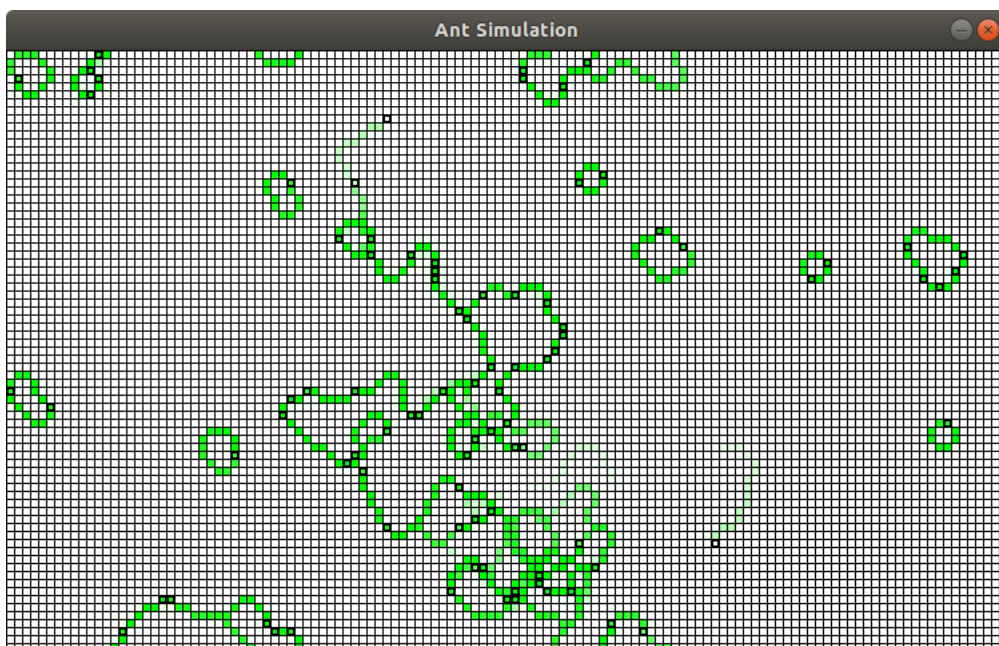
    int newDir(int sor, int oszlop, int vsor, int voszlop);
    void detDirs(int irany, int& ifrom, int& ito, int& jfrom, int& jto );
    int moveAnts(int **grid, int row, int col, int& retrow, int& retcol, ←
        int);
    double sumNbhs(int **grid, int row, int col, int);
    void setPheromone(int **grid, int row, int col);

signals:
    void step ( const int &);

};

```

Ezek az osztályok mind előbb definiálva vannak egy külön fájlba (ezek láthatóak fentebb), majd őket meghíván a program használatnak fel.



## 7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása: <https://www.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apb.html>

<https://github.com/gergokinczel/unideb/blob/prog1/Sejtautomata.java>

Tanulságok, tapasztalatok, magyarázat...

Az életjáték bővebb leírása a következő feladat résznél található, eredetétől kezdve egészen a működési elvekig. Ebben a feladatban inkább a forráskódra fókuszálnék, mely nem túl sokban, de eltér a C++-os változattól, itt ugyanis egyetlen egy osztályban hozzuk létre a sejtautomatánkat. Az első függvényünkben először létrehozuk a rácsot melyben a sejtek élni fognak:

```
public Sejtautomata(int szélesség, int magasság) {
    this.szélesség = szélesség;
    this.magasság = magasság;

    rácsok[0] = new boolean[magasság][szélesség];
    rácsok[1] = new boolean[magasság][szélesség];
    rácsIndex = 0;
    rács = rácsok[rácsIndex];

    for(int i=0; i<rács.length; ++i)
        for(int j=0; j<rács[0].length; ++j)
            rács[i][j] = HALOTT;
```

Ez után több billentyű- és egérfunkciónak a szerepeltetését adtuk meg:

```
addKeyListener(new java.awt.event.KeyAdapter() {

    public void keyPressed(java.awt.event.KeyEvent e) {
        if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {

            cellaSzélesség /= 2;
            cellaMagasság /= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {

            cellaSzélesség *= 2;
            cellaMagasság *= 2;
            setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                    Sejtautomata.this.magasság*cellaMagasság);
            validate();
        } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
            pillanatfelvétel = !pillanatfelvétel;
        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
            várakozás /= 2;
```

```

        else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
            várakozás *= 2;
        repaint();
    }
});
addMouseListener(new java.awt.event.MouseAdapter() {

    public void mousePressed(java.awt.event.MouseEvent m) {

        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
        repaint();
    }
});

addMouseMotionListener(new java.awt.event.MouseMotionAdapter()
{
    public void mouseDragged(java.awt.event.MouseEvent m)
    {
        int x = m.getX()/cellaSzélesség;
        int y = m.getY()/cellaMagasság;
        rácsok[rácsIndex][y][x] = ÉLŐ;
        repaint();
    }
});

```

Következő elengedhetetlen része a kódnak az ablak adatainak a megadása és a sejtter futtatása:

```

setTitle("Sejtautomata");
setResizable(false);
setSize(szélesség*cellaSzélesség, magasság*cellaMagasság);
setVisible(true);
// A sejtter futtatása
new Thread(this).start();

```

Ezek után meghívjuk még a rajzolásra alkalmazott függvényt, majd a szomszéd kritériumot ellenőrző függvényt melyet az idő eltoló függvény hív meg, ami pedig a run függvényben szerepel. Utolsó függvényeink közé tartozik a sikló, és siklókilövő, amelyek a siklók mozgatásáért felelősek, utánuk egy pillanatfelvételt készítő függvény, a rajzoláshoz tartozó update függvény, végezetül pedig a főfüggvényünk szerepel, mely a Sejtautomatát hívja meg.

```

public static void main(String[] args)
{
    new Sejtautomata(100, 75);
}

```

## 7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása: <https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/Qt/Sejtauto/>

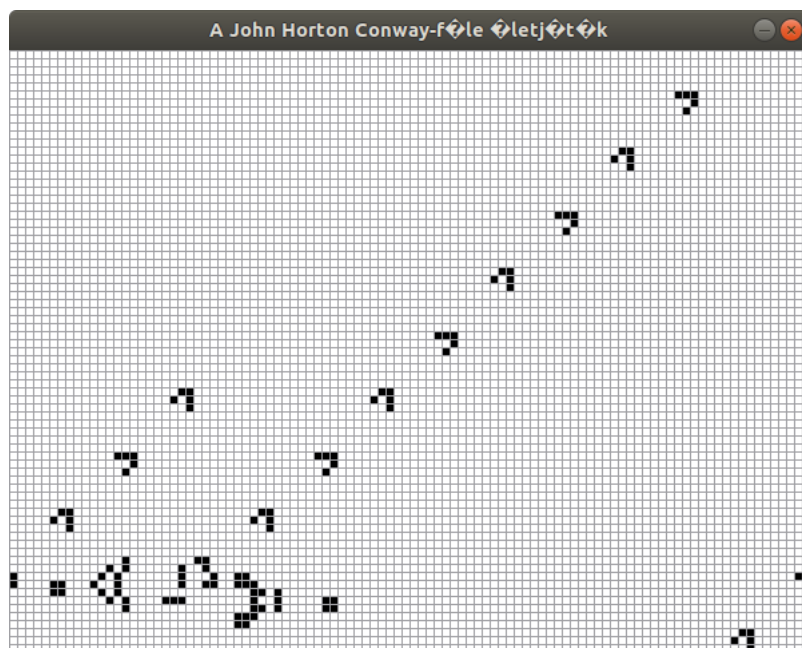
Tanulságok, tapasztalatok, magyarázat...

Az Életjáték (Game of Life) egy sejtautomata, melyet John Horton Conway Britt matematikus fejlesztett ki 1970-ben. Sejtautomatának nevezünk egy olyan diszkrét modellt, melyet mikrostrukturák modellezésében használnak fel: egy négyzetrácsban helyezkedik el, a négyzetrácsok által közrefogott cellákat sejteknek nevezzük. A sejteknek több különböző állapota lehet, és ahogy telik az idő a cellák változtatják állapotukat valamilyen feltétel szerint.

A John Horton féle életjátékban nincs megadott játékos, az evolúciót a kezdeti pozíció határozza meg így nem szükséges semmi beavatkozás. Ezen felül a játékot négy különböző szabály határozza meg:

- Bármely olyan élő cella, melynek kevesebb, mint két élő szomszédja van meghal,
- Bármely olyan élő cella, melynek kettő vagy három élő szomszédja van, élhet tovább
- Bármely olyan élő cella, melynek több mint három élő szomszédja van meghal, túlnépesedés miatt
- Minden halott cella, melynek pontosan három élő szomszédja van, élő cellává válik

Programunk, kivitelezéséhez két osztállyal rendelkezik, mindkettő előbb külön-külön definiálva egy különálló fájlba. Az első meghívott osztály a `SejtAblak`, ebbe szerepel a konstruktor, dekonstruktor és a rajzoló esemény (paint event). A második osztályunk a `SejtSzal`, itt pedig azok a programrészek futnak le, melyek melyek ellenőrzik a szabályok betartását és befolyásolják, hogy mely cellába van élet és melybe szűnik meg.



## 7.4. BrainB Benchmark

Megoldás forrása: <https://github.com/nbatfai/esport-talent-search>

Tanulságok, tapasztalatok, magyarázat...

A videójátékok világában gyakran találkozhatunk azzal az eseménnyel, amikor a játékos elveszíti az irányítást a karaktere fölött, pusztán a miatt, mert azt elveszítette a különböző játékelemek sokasága miatt. Ez a szintfelmérő program pont ezt a jelenséget próbálja meg előidézni és a játékost próbára tenni, megerősítvén ennek az eseménynek a létezését.

Nekünk, mint játékos nincs más dolgunk, mint követni az egyik kockában elhelyezett karikát a kurzorral tíz percen keresztül, miközben egyre több környezeti elem jelenik meg a képernyőn az általunk kiválasztott "karakter" mellett. Mindezt azért, hogy ezek után egy teljes képet tudjunk nyerni a játékos képességeiről és a jelenség tényleges létezéséről.

Forráskód szempontjából nagyban hasonlít az előző feladatokra: két fő osztállyal rendelkezik. Az egyik osztály elvégzi a pálya rajzolását és ellenőrzi, hogy mikor kezdődik a játék, mikor van lenyomva az egérgomb. A másik osztály végzi el a számításokat, azaz új akadályok beépítése, mozgatása, régiek eltüntetése abban az esetben ha már nincs lenyomva az egérgomb.



## 8. fejezet

# Helló, Schwarzenegger!

### 8.1. Szoftmax Py MNIST

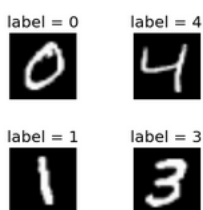
Python

Megoldás videó: <https://youtu.be/j7f9SkJR3oc>

Megoldás forrása: <https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0> [https://progpater.blog.hu/-/2016/11/13/hello\\_samu\\_a\\_tensorflow-bol](https://progpater.blog.hu/-/2016/11/13/hello_samu_a_tensorflow-bol)

Tanulságok, tapasztalatok, magyarázat...

A MNIST(Modified National Institute of Standards and Technology database) egy kezdő szintű, gépi képfeldolgozó adatbázis, mely több különböző kézzel írott digitális képet tartalmaz, mint például ez:



Ugyanakkor tartalmazza az összes képhez kapcsolódó címkét is, ahogyan azt a fenti kép is reprezentálja.

A Szoftmax MNIST segítségével feltaníthatjuk programunkat, hogy akár 90%-os pontossággal meghatározza éppen milyen számjegyet "lát" a képernyőn, ehhez pedig elengedhetetlenül szükséges egy nagyobb adatbázis amiből tanítjuk, és melynek segítségével tesztelhetjük is programunkat.

Python kódot használunk, melybe implementáljuk a Tensorflow modult, ami többek között lehetővé teszi, hogy a CUDA kompatibilis GPU-nkat is felhasználjuk a számolásokhoz, így gyorsítva és pontosítva a gépi tanulást, ugyanakkor pythonon kívüli komplex számításoknak ad helyet. Ezt a modult a következőképpen implementáljuk a kódunkba:

```
import tensorflow as tf
```

Ezen kívül szükséges az adatbázist is implementálnunk a kódba:

```
from tensorflow.examples.tutorials.mnist import input_data  
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
```

Ez az adatbázis két külön részre van osztva: `mnist.train` és `mnist.test`. Ezzel megkülönböztetjük azokat a képeket melyeket tesztelésre és azokat melyeket tanításra használunk. Ahogyan az korábban is említve volt, minden tagja az adatbázisnak két elemet tartalmaz, egy képet és a hozzá tartozó címkét, melyet így találunk a kódban: `mnist.test.images` `mnist.test.labels`.

Mindezek után elkezdhetjük elkészíteni a modellünket:

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Az `x` itt nem egy konkrét változó, hanem egy helykitöltő, mellyel meghatározzuk, hogy bármennyi képet felhasználhatunk, és ezek mindegyikét egy 784 tagú vektorba sűrítjük bele. A `W` és `b` változó jelölik a súlyt és az alakot melyeket az `y` értékének kiszámításához fogunk használni, amivel pedig megkapjuk már maga a modellt.

A következő részben tanítjuk a modellt:

```
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y),
    reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

tf.initialize_all_variables().run()
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    train_step.run({x: batch_xs, y_: batch_ys})
```

Majd pedig az elkészült modellt leteszteljük:

```
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(accuracy.eval({x: mnist.test.images, y_: mnist.test.labels}))
```

Ennek a modellnek egy körülbelül 91%-os sikerességi aránya kellene, hogy legyen melyet persze lehet még javítani, de mint kezdők megelégszünk ezzel az eredménnyel.

## 8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása: [https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/examples/tutorials/mnist/-mnist\\_deep.py](https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/examples/tutorials/mnist/-mnist_deep.py)

Tanulságok, tapasztalatok, magyarázat...



## 8.3. Minecraft-MALMÖ

Megoldás videó: <https://youtu.be/bAPSu3Rndi8>

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

## 9. fejezet

# Helló, Chaitin!

### 9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó: <https://youtu.be/z6NJE2a1zIA>

Megoldás forrása:

```
(princ "Adja meg a kívánt faktoriális: " )  
(setq f (read))  
(defun faktor (s)  
  (if(< s 1)  
      1(* s (faktor (1- s))))  
(princ "Az eredmény: ")  
(write (faktor f))
```

Tanulságok, tapasztalatok, magyarázat...

A LISP egy programozási nyelvcsalád, melyet már ősidők óta használnak az emberek, és amely talán a második legidősebb magas szintű programozási nyelv. Fő ismertető jele talán a teljesen zárójelezett prefix alakja, viszont ettől még eltéréseket találhatunk LISP kódok között a több különböző dialektus miatt, mint például a Clojure, Common Lisp vagy a Scheme.

A fenti rekurzív faktoriális számoló program jól érzékelteti a nyelv sajátosságait: rengeteg zárójel, prefixezett alak, szimbolikus kifejezések. Egy kezdőértékből indul ki amit a billentyűzetről adunk meg, majd ezt iteratíván csökkentve önmagával megszorozza amíg az eléri az 1-et, és az így megkapott eredményt kiíratjuk a képernyőre.

### 9.2. Gimp Scheme Script-fu: króm effekt

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

Megoldás 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...

Az előző feladatra építve egy kicsit komolyabb Lisp kóddal foglalkozik a Gimp Scheme. Feladata egy króm effektet tenni egy bemenő szövegre, és mindezt a GIMP képszerkeztő programba fogjuk implementálni. Először készítünk egy függvényt, amely tömböket hoz létre:

```
(define (color-curve)
  (let* (
    (tomb (cons-array 8 'byte))
  )
    (aset tomb 0 0)
    (aset tomb 1 0)
    (aset tomb 2 50)
    (aset tomb 3 190)
    (aset tomb 4 110)
    (aset tomb 5 20)
    (aset tomb 6 200)
    (aset tomb 7 190)
  tomb)
)
```

A következő függvényünk egy lista általunk megadott elemét fogja visszaadni:

```
(define (elem x lista)
  (if (= x 1) (car lista) (elem (- x 1) (cdr lista) ) )
)
```

Az ezt követő függvény a szövegünk szélességét és magasságát határozza meg:

```
(define (text-wh text font fontsize)
  (let*
    (
      (text-width 1)
      (text-height 1)
    )

    (set! text-width (car (gimp-text-get-extents-fontname text fontsize ↵
      PIXELS font)))
    (set! text-height (elem 2 (gimp-text-get-extents-fontname text ↵
      fontsize PIXELS font)))

    (list text-width text-height)
  )
)
```

Utolsó előtti kódrészletünk maga a króm effekt létrehozásáért lesz felelős, melyhez felhasználja az előzőleg definiált függvényeket, és összesen kilenc különböző lépésben elvégzi a krómosítást:

```
(define (script-fu-bhax-chrome text font fontsize width height color ↵
  gradient)
  (let*
    (
```

```

    (image (car (gimp-image-new width height 0)))
    (layer (car (gimp-layer-new image width height RGB-IMAGE "bg" 100 ↔
        LAYER-MODE-NORMAL-LEGACY)))
    (textfs)
    (text-width (car (text-wh text font fontsize)))
    (text-height (elem 2 (text-wh text font fontsize)))
    (layer2)
)

;step 1
(gimp-image-insert-layer image layer 0 0)
(gimp-context-set-foreground '(0 0 0))
(gimp-drawable-fill layer FILL-FOREGROUND )
(gimp-context-set-foreground '(255 255 255))

(set! textfs (car (gimp-text-layer-new image text font fontsize PIXELS) ↔
))
(gimp-image-insert-layer image textfs 0 0)
(gimp-layer-set-offsets textfs (- (/ width 2) (/ text-width 2)) (- (/ ↔
    height 2) (/ text-height 2)))

(set! layer (car (gimp-image-merge-down image textfs ↔
    CLIP-TO-BOTTOM-LAYER)))

;step 2
(plug-in-gauss-iir RUN-INTERACTIVE image layer 15 TRUE TRUE)

;step 3
(gimp-drawable-levels layer HISTOGRAM-VALUE .11 .42 TRUE 1 0 1 TRUE)

;step 4
(plug-in-gauss-iir RUN-INTERACTIVE image layer 2 TRUE TRUE)

;step 5
(gimp-image-select-color image CHANNEL-OP-REPLACE layer '(0 0 0))
(gimp-selection-invert image)

;step 6
(set! layer2 (car (gimp-layer-new image width height RGB-IMAGE "2" 100 ↔
    LAYER-MODE-NORMAL-LEGACY)))
(gimp-image-insert-layer image layer2 0 0)

;step 7
(gimp-context-set-gradient gradient)
(gimp-edit-blend layer2 BLEND-CUSTOM LAYER-MODE-NORMAL-LEGACY ↔
    GRADIENT-LINEAR 100 0 REPEAT-NONE
    FALSE TRUE 5 .1 TRUE width (/ height 3) width (- height (/ height ↔
        3)))

;step 8

```

```

(plug-in-bump-map RUN-NONINTERACTIVE image layer2 layer 120 25 7 5 5 0 ↔
 0 TRUE FALSE 2)

;step 9
(gimp-curves-spline layer2 HISTOGRAM-VALUE 8 (color-curve))

(gimp-display-new image)
(gimp-image-clean-all image)
)
)

```

Legvégül pedig az a kódcsipet kerül beiktatásba, mely a Gimp-en belüli menüben megjeleníti az effektusunkat:

```

(script-fu-register "script-fu-bhax-chrome"
  "Chrome3"
  "Creates a chrome effect on a given text."
  "Norbert Bátfai"
  "Copyright 2019, Norbert Bátfai"
  "January 19, 2019"
  ""
  SF-STRING      "Text"      "Bátf41 Haxor"
  SF-FONT        "Font"      "Sans"
  SF-ADJUSTMENT  "Font size" '(100 1 1000 1 10 0 1)
  SF-VALUE      "Width"      "1000"
  SF-VALUE      "Height"     "1000"
  SF-COLOR       "Color"      '(255 0 0)
  SF-GRADIENT    "Gradient"   "Crown molding"
)
(script-fu-menu-register "script-fu-bhax-chrome"
  "<Image>/File/Create/BHAX"
)

```

### 9.3. 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...

## 10. fejezet

# Helló, Gutenberg!

### 10.1. Programozási alapfogalmak

[PICI]

Juhász István ezen könyve a fiatal informatikusoknak nyújt bevezetést ennek a csodálatos tudományágnak a világába.

Az első fejezetekben megismerkedhetünk a különféle alapfogalmakkal, melyeket kötelező tudnia mindenkinek, aki csak ebben a szakmában elszeretne helyezkedni. Olyan dolgokról szerezhethünk tudást, mint például a különböző programozási nyelvek, típusok, ezeknek fontossága és alapvetőleges működése.

Ez után elkezdjük belemélyíteni magunkat a magas szintű programozási nyelvek világába, melynek egyik első fontos kulcsszava az adattípus(ok), melyek lehetnek egyszerű, összetett vagy akár mutató típusok. Ezen részt követve említődnek meg a konstansok és változók, amik ugyancsak elengedhetetlen alapkövei egy programnak.

Harmadik fő fejezetünk a kifejezésekről szól, melyet megmagyaráz és példákkal bemutat C programozási nyelvben.

A soron következő cím az Utasítások, mely talán a legnagyobb fejezetet foglalja magába az egész könyvben, nem is hiába, ugyanis ezek az eljárásorientált nyelveknek olyan egységei, melyeken alapulhat maga a program. Két fő csoportba osztható: a deklarációs és végrehajtható utasítások. Ez utóbbira kerül inkább a nagyobb hangsúly, ugyanis kilenc másik csoportot foglal magába: értékadó, üres, ugró, elágazó, ciklusszervező, hívó, vezérlésátadó, I/O és egyéb utasítások.

Ezek után ugrunk egy kicsit a könyvben, hogy megismerkedjünk a programok szerkezete című fejezettel. Itt ahogyan azt a cím is sugallja, nem másról, mint a programoknak a felépítéséről lesz szó, melyek a nagyobb alkotóelemei, és melyek azok az eljárások, amikkel ezeket módosítjuk, kihasználjuk. A legfőbb alkotóelem, melyet megtanulunk az az alprogram, aminek a segítségével több különböző részre tudunk egy programot felosztani, lehetővé és egyszerűbbé téve a csoportban való dolgozást, ugyanakkor pedig sokkal átláthatóbbá is teszi a programunkat. De nem ezek az egyedüli előnyei, több más gyakorlati haszna is van mely felmerülhet, ilyen például a rekurzió. Más is beletartozik egy program szerkezetébe persze, nem csak az alprogram, ide tartozik még a: Másodlagos belépési pontok, paraméterkiértékelés, paraméterátadás, a blokk és a hatáskör.

Kicsit tapasztalt programozóként már azért megtanulhattuk azt is, hogy nem mindig minden tökéletes elsőre, sőt sokadik alkalomra se, ugyanakkor nem mindig tudni mindennek a pontos végkimenetelét. Ezért is

olyan fontos a kivételkezelés, mellyel ugyancsak foglalkozik egy későbbi fejezet, és amely kapcsán kérdéseket is feltesz, amiket egy programnyelvnek mindenképpen meg kell válaszolnia ezen a téren.

A könyv végére megismerünk még olyan hasznos dolgokat is, mint az Input/Output eszközök és azoknak különböző változatait számos programnyelven, így munkánk nem csak a program környezetén belül, hanem azon kívül is változtathat dolgokat.

## 10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

A Kernighan és Ritchie által készített C programozási nyelvről szóló könyvben átnézzük a nyelvhez tartozó utasításokat, azok szerkezetét. A legfontosabb vezérlési szerkezetek kerülnek bemutatásra a soron következő felsorolásban:

- Utasítások és blokkok: Átlagosan, egy kifejezés utasítássá válik, ha egy pontosvessző kerül utána. Blokkokat { } kapcsos zárójelek segítségével jelöljük.
- If-else: Döntés kifejezésére használható, az else rész tetszés szerint elhagyható. Amennyiben az if(kifejezés)-ben teljesül a kifejezés (értéke nem nulla), az if után szereplő utasítás(-ok) elvégződnek. Ellenkező esetben, vagy nem végződik el az utasítás, vagy amennyiben van else rész, az kerül elvégzésre.
- Else if: Ennek segítségével az előző utasítást tudjuk bővíteni, annyi döntést adhatunk hozzá a folyamat-hoz, ahányat szeretnénk.
- Switch: Lehetővé teszi a többirányú programelágazást, azaz összehasonlítja a kapott kifejezés értékét több állandó kifejezés értékével, majd kiválasztván az eleget tevőt, annak az utasításait elvégzi. Ezeket az eseteket `case` jelöli, és amennyiben egyik eset sem teljesül, a `default` értékre áll át.
- While és for: Ismétlő utasítások, melyek mindenen előtt elvégeznek egy kiértékelést, ha az teljesül(nem nulla a kifejezés), elvégzi az utasítást, majd újra elvégzi a kiértékelést, mindaddig, amíg az már nem teljesül. Egyedüli különbség a két utasítás között, hogy a for ciklus három komponenssel rendelkezik, melyek közül általában az 1. és a 3. értékadás vagy függvényhívás, a 2. pedig egy relációs kifejezés.
- Do-while: Egyedüli különbség az előző ismétlő utasításokhoz képest az, hogy mivel a leállítási feltétele a ciklus végén van, az utasítást legalább egyszer mindenképpen elvégzi.
- Break: Ennek segítségével ki tudunk lépni a for,while, do ciklusból és a switch-ből, anélkül, hogy teljesülne a feltétel. Megjegyzés: csak a legbelső ciklusból lép ki.
- Continue: Elég ritkán alkalmazott utasítás, szerepe, hogy hatására a ciklus, melyben található, a következő iterációs lépést elvégzi.
- Goto: Segítségével tetszésünkre ugrálhatunk a megadott címkére bárhol a programunkban, meglehetősen kényelmesen. Ennek ellenére nem túl javasolt a használata, mivel kevésbé átláthatóbb tőle a program. Szerencsére minden goto-val írt program megírható nélküle is.

A könyv függelékében rátalálhatunk még az utasításokról szóló fejezetre is, mely különböző csoportokba sorolja be az előzőleg felsorolt utasításokat. Ezek lennének a: címkézett utasítások, kifejezésutasítások, összetett utasítások, kiválasztó utasítások, iterációs utasítások és vezérlésátadó utasítások

## 10.3. Programozás

### [BMECPP]

A Budapesti Műszaki Egyetem ezen könyvében a C++ rejtelseiről, újításairól kaphatunk egy átfogó bemutatást.

A bevezetést követő fejezetben a C++ nem objektumorientált újdonságairól számol be a könyv, ugyanakkor összehasonlításba kerül elődjével, a C-vel, rávezetve minket a C++-ban rejlő potenciálra és a rengeteg újításra, ami bekerült. Szó esik többek között: függvényparaméterek és visszatérési értékekről, a main függvényről, bool típusról, több bájtos sztringekről, változódeklarációról mint utasítás, függvénynevek túlterheléséről, függvényargumentumokról és még sok másról.

A harmadik fejezetben értelemszerűen a nyelv objektumorientáltságáról számol be, ami az egyik legfőbb tulajdonsága a C++ nyelvnek. Ez az attribútum ugyanis rengeteg különféle módon segíti a programnyelvet kényelmesebbé és átláthatóbbá tenni. Ahogyan azt a könyv is leírja: egységbe zárjuk az adatokat(encapsulation), az ezáltal kinyert egységet osztálynak nevezzük(class), ennek az osztálynak pedig az önálló példányait objektumoknak hívjuk(object). Persze ez nem csak ennyire egyszerű, sok más dologra is fel lehet használni ezt a fajta adat struktúrát, és sok mindent is követel egy osztály helyes használata. Ezekről mind beszámol a könyv ezen fejezete: tagfüggvények, adatrejtés, konstruktorok - dekonstruktorok, dinamikus adatkezelés(egy nagyon esszenciális rész), friend függvények, tagváltozó inicializálás, statikus tagok és beágyazott definíciók.

Ugorva egy pár fejezetet a könyvben találjuk meg az operátor túlterhelésről szóló részt, ami azért olyan jelentős, mert lehetővé teszi, hogy az olyan alap műveletekhez használt operátorokat, mint az összeadás +, kivonás -, szorzás \*, osztás /, nem csak egész vagy akár lebegőpontos típusú változókkal tudjuk használni, hanem az általunk osztályok segítségével elkészített mátrixok és komplex számokkal is. Találkozhatunk speciális esetekkel is, amikor például az operator==t kell definiálnunk hogyha éppen mélymásolatot szeretnénk, legfőképpen dinamikus adattag esetén.

Még egy keveset ugorván, eljutunk a kivételkezelést bemutató fejezethez, ami abból az okból kifolyólag jelentős, hogy C-vel ellentétben, itt egy sokkal strukturáltabb, átláthatóbb és könnyebben karbantartható hibakezelési lehetőség áll rendelkezésünkre. Ugyanakkor nem hiába nevezik kivétel-kezelésnek, mivel nem csak hibákat, de bármilyen általunk előre nem belátott eredmény orvosolni tudunk vele pl. egy if() throw segítségével.



## **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.

[PICI] Juhász, István, *Magas szintű programozási nyelvek I*, mobiDIÁK, 2008.

### 11.4. C

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

### 11.5. C++

[BMECPP] Benedek, Zoltán És 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](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.