Univerzális programozás

Így neveld a programozód!

DRAFT

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

PDF BY DBLATEX

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

DRAFT

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai Norbert, Bátfai Mátyás, Bátfai Nándor, Bátfai Margaréta, és Czanik András	2020. 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 https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai

DRAFT

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]

DRAFT

Tartalomjegyzék

DRAFT

DRAFT

DRAFT

Ábrák jegyzéke

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 (hogy lássuk mást is) példával.

Hogyan nyomjuk?

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

DRAFT

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

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

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

A DocBook XML 5.1 új neked?
Ez esetben forgasd a https://tdg.docbook.org/tdg/5.1/ könyvet, a végén találod az informatikai szövegek jelölésére használható gazdag „API” elemenkénti bemutatását.

DRAFT

I. rész

Bevezetés

DRAFT

1. fejezet

Vízió

1.1. Mi a programozás?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk?

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: http://esr.fsf.hu/hacker-howto.html!
- Olvasgasd aztán 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
- C kapcsán a [KERNIGHANRITCHIE] könyv adott részei.
- C++ kapcsán a [BMECPP] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a The GNU C Reference Manual, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf`
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [BMECPP] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, https://www.imdb.com/title/tt0478087/, benne a Monty Hall probléma bemutatása.
- Kódjátszma, https://www.imdb.com/title/tt2084970, benne a kódtörő feladat élménye.

DRAFT

- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.

DRAFT

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Futtatási videó:

https://youtu.be/bkB5i1uMoKo

2.2. 1. Feladat: Végtelen ciklusok

Egy mag kihasználása 100%-on: Két megoldás is készült a probléma feloldására. A két általános végtelen ciklus ("while(true); és for(;;);") felhasználásával, de ezeket tekinthetjük akár csak egy féle ciklusként is, hiszen ha átfordítjuk őket assembly-re, akkor a két forráskódunk, a nevén kívűl mindenben megegyezik. Na de miért használja ki egy mag százszázalékát? A válasz egyszerű. Azért mert egy mag egyszerre egy feladatot, tud ellátni, de a feladatok közt nagyon gyorsan tud váltani. Viszont, ha egy végtelen ciklust kap szerencsétlen, akkor kénytelen annak feltételét folyamatosan ellenőrizni, így nincs lehetősége, hogy váltson más feladatra vagy pihenjen.

Egy mag 0%-on: Hasonlóan indulunk el mint az előző esetén. Itt is megírjuk a kis végtelen ciklusunkat, viszont annyit módosítunk rajta, hogy adunk egy kis extra feladatot a processzornak azon túl, hogy a feltételt ellenőrizgesse. Ez lehet elsőre furcsán hangzik, de ha kap egy feladatott, ami méghozzá egy szüneteltetés ("sleep(1)") mégha csak egy nagyon kicsi időintervallumig is, de van lehetősége megpihenni, ami a processzor magnak bőven elég is lesz. Ezért látjuk azt, hogy közel/teljesen 0%-on fut a mag.

Az összes mag 100%-on: Ha már játszunk a magokkal, akkor játszunk rendesen. All in! Kicsit kiegészítjük a végtelen for ciklusunkat. Az első, hogy meghívjuk a megfelelő könyvtárat, ami jelen esetben az 'omp.h'. Erre azért lesz szükségünk, hogy parallax programmá alakíthassuk a programunkat. A parallax programok lényege, minden minőségi kifejtési igény nélkül, hogy kihasználja az összes erőforrást futtatáskor. Pontosan, ez kapóra is jön nekünk, mert ha már egy magot tudunk pörgetni maxon, akkor így az összeset tudjuk majd. És láss csodát, valóban!

A legnagyobb tanulság amit levonhatunk ebből a rövid kisértleből, az nem más, mint hogy a végtelen ciklusok ugyan hasznosak, de kilépési feltétel és megszakítás nélkül, rendesen el tudják foglalni a processzor magokat. Ami csak azért lehet problémás, mert akkor váltani se tudnak a feladatok között, amiből mi csak annyit érzékelünk, hogy gépünk nem végzi el a feladatát, sőt semmit se csinál, azaz lefagy. Ha lehet kerüljétek, de ha mindenképp kell, akkor is legyen benne feltétel a megszakításra.

DRAFT

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

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c,bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-w.c.

```
int
main ()
{
  for (;;);
  return 0;
}
```

vagy az olvashatóbb, de a programozók és fordítók (szabványok) között kevésbé hordozható

```
int
#include <stdbool.h>
main ()
{
  while(true);
  return 0;
}
```

```
$
            gcc -S -o infty-f.S infty-f.c
            $ gcc -S -o infty-w.S infty-w.c
            $ diff infty-w.S infty-f.S
            1c1
            <   .file "infty-w.c"
            ---
            >   .file "infty-f.c"
```

Egy mag 0 százalékban:

```
            #include <unistd.h>
            int
            main ()
            {
            for (;;)
                sleep(1);

            return 0;
            }
```

Minden mag 100 százalékban:

```
#include <omp.h>
int
main ()
{
#pragma omp parallel
{
for (;;);
}
return 0;
}
```

A **gcc infty-f.c -o infty-f -fopenmp** parancssorral készítve a futtathatót, majd futtatva, közben egy másik terminálban a **top** parancsot kiadva tanulmányozzuk, mennyi CPU-t használunk:

```
top - 20:09:06 up  3:35,  1 user,  load average: 5,68, 2,91, 1,38
Tasks: 329 total,   2 running, 256 sleeping,   0 stopped,   1 zombie
%Cpu0 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu6 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu7 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem :16373532 total,11701240 free, 2254256 used, 2418036 buff/cache
KiB Swap:16724988 total,16724988 free,       0 used. 13751608 avail Mem
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 5850 batfai    20   0   68360    932    836 R 798,3  0,0   8:14.23 infty-f
```

2.3. 2. Feladat: Lefagyott, nem fagyott, akkor most mi van?

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{
  boolean Lefagy(Program P)
  {
     if(P-ben van végtelen ciklus)
      return true;
     else
      return false;
  }
  boolean Lefagy2(Program P)
  {
     if(Lefagy(P))
      return true;
     else
      for(;;);
  }
  main(Input Q)
  {
    Lefagy2(Q)
  }
}
```

Mit for kiírni erre a `T1000(T1000)` futtatásra?

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

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

A fő probléma abban rejlik, hogy az if-else ágban egy ellentmondásba ütközünk. Mivel ha a P programban van végtelen ciklus, akkor a `Lefagy2` egy true értékkel, ellenben ha meg nincs benne, akkor visszatérne egy további for ciklussal, hogy tovább figyeljen, ami viszont ahhoz vezet, hogy akkor már lesz benne végtelen ciklus. Tehát ha nem fagyott le eddig, akkor mostmár le fog.

Szerencsére sok modern IDE-ban van olyan kisegítő lehetőség, mely figyelmeztet minket arra, ha programunk olyan végtelen ciklust tartalmaz, melynek nincs kilépési feltétele. Ez sokat segíthet, de még így is könnyen megtörténhet hogy figyelmetlenség miatt benne marad. És mivel ez csak egy figyelmeztetés és nem hibaüzenet, ezért a program fordítható és futtatható, így lehet jó ideig elő se jön a probléma.

2.4. 3. Feladat: Változók értékének felcserélése

Az alábbi programban három általános módját tekinthejük meg a változók közti értékcserének. A használat könnyítése érdekében, ez tartalmaz egy **switch** utasítást, hogy könnyen kiválaszthassuk, hogy melyik módszert szeretnénk alkalmazni

```
#include <stdio.h>
                #include <stdlib.h>
                int main()
                {
                    int a = 0;
                    int b = 0;
                    int tmp = 0;
                    int choice;
                    while(1)
                    {
                        printf("a = ");
                        scanf("%d",&a);
                        printf("\nb = ");
                        scanf("%d",&b);
                        printf("\n---x{[MENU]}x---\n");
                        printf("1 - Seged valtozoval\n");
                        printf("2 - Kivonassal\n");
                        printf("3 - Osszeadassal\n");
                        printf("9 - Exit\n");
                        fflush(stdin);
                        scanf("%d",&choice);

                        switch(choice)
                        {
                            case 1:
                                tmp = a;
                                a = b;
                                b = tmp;
                                printf("a = %d, b = %d\n", a, b);
                                break;

                            case 2:
                                a = a - b;
                                b = a + b;
                                a = b - a;
                                printf("a = %d, b = %d\n", a, b);
                                break;
```

```
                    case 3:
                        a = a + b;
                        b = a - b;
                        a = a - b;
                        printf("a = %d, b = %d\n", a, b);
                        break;

                    case 9:
                        break;
                    default:
                        printf("Invalid bemenet!\n");
                }
                break;
            }
            return 0;
        }
```

Nézzük meg a három értékcserét egyesével:

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

A segédváltós értékcserét egy egyszerű példával szemléltetném: Van az asztalon egy pohár almalé és egy pohár narancslé és szeretnénk a két pohár tartalmát felcserélni. Ez a feladat így nem más mint egy gordiuszi csomó. De mielőtt pánikba esünk, hívjunk segítségül egy harmadik, de üres poharat. Ez a pohár lesz a segédváltozónk, vagy jelen esetben segédpoharunk. Ha a segédpohárba átöntjük az almalevet, akkor az a pohár üres lesz, így abba átönthetjük a narancslevet. Ami által a narancsleves pohár is felszabadúl. Ezután a segédpohárból beleönthetjük az almalevet az immár üres pohárba. Egyszerű, nem igaz?

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

A segédváltozós értékcserével ellentétben ezt a kettőt csak számokkal tudjuk alkalmazni, mivel például a sztringek vagy boolenek kivonása és összeadása nem igazán értelmezhető. Itt a kivonásos példában azzal kezdjük, hogy az `a`-ból kivonjuk a `b`-t. A `b`-hez hozzáadjuk az `a`-t , ezzel kiegészítve, tehát megkapjuk az `a` értéket és végül `b`-ből kivonjuk az `a`-t és kész is vagyunk.

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

Végül az összeadásos: `a`-hoz hozzáadjuk `b`-t, tehát tartalmazza mindkét értéket. Utánna `b`-nél kivonjuk `a`-ból önmagát, így már csak az eredeti `a`-t tartalmazza. Végül az `a`-t is meghatározzuk, úgy hogy kivonjuk

belőle a b-t, ami ugye az új a , tehát az új b lesz. Kicsit katyvasz, de csak egyszer kell megérteni és követhető lesz.

2.5. 4. Feladat: Labdapattogás

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

Megoldás forrása:

Fordítás:

```
gcc bouncy0.cpp -o bouncy0 -lncurses
```

Eleve az adott példa alapján megírtam a program if-es változatát, mely kommentként szerepel a forrásban. Ezután az if nélküli verzióban az if-ek feladatát kiszerveztem for ciklusoknak, melyek csak akkor lépnek életbe, ha a '(DVD)' karakter eléri az ablak valamelyik szélét, majd ez alapján x vagy y értékét negatív előjelűvé cseréljük, azaz visszapattinjuk a falról. Kicsit hasonlít a technika arra mint amit az Atari Blockbreaker játékában is alkalmaztak, hogy visszapattanjon a labda a játékosról, azzal a kis extrával, hogy ott vektorokat alkalmaztak, így a vektor hossz alapján lehetet kicsit befolyásolni a visszapattanási szöget. Egyszerű megoldás, de így nem volt olyan statikus és kiszámítható a labda új iránya, ezzel változatosabbá téve a játékot. Mellesleg az én példámban 'o' helyett a '(DVD)' szerepel, mivel kisebb koromban szerettem nézni a DVD lejátszókon a képernyőkimélőt, melyen a felirat minden visszapattanás után színt váltott. Ezt egy kis "easteregg"-nek szántam.

```
#include <stdio.h>
#include <curses.h>
#include <unistd.h>
int main()
{
    WINDOW *ablak;
    ablak = initscr();
    int x = 0;
    int y = 0;
    int delX = 1;
    int delY = 1;
    int mx;
    int my;

    while(true)
    {
        printf("\033[0;32m");
        getmaxyx(ablak, my, mx);
        mvprintw(y, x, "(DVD)");
        refresh();
        usleep(100000);

        clear();
```

```
                x = x + delX;
                y = y + delY;
```

Először behívjuk a kellő könyvtárakat, felvesszük a változókat és közben az `ablak`-ban inicializáljuk a terminál ablakot, amit majd a `getmaxyx()` függvényben meghatározzuk, az ablak maximális szélességét és magasságát. Ez ugye elengedhetetlen egy labdapattogtatós program esetén. Továbbá mivel a while-on belül van, így az ablak méretváltoztatására is jól reagál, mondhatni rekurzív megjelenítést érünk el. Ezen kívűl a `mvprintw()` függvény pedig felel a kis DVD felíratunk kirajzolásáért az adott x és y koordinátákon.

```
                /*
                if(x <= 0)
                {
                    delX = delX * -1;
                    //printf("\033[0;31m");
                }
                if(x >= mx-1)
                {
                    delX = delX * -1;
                    //printf("\033[0;32m");
                }
                if(y <= 0)
                {
                    delY = delY * -1;
                    //printf("\033[0;33m");
                }
                if(y >= my-1)
                {
                    delY = delY * -1;
                    //printf("\033[0;34m");
                }
                */
                for(;x <= 0;)
                {
                    delX = delX * -1;
                    break;
                }
                for(;x >= mx-1;)
                {
                    delX = delX * -1;
                    break;
                }
                for(;y <= 0;)
                {
                    delY = delY * -1;
                    break;
                }
                for(;y >= my-1;)
                {
                    delY = delY * -1;
```

```
                    break;
                }
            }
            return 0;
        }
```

Maga a visszapattanás pedig pofon egyszerű. Megnézzük hogy melyik szélét érte el az ablaknak. Itt ugye sok lehetőség nincs, csak négy. Ezt ugye a felírat koordinátáival és az ablak maximális méretinek a segítségével tesszük meg (itt kihasználjuk, hogy a bal szél az x = 0, az alja meg az y = 0). Ha elérte egyszerűen megfordítjuk a menetirányát, ugy hogy az ezért felelős `delX`-nek és `delY`-nak megcseréljük az előjelét (laikusoknak: megszorozzuk (-1)-el).

Ezeket meg ismételjük a végtelenségig, vagy legalábbis a megunásig. Mondjuk én kiskoromban huzamosabb ideig is el tudtam nézni, várva hogy hátha más szögbe pattan vissza, de hiába, nem lehet minden Pong...

2.6. 5. Feladat: 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/KRZlt1ZJ3qk, .

Megoldás forrása: `bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bogomips.c`

Nyílván nem én vagyok az egyetlen aki korábban nem hallott még a BogoMIPS-ről, de ezzel alapvetőleg nincs is baj. Két kifejezés játékos összemosásából jött létre: Bogus (kamu, nem valódi) és MIPS, ami két dolognak is a rövidítése. Az első a tudományosan hangzó: Millions of Instructions per Second (másodpercenként végrehajtott műveletek millióban) és a második, de találóbb jelentése: Meaningless Indication of Processor Speed (értelmetlen jellemzése a processzor sebességének). Ez mutatja is, hogy ez inkább egy vicces apróság, mint egy komoly és pontos teszt, de azért elemzése megér egy misét. Egyébként már valószínűleg mindenki találkozott ezzel, bár esélyes hogy nem vette észre, vagy csak jelentéktelen random karaktereknek vette, mint a többi hardware checket bootoláskor, mintha csak valami mátrix easter egg lenne...

Alapvetőleg egy ehhez hasonló értéket kell kapnunk:

```
386SX           clock * 0.14
386DX           clock * 0.18
486Cyrix/IBM    clock * 0.33
486SX/DX/DX2    clock * 0.50
586             clock * 0.39
```

Ha ennél jóval kisebbet mutat a gépünk akkor érdemes ránézni a CPU beállításokra, vagy a cache-re. Itt még meg kell jegyezni, hogy a BogoMIPS érték igazából két dologra jó. Le lehet ellenőrizni, hogy a processzor a beállításainak megfelelően működik és persze lehet izmozni a haveroknak, hogy kinek mekkora...

DRAFT

A program lényege, hogy bináris eltolással végig lépkedjünk egy long long típusú változón, miközben mérjük, hogy mennyi ideig tart ez neki. Ezután a kapott hatalmas számot el kell osztanunk egy számmal (ez egy tetszőleges szám lehet, a cél csak az, hogy az eredmény 100-1000 között legyen). Ennek a számnak nincs mértékegysége, de ha összehasonlítjuk más gépek értékével, melyeken ugyan ezt a programot futtatjuk, akkor egyszerűen megállapítható, hogy melyik gép processzora számít gyorsabban.

Ha a program while() ciklusa előtt létrehozunk egy változót és ennek értékét ciklus futásonként növeljük, akkor visszakapjuk a long long típus méretét bitben.

```
#include <stdio.h>

    int main()
    {
        int wordLength = 0, word = 1;

        do
        {

            wordLength++;

        } while(word <<= 1);

        printf("A szó ezen a gépen %d bites\n", wordLength);

        return 0;
    }
```

A szóhossz program is nagyon hasonló csak ez még egyszerűbb alapvetőleg. Bitenként lépkedünk és minden lépésnél növeljük a `wordLength`-et, azaz megszámoljuk lényegében a bitjeit. Így megkapván a szóhosszt:

```
A szó ezen a gépen 32 bites
```

2.7. 6. Feladat: Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Nevét Larry Page után kapta, a Google cégóriás egyik alapítójáról. Úgy gondolták, hogy a keresési szó szerepelésén túl úgy lehetne legjobban rangsorolni az oldalakat, hogyha a rájuk mutató oldalak száma és minősége szerint rangsorolnák a találatokat. Az ötlet mögött az a feltételezés ált, hogyha megbízható oldalak hivatkozzák meg az adott oldalt, akkor lényegében azok az oldalak validálják annak megbízhatóságát

és növelik annak az esélyét, hogy a keresett tartalom előre kerül. Valószínűleg igazuk is lett, mert a Google ezáltal hamar a legnépszerűbb keresőmotorrá vált.

Az algoritmus alapból (0. iteráció) minden oldalnak azonos pageranket határoz meg (1/N, ahol N a lapok száma). Ezután a rá mutató lapok előző iterációban kiszámítóttot pagerankjét elosztjuk a rá mutató lapok számával. Könnyen észrevehető, hogy ezt a végtelenségig csinálhatnánk.

Ennek kiküszöbölésére hivatott a damp factor bevezetése. Igazából nem ez a fő célja, hanem a Sztochasztikus szörföző metódushoz szükséges, ami azt próbálja szimulálni, hogy a weben véletlen szerűen nézelődő felhasználó milyen valószínűséggel talál rá az oldalra. De sokat segít hogy a számításaink ne menjenek a végtelenségig.

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

double distance (double PR[], double PRv[], int db)
{
    double dist = 0.0;
    for(int i = 0; i < db; i++)
    {
        dist += pow((PRv[i] - PR[i]), 2);
    }
    return sqrt(dist);
}

void kiir (double list[], int db)
{
    for(int i = 0; i < db; i++)
    {
        printf("PageRank [%d]: %lf\n", i, list[i]);
    }
}
```

Behívjuk először a szükséges könyvtárakat, mint például a `math.h`-t, amit a következő bekezdésben fogok részletezni.

A `distance()` függvényről külön elméláznék. Az eredeti terv az volt, hogy `abs()` segítségével a különbség abszolút értékével számolok. De ezt pár hibás futtatás után ki kellett, hogy szedjem, mivel mint kiderült, C-ben ez csak int típust tud kezelni, azaz egész számokat. Viszont a `PR` és `PRv` is double típus így nem lenne szerencsés konvertálni. Így kénytelen vagyok a négyzetértékkel számolni, majd végül abból gyököt vonni, ami ugyan azt az eredményt adja, hogy csak pozitív értéket vehet fel vagy nullát (mivel hogy két negatív szám szorzata pozitív, az pedig nem fordulhat elő, hogy egyik negatív míg a másik pozitív, hisz ugyan az a művelet). Megjegyzem, hogy az előbb említett két függvény indokolja a `math.h` headert és a -lm kapcsolót a fordításhoz.

Van még itt egy `kiir()`, de ez csak szimplán kiiratja a 4 darab oldal pagerank értékét.

```
int main(void)
{
    double L[4][4] =
    {
```

```
                {0.0, 0.0, 1.0/3.0, 0.0},
                {1.0, 1.0/2.0, 1.0/3.0, 1.0},
                {0.0, 1.0/2.0, 0.0, 0.0},
                {0.0, 0.0, 1.0/3.0, 0.0}
                };

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

                for(;;)
                {
                    for(int i = 0; i < 4; i++)
                    {
                        PR[i] = PRv[i];
                    }
                    for(int i = 0; i < 4; i++)
                    {
                        PRv[i] = 0;
                        for(int j = 0; j < 4; j++)
                        {
                            PRv[i] += L[i][j] * PR[j];
                        }
                    }
                    if(distance(PR, PRv, 4) < 0.00001)
                    {
                        break;
                    }
                }
                kiir (PR, 4);
                return 0;
            }
```

Itt először nézzük meg miből élünk. Van itt egy `L` szükszavú elnevezésű double típusú mátrixunk, mely a linkeket tartalmazza. Van még nekünk két szintén double típusú tömbünk `PR` és `PRv` néven. Ezek a pagerank értéket tartalmazzák, az egyik a jelenlegit (`PR`), a másik az iteráció előző körében szereplő pagerank értékét mutatja az oldalinknak (`PRv`).

A következő rész a végtelenciklusunk, ahol a munka oroszlánrésze zajlik.

```
                    for(int i = 0; i < 4; i++)
                    {
                        PR[i] = PRv[i];
                    }
```

Itt egyszerűen átadjuk a pagerank értékeket az előző körből a jelenleginek (ezért fontos, hogy a `PRv`-nek legyen kezdőértéke).

```
            for(int i = 0; i < 4; i++)
                    {
                        PRv[i] = 0;
```

```
                    for(int j = 0; j < 4; j++)
                    {
                        PRv[i] += L[i][j] * PR[j];
                    }
                }
```

Itt következik egy dupla for ciklus. Ez azért fontos mert itt mátrixal fogunk dolgozni, ami ugye egy két dimenziós tömb. Először lenullázzuk a `PRv`-t, majd új értéket adunk neki, azáltal, hogy a jelenlegi `PR` minden elemét összeszorozzuk az `L` linkmátrixal (mátrix szorzása vektorral).

```
            if(distance(PR, PRv, 4) < 0.00001)
                    {
                        break;
                    }
                }
                kiir (PR, 4);
                return 0;
```

Végül de nem utolsó sorban ez az if. Az elején taglaltam, hogy ez a számítás a végtelenig mehetne és hogy ezt elkerüljük felhasználjuk a damp factort (a 0.00001 az). Tehát ha a `distance()` értéke eléggé megközelíti a nullát akkor egyszerűen megszakítjuk a végtelen ciklust (break) és kiírjuk az eredményt a felhasználónak.

Fordítás:

```
gcc PageRank.c -o pr -lm
```

Kimenet:

```
PageRank [0]: 0.090907
PageRank [1]: 0.545460
PageRank [2]: 0.272725
PageRank [3]: 0.090907
```

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

A Monty Hall paradoxon nevét a Monty Hall Show (és műsorvezetője akit meglepő mód szintén így hívtak) után kapta. A probléma megértéséhez vázolnám röviden a műsor menetét: Adott volt három ajtó. Az egyik ajtó mögött egy méregdrága sportkocsi volt, míg a másik kettő mögött egy-egy kecske. A játékosnak nem

volt más dolga mint megtippelnie, hogy melyik ajtó rejti a mesés fődíjat. Miután a játékos megtette a voksát, utánna Monty feltárta a maradék két ajtó közül az egyiket amelyik kecsét rejtett és lehetőséget adott a játékosnak az újraválasztásra. Na de joggal kérdezhetnénk, hogy hol ebben a paradoxon. A probléma ott kezdődött mikor előkerültek a műsor statisztikái, miszerint akik újra választottak ajtót, miután Monty felfedte az egyik ajtó titkát, azok majdnem kétszer olyan gyakran nyertek, mint akik nem.

Amikor elsőnek választ ajtót a játékos, az egy egyszerű valószínűség számítási probléma, hiszen három ajtónk van, de csak egy fődíj, azaz minden ajtónak 1/3 esélye van, hogy az autót rejti. Lényegében mindegy melyik ajtót választja az ember. A probléma gyökere abból adódik, hogy mikor Monty választ egy ajtót ami mögött ő tudja, hogy kecske lesz és felkínálja az újraválasztás lehetőségét, a dolgok nem úgy folytatódnak ahogy az logikus lenne (legalábbis a statisztika szerint). Ugyebár logikusan azt gondolnánk, hogy maradt két ajtó, az egyik mögött még mindig ott a díj, mindkét ajtónak ugyan annyi esélye van (50%-50%), tehát felesleges lenne újat választani. Viszont ahogy azt az analitika mutatja, a dolgok nem így mennek, hanem az ajtók esélyei, maradnak az eredetiek, viszont mikor Monty felfedi az egyik ajtót, akkor viszont annak az esélyei, átszállnak a megmaradt ajtóra, amivel jelentősen romlanak eredeti esélyeink (1/3:2/3).

```
trials=1000000
trial = sample(1:3, trials, replace=T)
player = sample(1:3, trials, replace=T)
host=vector(length = trials)

for (i in 1:trials)
{

    if(trial[i]==player[i])
    {

        leftOver=setdiff(c(1,2,3), trial[i])

    }
    else
    {

        leftOver=setdiff(c(1,2,3), c(trial[i], player[i]))

    }

    host[i] = leftOver[sample(1:length(leftOver),1)]

}

stayingChance= which(trial==player)
changingVec=vector(length = trials)

for (i in 1:trials)
{

    changeIndexes = setdiff(c(1,2,3), c(host[i], player[i]))
    changingVec[i] = changeIndexes[1]
```

```
    }

    changingChance = which(trial==changingVec)


    sprintf("Kiserletek szama: %i", trials)
    length(stayingChance)
    length(changingChance)
    length(stayingChance)/length(changingChance)
    length(stayingChance)+length(changingChance)
```

Daraboljuk fel egy kicsit:

```
trials=1000000
trial = sample(1:3, trials, replace=T)
player = sample(1:3, trials, replace=T)
host=vector(length = trials)
```

Beállítjuk a kisérletek számát egymillióra, majd kisorsoljuk a nyertes ajtót és a játékos tippjét (nem egyet, hanem 1000000 darabot) majd elkészítünk egy vektort. Ezután jön a játék első köre:

```
for (i in 1:trials)
{

    if(trial[i]==player[i])
    {

        leftOver=setdiff(c(1,2,3), trial[i])

    }
    else
    {

        leftOver=setdiff(c(1,2,3), c(trial[i], player[i]))

    }

    host[i] = leftOver[sample(1:length(leftOver),1)]

}
```

Megvizsgálunk két eshetőséget mielőtt feltárunk egy ajtót. Az első hogy a játékos bár nem tudja, de telibe eltalálta a nyertes ajtót, ilyenkor a fennmaradt két ajtó kerül a listánkra, mint hátralévő. A második, hogy másik ajtóra tippelt, ilyenkor egyszerűen a harmadik, még egyik fél által sem választott ajtót használjuk.

Ehhez mindkét esetben a `setdiff()` függvényt alkalmazzuk, amely a matematikai halmaz elmélet alapján két halmaz különbségét határozza meg. Ezeket az ajtókat eltároljuk a `leftOver`-be, hogy majd a `host`-ban kiválasztjuk hogy a maradék egy vagy két ajtó közül melyiket fedjük fel jelképesen, azaz melyik esik ki a játékból.

```
stayingChance= which(trial==player)
changingVec=vector(length = trials)

for (i in 1:trials)
{

    changeIndexes = setdiff(c(1,2,3), c(host[i], player[i]))
    changingVec[i] = changeIndexes[1]

}

changingChance = which(trial==changingVec)
```

A `stayingChance`-be eltároljuk, hogy az összes esetből hányszor nyernénk, ha nem változtatunk, tehát ha elsőre eltatlájuk. Aztán egy forciklusban végig megyünk az összes eseten és megnézzük, hogy mi lesz akkor ha ajtót váltunk. Az új ajtó indexe a `changeIndexes`-be kerül, azáltal, hogy megnézzük, hogy melyik az az ajtó, ami megmarad, ha kivesszük a választott ajtónkat és a feltárt ajtót. Majd a `changingVec` vektorban eltároljuk ezeket az indexeket. Végül a `changingChance`-ben megvizsgáljuk, hogy ajtócserével hány esetben nyernénk.

```
sprintf("Kiserletek szama: %i", trials)
length(stayingChance)
length(changingChance)
length(stayingChance)/length(changingChance)
length(stayingChance)+length(changingChance)
```

Végül pedig kiíratjuk a kapott eredményeket. Először kiírjuk a kísérletek számát, majd hogy hány esetben nyert az ajtó váltás nélküli (length(stayingChance)). Ezután jön hogy hány esetben nyert az ajtóváltással (length(changingChance)). Végül pedig kis ellenörzésként össze van hasonlítva, hogy az első érték majdnem a fele a másodiknak és hogy a kettő összege valóban egyezik a kísérletek számával.

Lássuk a futtatást:

```
[1] "Kiserletek szama: 1000000"
[1] 333982
[1] 666018
[1] 0.5014609
[1] 1000000
```

2.9. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

A Brun tétel röviden azt mondja ki, hogy az ikerprímek reciprokértéke egy speciális konstanshoz tart, azaz konvergál. Ez a konstans az úgynevezett Brun-konstans vagy Brun szám, melynek a pontos értékét csak becsülni tudjuk, de kb. 1.902160583104. De mik is azok az ikerprímek? Olyan prímszám párok, melyeknek különbsége pontosan 2, ilyen például a 3 és az 5 vagy a 11 és 13. Azt már Euklidesz bebizonyította, hogy a prím számok száma végtelen, viszont az még a mai napig kérdéses, hogy vajon a ikerprímek száma is végtelen-e. És akkor még nem is beszéltünk a prímnégyesekről, ami megint csak egy különleges esett, ugyani azok olyan prímek ahol teljesül, hogy ha n prím, akkor 10n+1, 10n+3, 10n+7, 10n+9 is prímek. Na de ne térjünk el a tárgytól. Az ikerprímek helyzete azért is macerás, mert könnyű lenne rávágni, hogy ha a prímek végtelenek akkor az ikerprímek is, de ez nem ilyen egyszerű, mert minél nagyobb értékeket vizsgálunk, annál nagyobbak a prímek közti távolságok is (tehát egyre ritkábbak lesznek az ikrek). A kérdés hogy van-e vége és ebben talán segít kicsit a Brun tétel.

A következő példaprogramban 1.000.000 adjuk össze az ikrek reciprokját, majd végül egy koordináta rendszerben ábrázoljuk, hogy hogyan jut egyre közelebb az érték a Brun-konstanshoz. Kezdjük is az elemzést.

```
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/ ↩
   or modify
#    it under the terms of the GNU General Public License as  ↩
   published by
#    the Free Software Foundation, either version 3 of the  ↩
   License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be  ↩
   useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty ↩
    of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See  ↩
   the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public  ↩
   License
#    along with this program.  If not, see <http://www.gnu.org/ ↩
   licenses/>
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 program Bátfai Norbert munkája és R nyelven íródott. A Monty Hall Paradoxonos feladat is szintén R nyelvben készült, mely szakspeciális programozási nyelv. Statisztikai kalkulációk és statisztikai ábrázolás során használják a leggyakrabban. Daraboljuk fel picit:

```
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))
}
```

A licenceket most lecsíptem róla, azt nem részletezném. Amivel kezdünk, az az hogy meghívjuk a matlab függvénykönyvtárat. Utána létrehozzuk az `stp` függvényt.

```
    primes = primes(x)
```

Elsőnek megkeressük a prímeket.

```
diff = primes[2:length(primes)]-primes[1:length(primes)-1]
idx = which(diff==2)
```

Megnézzük az egymást követő prímek közti különbséget amit eltárolunk a `diff`-ben, ha ez kettő, az azt jelenti, hogy a két prím ikerprímet alkot és eltároljuk az indexét az elsőnek, az `idx` változóba.

```
t1primes = primes[idx]
t2primes = primes[idx]+2
```

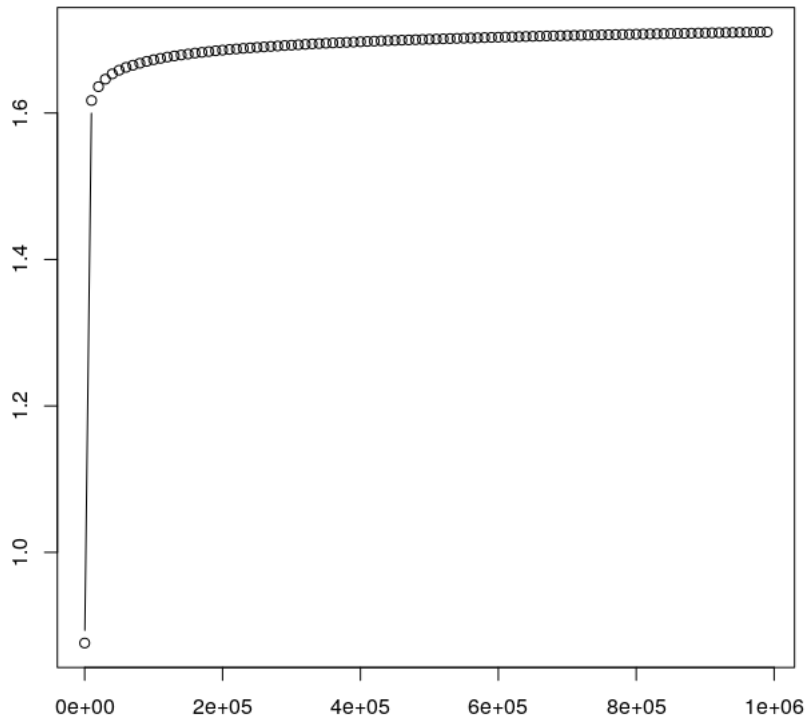
Az előző `idx` indexértékkel megállapítjuk a két prímet.

```
    rt1plust2 = 1/t1primes+1/t2primes
    return(sum(rt1plust2))
```

A két prímnek a reciprokát összeadjuk, ez lesz a `rt1plust2`, majd a függvény visszatér ennek a szummájával. Az `stp` ennyi volt, most pedig nézzük mi maradt még:

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

Az `x`-be készítünk egy sorozatot 13-tól 1000000-ig a `seq()` függvénnyel. Az `y`-ban pedig összepárosítjuk az `x`-et az `stp`-vel, azaz meghívjuk a függvényt. Majd végül de nem utolsósorban a `plot` segítségével ábrázoljuk a számításainkat. Ha minden jól ment akkor ezt kell kapnunk:



Gyönyörűen mutatja, hogy hogyan közelíti meg a Brun-konstanst.

Werkfilm

- https://youtu.be/VkMFrgBhN1g
- https://youtu.be/aF4YK6mBwf4

2.10. Malmo - Csiga

Ez volt az első önálló feladatunk a Minecraft Malmo platformon. Ezen még nagy érződnek a kezdetleges szárnypróbálgatások. Új volt mind a malmo, mind a python mindkettőnknek (Tutor Tünde), de mondhatom, hogy hamar megszerettük mindkettőt. Ebben a feladatban Stevenek még nagyon egyszerű dolga van: Csiga mozgásban kell feljebb és feljebb jutnia a RedFlowerHell arénában miközben fentről vészesen közeledik a láva.

Többféle képpen is próbáltuk megközelíteni a problémát, az alap koncepciónk az az volt, hogy egy dupla for ciklus segítségével döntse el, hogy mikor mennyit menjen és hogy mikor forduljon. Ezt végül is elvetettük, mert nem volt túl átgondolt és inkább újragondoltuk. Nyílván ezek a megoldások röhejesnek tűnhetnek, de még az alap szintaktikáját tanultuk a nyelvnek és a környezetnek.

Nyílván a második nekifutás se tökéletes, sok sebből vérzik és visszanézve már azt mondhatjuk, hogy nagyon kezdetleges a mostani kódjainkhoz (hálisten, hiszen ezek szerint tanultunk belőle). Itt az alap koncepció az volt, hogy Steve meghatározott ideig megy előre ezáltal előbb vagy utóbb elérve a szakasz végét. Akkor fordul egyet és növeli a `tmp` nevű változót. Ha ez a `tmp` eléri a 4-et az azt jelenti, hogy Steve körbeért, tehát mehet egyel feljebb.

Kőkorszaki megoldás, de ismerkedésnek jó volt.

Python:

```
 from __future__ import print_function
#  ↩
   ------------------------------------------------------------------------

# Copyright (c) 2016 Microsoft Corporation
#
# Permission is hereby granted, free of charge, to any person  ↩
   obtaining a copy of this software and
# associated documentation files (the "Software"), to deal in  ↩
   the Software without restriction,
# including without limitation the rights to use, copy, modify, ↩
    merge, publish, distribute,
# sublicense, and/or sell copies of the Software, and to permit ↩
    persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall  ↩
   be included in all copies or
# substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY  ↩
   KIND, EXPRESS OR IMPLIED, INCLUDING BUT
# NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR ↩
    A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT  ↩
   HOLDERS BE LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, ↩
    TORT OR OTHERWISE, ARISING FROM,
```

```
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER ↩
    DEALINGS IN THE SOFTWARE.
#  ↩
   ------------------------------------------------------------------------------------------------


# Tutorial sample #2: Run simple mission using raw XML

# Added modifications by Norbert Bátfai (nb4tf4i) batfai. ↩
   norbert@inf.unideb.hu, mine.ly/nb4tf4i.1
# 2018.10.18, https://bhaxor.blog.hu/2018/10/18/malmo_minecraft
# 2020.02.02, NB4tf4i's Red Flowers, http://smartcity.inf. ↩
   unideb.hu/~norbi/NB4tf4iRedFlowerHell


from builtins import range
import MalmoPython
import os
import sys
import time
import random
import json
import math

if sys.version_info[0] == 2:
    sys.stdout = os.fdopen(sys.stdout.fileno(), 'w', 0)  #  ↩
       flush print output immediately
else:
    import functools
    print = functools.partial(print, flush=True)

# Create default Malmo objects:

agent_host = MalmoPython.AgentHost()
try:
    agent_host.parse( sys.argv )
except RuntimeError as e:
    print('ERROR:',e)
    print(agent_host.getUsage())
    exit(1)
if agent_host.receivedArgument("help"):
    print(agent_host.getUsage())
    exit(0)

# -- set up the mission -- #
missionXML_file='nb4tf4i.xml'
with open(missionXML_file, 'r') as f:
    print("NB4tf4i's Red Flowers (Red Flower Hell) - DEAC- ↩
       Hackers Battle Royale Arena\n")
    print("NB4tf4i vörös pipacsai (Vörös Pipacs Pokol) - DEAC- ↩
```

```
                Hackers Battle Royale Arena\n\n")
            print("The aim of this first challenge, called nb4tf4i's ↩
                red flowers, is to collect as many red flowers as ↩
                possible before the lava flows down the hillside.\n")
            print("Ennek az első, az nb4tf4i vörös virágai nevű ↩
                kihívásnak a célja összegyűjteni annyi piros virágot, ↩
                amennyit csak lehet, mielőtt a láva lefolyik a ↩
                hegyoldalon.\n")
            print("Norbert Bátfai, batfai.norbert@inf.unideb.hu, https ↩
                ://arato.inf.unideb.hu/batfai.norbert/\n\n")
            print("Loading mission from %s" % missionXML_file)
            mission_xml = f.read()
            my_mission = MalmoPython.MissionSpec(mission_xml, True)
            my_mission.drawBlock( 0, 0, 0, "lava")


        class Hourglass:
            def __init__(self, charSet):
                self.charSet = charSet
                self.index = 0
                #self.pitch = 0 #ide tettem a pitch miatt
            def cursor(self):
                self.index=(self.index+1)%len(self.charSet)
                return self.charSet[self.index]

        hg = Hourglass('|/-\|')

        class Steve:
            def __init__(self, agent_host):
                self.agent_host = agent_host
                self.x = 0
                self.y = 0
                self.z = 0
                self.yaw = 0
                self.pitch = 0
                self.lookingat = 0
                self.nof_red_flower = 0

            def run(self):
                world_state = self.agent_host.getWorldState()
                i = 2
                j = 1
                tmp = 1
                #Loop until mission ends:
                while world_state.is_mission_running:
                    print("--- nb4tf4i arena ↩
                        ----------------------------------\n")

                    if world_state. ↩
                        number_of_observations_since_last_state != 0:
```

```
                sensations = world_state.observations[-1].text
                print("    sensations: ", sensations)
                observations = json.loads(sensations)
                nbr3x3x3 = observations.get("nbr3x3", 0)
                print("    3x3x3 neighborhood of Steve: ",  ↩
                   nbr3x3x3)

                if "Yaw" in observations:
                    self.yaw = int(observations["Yaw"])
                if "Pitch" in observations:
                    self.pitch = int(observations["Pitch"])
                if "XPos" in observations:
                    self.x = int(observations["XPos"])
                if "ZPos" in observations:
                    self.z = int(observations["ZPos"])
                if "YPos" in observations:
                    self.y = int(observations["YPos"])

                print("    Steve's Coords: ", self.x, self.y,  ↩
                   self.z)
                print("    Steve's Yaw: ", self.yaw)
                print("    Steve's Pitch: ", self.pitch)

                if "LineOfSight" in observations:
                    LineOfSight = observations["LineOfSight"]
                    self.lookingat = LineOfSight["type"]
                print("    Steve's <): ", self.lookingat)
                """
                if self.lookingat == "red_flower":
                    print("    viraaag!!4!!negy!!")
                    self.agent_host.sendCommand("move 1")

                if "Pitch" in observations:
                    self.pitch = int(observations["Pitch"])

                self.agent_host.sendCommand( "Pitch .3" )
                time.sleep(.5)
                if self.lookingat == "dirt":
                    print("FOOOLD!!!")
                """

                    #Mozgás

                self.agent_host.sendCommand( "move 1" )
                time.sleep(1.2*i)
                self.agent_host.sendCommand( "move 0" )
                time.sleep(.5)
                if tmp == 4:
                    tmp = 0
```

```
                            self.agent_host.sendCommand( "jump 1" )
                            time.sleep(.5)
                            self.agent_host.sendCommand( "move 1" )
                            time.sleep(.5)
                            self.agent_host.sendCommand( "jump 0" )
                            time.sleep(.5)
                            self.agent_host.sendCommand( "move 0" )
                            time.sleep(.5)
                        self.agent_host.sendCommand( "turn 1" )
                        time.sleep(.5)
                        self.agent_host.sendCommand( "turn 0" )
                        time.sleep(.5)
                        tmp = tmp + 1
                        i = i + 1


                    world_state = self.agent_host.getWorldState()



                    """
                    for i in range(64):
                            for j in range(4):
                                self.agent_host.sendCommand( "move 1" )
                                time.sleep(1.7*i)
                                self.agent_host.sendCommand( "move 0" )
                                time.sleep(.5)
                                self.agent_host.sendCommand( "turn 1" )
                                time.sleep(.5)
                                self.agent_host.sendCommand( "turn 0" )
                                time.sleep(.5)
                            self.agent_host.sendCommand( "jump 1" )
                            time.sleep(.5)
                            self.agent_host.sendCommand( "move 1" )
                            time.sleep(.5)
                            self.agent_host.sendCommand( "jump 0" )
                            time.sleep(.5)
                    """

                    #for i in range(64):
                        #ide tettem a pitch miatt

        num_repeats = 1
        for ii in range(num_repeats):

            my_mission_record = MalmoPython.MissionRecordSpec()

            # Attempt to start a mission:
            max_retries = 6
            for retry in range(max_retries):
```

```
                try:
                    agent_host.startMission( my_mission,  ↩
                       my_mission_record )
                    break
                except RuntimeError as e:
                    if retry == max_retries - 1:
                        print("Error starting mission:", e)
                        exit(1)
                    else:
                        print("Attempting to start the mission:")
                        time.sleep(2)

            # Loop until mission starts:
            print("   Waiting for the mission to start ")
            world_state = agent_host.getWorldState()

            while not world_state.has_mission_begun:
                print("\r"+hg.cursor(), end="")
                time.sleep(0.15)
                world_state = agent_host.getWorldState()
                for error in world_state.errors:
                    print("Error:",error.text)

            print("NB4tf4i Red Flower Hell running\n")
            steve = Steve(agent_host)
            steve.run()
            print("Number of flowers: "+ str(steve.nof_red_flower))

        print("Mission ended")
        # Mission has ended.
```

DRAFT

3. fejezet

Helló, Chomsky!

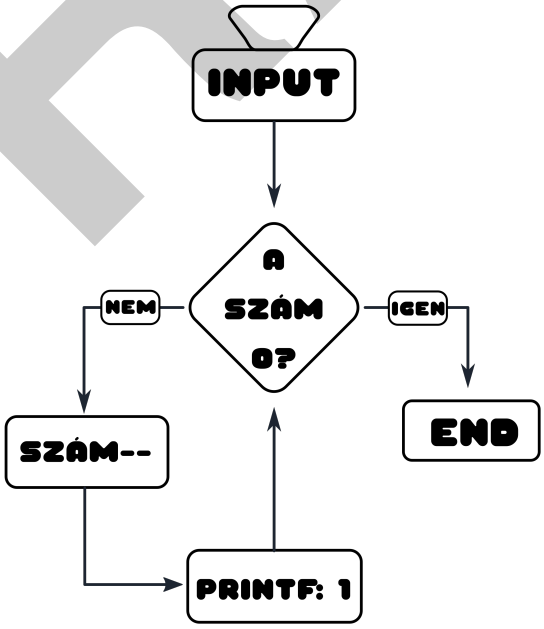
3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Az unáris számrendszer az egyes számrendszer. Ez azt jelenti, hogy kizárólag az ’1’-es számmal dolgozik, így leegyszerűsítve a decimálisból (10-es számrendszer) unárisba való átváltás annyit tesz, hogy annyiszor írjuk le az egyest, amennyi a decimális szám értéke (pl.: a 3: 111 vagy a 8: 111 11111). Gyakran szokták az egyeseket ötösével csoportosítani, a könnyebb olvasás érdekében. Pl.: A klasszikus ábrázolás, mikor a rabok a börtön falán vonalakkal jegyzik fel az eltelt napok számát.



DRAFT

Az ábrán a Turing gép ezt úgy oldja meg, hogy az adott számból nulla nem lesz. Közben minden kivonás után eltárolja a levont egyeseket. A műveletet az utolsó számjegytől kezdi. Ha ez az érték nulla, akkor kilenccel folytatja a 'kék' állapottal, majd addig folytatja a kivonást míg az megint nulla lesz. Ha nem nullával kezdődik, akkor addig léptetjük, míg nulla értéket nem találunk. A kivonások számát eközben eltároljuk.

Ma már kicsit egyszerűbb a helyzet amit az alábbi példa is szemléltet:

```
#include <stdio.h>

   void converter(int dec)
   {
       if (dec != 0)
       {
           converter(dec - 1);
           printf("1");
       }
   }

   int main()
   {
       int dec;

       printf("Ird be a valtani kivant decimalis szamot: \ ↩
          n");
       scanf("%d", &dec);

       converter(dec);

       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:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

Mi is az a BNF? A BNF környezet független szintaxisokat leíró szintaxis, tehát lényegében egy nyelvtan a nyelvtanokhoz.

A C utasítás BNF megfogalmazása:

```
<utasítás>::=
        <összetett_utasítás>
        <kifejezés>; (értékadás pl, num=10)
        if(<kifejezés>) <utasítás>
        else if(<kifejezés>) <utasítás>
        else <utasítás>
        switch (<kifejezés)
        <egész_konstans_kifejezés> : <utasítás>
        goto <azonosító>;
        <azonosító> : <utasítás>
        break; continue; return<kifejezés>;
        or(<kifejezés1><kifejezés2><kifejezés3>) <utasítás>
        while(<kifejezés>) <utasítás>
        do <utasítás> while<kifejezés>
        ; (üres utasítás, pl FORTRAN continue-ja)
```

Példa c89-el nem forduló, de c99-el forduló programra:

```
int main()
{
    for (int i = 0; i < 10; i++)
    {
        //do something
    }

    return 0;
}
```

Ennél a példánál 2 hibát is fogunk kapni, ha c89-el próbáljuk fordítani. Az első, hogy nem deklarálhatunk változót a ciklusban (c99-től támogatott). A másik, hogy nem használható a '//' jelölés kommenthez, mert az csak c90-től támogatott.

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 (15:01-től).

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.l

A lexer programok segítségével tudunk generálni szövegolvasó/elemző programokat. Lényegében arról van szó, hogy olyan programokat írhatunk amik képesek más szövegeket (akár programok forráskódját is) "értelmezni" és előre megadott típusú adatokat kinyerni belőle vagy megváltoztatni azt.

```
%{
#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 forráskód 3 fő részre bontható:

1. Definíciók helye. Ide olyan dolgok kerülnek (pl változók), melyek biztosan benne lesznek a forrásszövegben.
2. Szabályok helye. A példában valós számokat keresünk, tehát a szabályunk olyan számokat keress, hogy tetszőleges számjegy mely után lehet (de nem kötelezően) '.' karakterrel elválasztva további tetszőleges számú számjegy.
3. Az utolsó részben helyezkedik el a main függvény, ahova a saját utasításainkat írjuk.

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás videó: https://youtu.be/06C_PqDpD_k

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.l

```
%{
  #include <stdio.h>
```

```
#include <stdlib.h>
#include <time.h>
#include <ctype.h>

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

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

{'a', {"4", "4", "@", "/-\\"}},
{'b', {"b", "8", "|3", "|}"}},
{'c', {"c", "(", "<", "{"}},
{'d', {"d", "|)", "|]", "|}"}},
{'e', {"3", "3", "3", "3"}},
{'f', {"f", "|=", "ph", "|#"}},
{'g', {"g", "6", "[", "[+"}},
{'h', {"h", "4", "|-|", "[-]"}},
{'i', {"1", "1", "|", "!"}},
{'j', {"j", "7", "_|", "_/"}},
{'k', {"k", "|<", "1<", "|{"}},
{'l', {"l", "1", "|", "|_"}},
{'m', {"m", "44", "(V)", "|\\/|"}},
{'n', {"n", "|\\|", "/\\/", "/V"}},
{'o', {"0", "0", "()", "[]"}},
{'p', {"p", "/o", "|D", "|o"}},
{'q', {"q", "9", "O_", "(,)"}},
{'r', {"r", "12", "12", "|2"}},
{'s', {"s", "5", "$", "$"}},
{'t', {"t", "7", "7", "'|'"}},
{'u', {"u", "|_|", "(_)", "[_]"}},
{'v', {"v", "\\/", "\\/", "\\/"}},
{'w', {"w", "VV", "\\/\\/", "(/\\)"}},
{'x', {"x", "%", ")(", ")("}},
{'y', {"y", "", "", ""}},
{'z', {"z", "2", "7_", ">_"}},

{'0', {"D", "0", "D", "0"}},
{'1', {"I", "I", "L", "L"}},
{'2', {"Z", "Z", "Z", "e"}},
{'3', {"E", "E", "E", "E"}},
{'4', {"h", "h", "A", "A"}},
{'5', {"S", "S", "S", "S"}},
{'6', {"b", "b", "G", "G"}},
{'7', {"T", "T", "j", "j"}},
{'8', {"X", "X", "X", "X"}},
{'9', {"g", "g", "j", "j"}}

// https://simple.wikipedia.org/wiki/Leet
```

```
  };

%}
%%
. {

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

      if(l337d1c7[i].c == tolower(*yytext))
      {

        int r = 1+(int) (100.0*rand()/(RAND_MAX+1.0));

          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);

  }
%%
int
main()
{
  srand(time(NULL)+getpid());
  yylex();
  return 0;
}
```

A program lényege, hogy minden karakterhez társítunk négy darab hozzá hasonló karaktert/karaktersorozatot, majd egy bekért szövegből véletlenszerűen kisorsolunk minden karakternek (ami megtalálható a listában) egyet a párjai közül.

A program elején van definiálva a szótárunk, mely az angol ABC betűihez és a számokhoz társít párokat.

Működés közben egész egyszerűen végig lépked a bemenet szövegén karakterről karakterre és ahol egyezik a karakter, ott lecseréli egyre. Ahhoz hogy megállapítsuk, hogy az adott karaktert kell e cserélni, a 'found'

nevű segédváltozót használjuk, mely ha '1' akkor kell cserélni, ha viszont kettő, akkor nem szerepel a szótárban, tehát úgy hagyjuk. Azt hogy melyikre, azt egy 1-100 között random generált szám határozza meg. Tehát százalékos esélyek alapján:

1. 90%: az első párjára cseréli.
2. 4%: a második párjára cseréli.
3. 3%: a harmadik párjára cseréli.
4. 3%: a negyedik párjára cseréli.

3.6. A források olvasása

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

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

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

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

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

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

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

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

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

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

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

DRAFT

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

Megoldás forrása:

Megoldás videó:

1. Akkor és csakis akkor kezelje a 'jelkezelo' függvény a SIGINT jelet, ha az nincs ignorálva.
2. Egy for ciklus ami a ötször hajtja végre a hozzá rendelt utasításokat. Preorder módon először az i-t növeli és csak aztán végzi el az utasításokat.
3. Egy for ciklus ami a ötször hajtja végre a hozzá rendelt utasításokat. Postorder módon először elvégzi az utasításokat és csak azután növeli az i-t.
4. Egy for ciklus ami a ötször hajtja végre a hozzá rendelt utasításokat. Viszont a tomb[] első öt értékét lecseréli az aktuális i értékre. Tehát 0,1,2,3,4.
5.
6. A standard outputra kiíratjuk az f() függvény visszatérési értékét, decimális számban.
7. A standard outputra kiíratjuk az f() függvény visszatérési értékét 'a'-ra és magát az 'a'-t is, decimális számban.
8. A standard outputra kiíratjuk az f() függvény visszatérési értékét 'a'-ra (mivel annak a memória címére mutatunk) és magát az 'a'-t is, decimális számban.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

Ha lefuttatjuk a 'tex' kódot, akkor jól látszanak az elsőrendű formuláink. Melyek a tavaly tanultak segítségével értelmezhetőek.

- A prímszámok száma végtelen.
- Az ikerprímek száma végtelen.
- A prímszámok száma véges.

DRAFT

- A prímszámok száma végtelen.

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

```
int* getNumPointer(int* numArray);

int main()
{

    //egész
    int num = 32;

    //egészre mutató mutató
    int* toNum = &num;

    //egészek tömbje
    int numArray[4] = {0,1,2,3};

    //egészek tömbjének referenciája
    int (&numArrayRef)[4] = numArray;
```

DRAFT

```
            //egészre mutató mutatók tömbje
            int* pointerArray[4];

            //egészre mutató mutatót visszaadó függvény
            int* myReactor = getNumPointer(numArray);

            //egészre mutató mutatót visszaadó függvényre mutató mutató
            int* (*pointerFUN)(int*) = getNumPointer;

            //egészet visszaadó és két egészet kapó függvényre mutató ↩
               mutatót visszaadó, egészet kapó függvény


            return 0;

        }

        int* getNumPointer(int* numArray)
        {
            return numArray;
        }
```

Mit vezetnek be a programba a következő nevek?

- `int a;`
- `int *b = &a;`
- `int &r = a;`
- `int c[5];`
- `int (&tr)[5] = c;`
- `int *d[5];`
- `int *h ();`
- `int *(*l) ();`
- `int (*v (int c)) (int a, int b)`
- `int (*(*z) (int)) (int, int);`

- `a` mint egész típusú változó.
- `a` memória címére mutató mutató.
- Egy egészre mutató mutatót `r` névvel, ami az `a` értékét mint mutatócím tartalmazza.
- Öt elemű tömb, mely egészekből áll.
- Egy 5 elemű, egészeket tartalmazó tömbre mutató mutató, mely a `c` tömbre mutat.
- Öt elemű egészekre mutató mutatókból álló tömb.
- Egésszel visszatérő, paraméter nélküli függvényre mutató mutató.

Megoldás videó:

Megoldás forrása:

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összahasonlítása azt mutatja meg, hogy miért érdemes a **typedef** használata: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr.c, bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr2.c.

```
#include <stdio.h>

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

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

int (*sumormul (int c)) (int a, int b)
{
    if (c)
        return mul;
    else
        return sum;

}

int
main ()
{

    int (*f) (int, int);
```

```
    f = sum;

    printf ("%d\n", f (2, 3));

    int (*(*g) (int)) (int, int);

    g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

```
#include <stdio.h>

typedef int (*F) (int, int);
typedef int (*(*G) (int)) (int, int);

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

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

F sumormul (int c)
{
    if (c)
        return mul;
    else
        return sum;
}

int
main ()
{

    F f = sum;

    printf ("%d\n", f (2, 3));

    G g = sumormul;
```

```
    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

DRAFT

4. fejezet

Ave, Caesar!

4.1. double ** háromszögmátrix

Írj egy olyan `malloc` és `free` párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

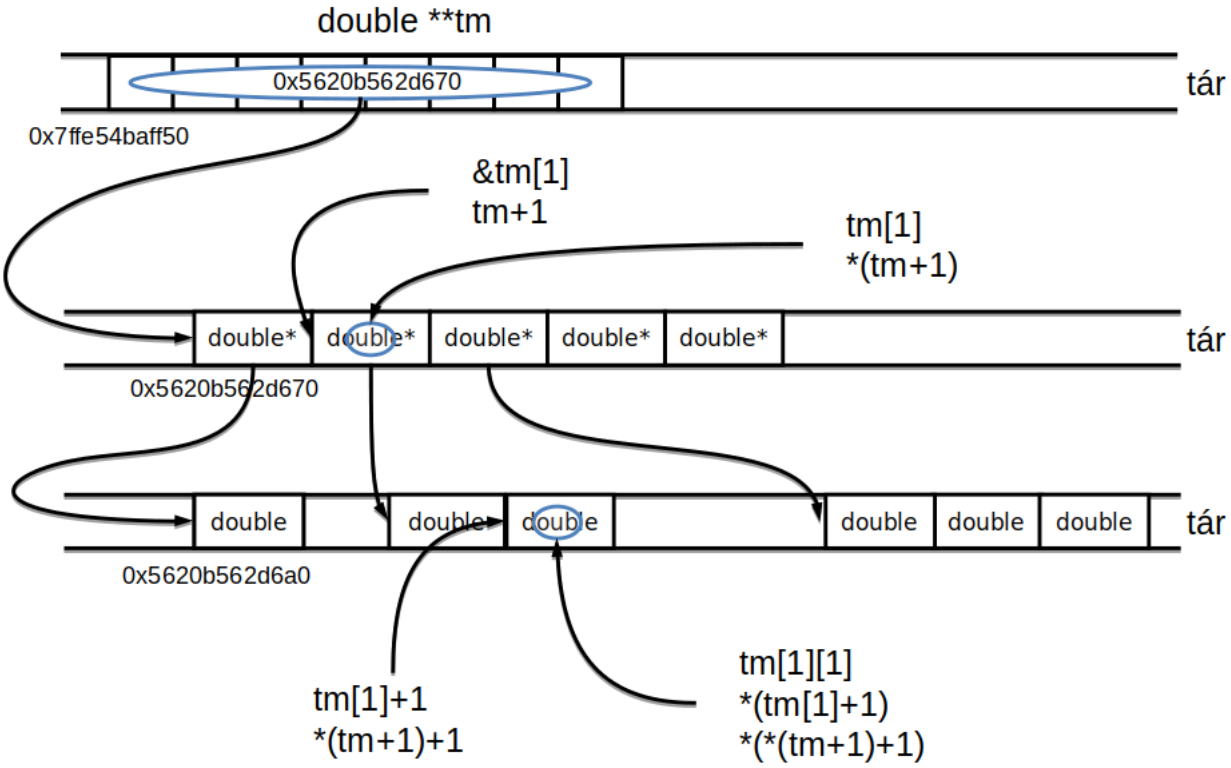
Megoldás videó: https://youtu.be/1MRTuKwRsB0, https://youtu.be/RKbX5-EWpzA.

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Caesar/tm.c

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

int
main ()
{
    int nr = 5;
    double **tm;

    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }

    for (int i = 0; i < nr; ++i)
    {
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL ↩
            )
        {
            return -1;
        }

    }

    for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
```

```
            tm[i][j] = i * (i + 1) / 2 + j;

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

    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
    *(tm[3] + 2) = 44.0;
    *(*(tm + 3) + 3) = 45.0;

    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;
}
```

DR



4.1. ábra. A double ** háromszögmátrix a memóriában

Elsőnek meghatározunk egy egész változót, mely a sorok számát fogja megadni, jelen esetben ez 5. Majd egy double típusú dupla mutatót, melyet a mátrixunk fog használni. Azért kell hogy dupla legyen, mert a mátrix is lényegében egy két dimenziós tömb, így nem elég egy mutató (mivel meg kell határozni a sort és az oszlopot is).

Ezután a malloc segítségével helyet foglalunk a tárban és ha eközben hiba lép fel, akkor visszatérünk -1 értékkel, azaz, hogy hiba történt a foglalás során, így az meghiusult.

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

int
main ()
{
    int nr = 5;
    double **tm;

    if ((tm = (double **) malloc (nr * sizeof (double *))) == NULL)
    {
        return -1;
    }

    for (int i = 0; i < nr; ++i)
    {
```

```
        if ((tm[i] = (double *) malloc ((i + 1) * sizeof (double))) == NULL ↩
            )
        {
            return -1;
        }
    }
```

Majd egy dupla for ciklus segítségével feltöltjük elemekkel az alsó háromszög mátrixunkat. Megjegyzendő, hogy a kimeneten az első sor csupa nullából áll, de ez azért van mert az informatikában a számozás általában a nullától kezdődik és nem az egytől. Ezután a következő dupla for pedig kiíratja a mátrixunkat a standard kimenetre.

```
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");
    }
```

Ezt követi egy szemléletes példa a pointerekről, hogy lássuk, hogy hány féle képpen meghivatkozhatunk egy memória címet. Itt a negyedik sor négy szereplőjét cseréljük ki 42-45ig, majd a csere után újra kirajzoljuk a mátrixunkat.

```
tm[3][0] = 42.0;
(*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
*(tm[3] + 2) = 44.0;
*(*(tm + 3) + 3) = 45.0;

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

Miután mindezzel végeztünk, felszabadítjuk a lefoglalt helyet. Fontos lehet megjegyezni, hogy a lefoglalással ellentétben, a felszabadítás bentről kifele történik, logikus módon.

```
for (int i = 0; i < nr; ++i)
    free (tm[i]);

free (tm);
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Az alábbi program Bátfai Norbert munkája:

```
#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);

    }
}
```

Megoldás videó:

Megoldás forrása:

Ez a program alapvetőleg egyszerű mechanikán alapszik. Adott egy kulcs és egy szöveg. Azt akarjuk hogy a szöveg olvashatatlan legyen annak aki nem ismeri a kulcsot. Ezt úgy valósítjuk meg, hogy beolvassuk a kulcsot, majd a szöveget. A kulcs lesz a kulcs, a szöveg meg kerül a buffer-be.

```
#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);

}
```

Ezután egy while ciklus végig megy a buffer-en, benne pedig egy for ciklus a ténylegesen olvasott bájtokon. Itt egyszerűen végig lépkedünk karakterről karakterre és az adott karakter bitjeit XOR (kizáró vagy) művelet segítségével összemossuk a kulcs aktuális karakterével, melyet utánna újra meghatározunk, méghozzá az aktuális indexet léptetjük eggyel, majd maradékossan osztjuk a kulcs méretével és a maradék képzi az új kulcs indexet. Ezután pedig lépünk a következő karakterre.

```
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);

}
```

Végül pedig kiíratjuk az így kapott kaotikus bitkupacot, melyben a felismerhető karakter is ritka, nem hogy az értelmes szöveg.

```
                write (1, buffer, olvasott_bajtok);
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Tutoriáltam volt Tutor Tünde (tutorception).

Megoldás videó:

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

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

public class main
{
    public static void encode (String key, InputStream in, OutputStream out ↩
       ) throws java.io.IOException
    {
        byte[] kulcs = key.getBytes(); //beolvassuk a key stringer a  ↩
           kulcsba
        byte[] buffer = new byte[256]; //létrehozunk egy 256 elemű buffert  ↩
           a bemenetnek
        int kulcsIndex = 0;
        int readBytes = 0;

        while((readBytes = in.read(buffer)) != -1) //addig próbálja amíg  ↩
           érkezik bemenet
        {
            for(int i=0; i<readBytes; i++)
            {
                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]); //XOR
                kulcsIndex = (kulcsIndex + 1) % key.length();
            }

            out.write(inputBuffer, 0, readBytes); //kiíratjuk a kimenetre
        }
    }

    public static main (String[] args)
    {
        if(args[0] != "") //ellenőrizzük, hogy kaptunk-e egyáltalán valamit
        {
            try
            {
                encode(args[0], System.in, System.out); //
```

```
            }
            catch(java.io.IOException e) //catching errors
            {
                e.printStackTrace();
            }
        }
        else //ha nem, közöljük a user-el
        {
            System.out.println("Please provide a key!");
            System.out.println("java main <key>");
        }

    }
}
```

Őszintén szólva nem vagyok egy nagy híve a Java-nak. Nem tudom miért, de tőlem életidegen volt mindig is. Talán azért mert úgy tűnt mintha túl lenne bonyolítva minden. Egyébként nem rossz kis nyelv, legjobban nekem a C#-hoz hasonlít, bár lehet hogy inkább a C# hasonlít a Java-ra. Kicsit azért bajba voltam szintaktikailag, hogy mit hogyan lehet, de végül is lett belőle valami. Na de szedjük szét:

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

public class main
{
    public static void encode (String key, InputStream in, OutputStream out ↩
       ) throws java.io.IOException
    {
        byte[] kulcs = key.getBytes(); //beolvassuk a key stringer a  ↩
           kulcsba
        byte[] buffer = new byte[256]; //létrehozunk egy 256 elemű buffert  ↩
           a bemenetnek
        int kulcsIndex = 0;
        int readBytes = 0;

    }
}
```

A main osztályon belül van két kis csinos függvényünk, az `encode` és ugye a `main`. Az `encode` 3 paramétert kap. Egy string-et, az input- és az output csatornát. Majd létrehozunk egy byte tömböt `kulcs` néven, melybe a `getBytes` segítségével beolvassuk a `key` string-et. Majd létrehozzunk egy 256 elemű buffert és két integert.

```
while((readBytes = in.read(buffer)) != -1) //addig próbálja amíg érkezik  ↩
   bemenet
    {
        for(int i=0; i<readBytes; i++)
        {
            buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]); //XOR
            kulcsIndex = (kulcsIndex + 1) % key.length();
        }
```

```
        out.write(inputBuffer, 0, readBytes); //kiíratjuk a kimenetre
    }
```

Ezután egy while ciklusban addig olvassuk a bájtokat amíg van bemenet, majd ezeken egyesével végig haladunk a for ciklussal. A buffer minden egyes ellemét XOR művelet (logikai kizáró vagy) segítségével keverjük a kulcs egyik bájtjával, melyet az aktuális kulcsindex határoz me. Majd új kulcsindexet határozunk meg, úgy hogy hozzáadunk egyet majd maradékosan osztjuk a kulcs hosszával. Az így kapott maradék lesz az új kulcsindex. Végül pedig egyszerűen kiíratjuk az így kapott katyvaszt.

```
public static main (String[] args)
{
    if(args[0] != "") //ellenőrizzük, hogy kaptunk-e egyáltalán valamit
    {
        try
        {
            encode(args[0], System.in, System.out); //
        }
        catch(java.io.IOException e) //catching errors
        {
            e.printStackTrace();
        }
    }
    else //ha nem, közöljük a user-el
    {
        System.out.println("Please provide a key!");
        System.out.println("java main <key>");
    }

}
```

Az egész persze mit sem érne a `main` nélkül. Elsőként ellenőrizzük, hogy egyáltalán kapunk-e valamit. Ha nem akkor közöljük a kedves felhasználóval, hogy egy kis proaktivitást is várnánk tőle egy kulcs formájában. Ha kaptunk bemenetet, akkor meghívjuk az `encode`-t és addig futtatjuk, míg kell vagy míg hibába nem ütközünk. Ha az utóbbi történik meg, akkor elkapjuk a hibát és kiíratjuk, hogy a felhasználó orvosolni tudja, ha tudja.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Ez a program egy klasszikus brute force törést mutat be. Az elv alapvetőleg egyszerű. Egy megadott szótár alapján végig próbálja az összes lehetséges kombinációt, míg meg nem találja azt amelyik nyitja. Ezzel meg lehet törni bármilyen jelszót vagy kulcsot, viszont van egy jelentősen problémás változó. Az idő. Ugyanis ez a folyamat rettentő számításkapacítás igényes. Gondoljunk csak bele, addig nincs baj míg mondjuk egy

pin kódot akarunk törni. 4 számjegy, a szótárunk 10 elemű (0-9), azaz a lehetséges kombinációk száma 10.000. Másodpercek alatt megvan. Viszont mi a helyzet egy erős jelszóval? Legalább 8 karakter, azaz ha még mindig csak számokat keresünk akkor is már 100.000.000 kombináció. Ez már most sokkal több, de akkor jön a gubanc, mikor bevezetjük, hogy nem csak számokat tartalmazhat, hanem az angol ABC betűjeit. Máris plusz 26 karakter, azaz 36 elemű a szótár. Így már 2.821.109.907.456-ra ugrott a lehetséges esetek száma. Akkor még a nagybetűk megint 26 karakter és még nem is beszéltünk a speciális karakterekről. Szerintem már érezhető, hogy egy bivaly PC-vel is inkább évtizedekről lenne szó, mintsem évekről...

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 4
#define _GNU_SOURCE

#include <stdio.h>
#include <unistd.h>
#include <string.h>

double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
  int sz = 0;
  for (int i = 0; i < titkos_meret; ++i)
    if (titkos[i] == ' ')
      ++sz;

  return (double) titkos_meret / sz;
}

int
tiszta_lehet (const char *titkos, int titkos_meret)
{
  // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
  // illetve az átlagos szóhossz vizsgálatával csökkentjük a
  // potenciális töréseket

  double szohossz = atlagos_szohossz (titkos, titkos_meret);

  return szohossz > 6.0 && szohossz < 9.0
    && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
    && strcasestr (titkos, "az") && strcasestr (titkos, "ha");

}

void
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{

  int kulcs_index = 0;
```

```
  for (int i = 0; i < titkos_meret; ++i)
    {

      titkos[i] = titkos[i] ^ kulcs[kulcs_index];
      kulcs_index = (kulcs_index + 1) % kulcs_meret;

    }

}

int
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
      int titkos_meret)
{

  exor (kulcs, kulcs_meret, titkos, titkos_meret);

  return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

  char kulcs[KULCS_MERET];
  char titkos[MAX_TITKOS];
  char betuk[5] = {'c','i','a','l','\0'};
  char *p = titkos;
  int olvasott_bajtok;

  // titkos fajt berantasa
  while ((olvasott_bajtok =
    read (0, (void *) p,
    (p - titkos + OLVASAS_BUFFER <
     MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
    p += olvasott_bajtok;

  // maradek hely nullazasa a titkos bufferben
  for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';

  // osszes kulcs eloallitasa
  for (int ii = 0; ii <= 4; ++ii)
    for (int ji = 0; ji <= 4; ++ji)
      for (int ki = 0; ki <= 4; ++ki)
    for (int pi = 0; pi <= 4; ++pi)
      {
        kulcs[0] = betuk[ii];
        kulcs[1] = betuk[ji];
```

```
        kulcs[2] = betuk[ki];
            kulcs[3] = betuk[pi];

        if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
          printf
            ("Kulcs: [%c%c%c%c]\nTiszta szoveg: [%s]\n",
            ii, ji, ki, pi, titkos);

        // ujra EXOR-ozunk, igy nem kell egy masodik buffer
        exor (kulcs, KULCS_MERET, titkos, p - titkos);
      }

  return 0;
}
```

A program Bátfai Norbert programjának egy kicsit egyszerűsített verziója, probléma specifikussági okok miatt. Az eredeti változat 8 karakter hosszú kulcsot tudott törni, mely csak számokból állhatott. Az én esetemben az ABC négy karaketeres volt, mivel a lehetséges karakterek ismertek voltak (név szerint: c, i, a, l), melyeket egy char tömbben tárolok. A másik különbbség, hogy az általam tört kulcs csupán 4 karakterből állt, tehát négy darabb for ciklussal rövidebb lett és a `KULCS_MERET`-et is le kellett csökkenteni. De nézzük meg kicsit darabokban is (most a könyvtárak meghívását, meg ilyeneket kihagynék):

```
double
atlagos_szohossz (const char *titkos, int titkos_meret)
{
  int sz = 0;
  for (int i = 0; i < titkos_meret; ++i)
    if (titkos[i] == ' ')
      ++sz;

  return (double) titkos_meret / sz;
}
```

Az első függvényünk az `atlagos_szohossz` mely ahogy a neve is jól mutatja, az átlagos szóhosszt kívánja meghatározni a számunkra. Ezt gyermeki könnyedséggel meg is tehetjük, miután paraméterként bekértük a titkos szöveg méretét és az üzenetet magát. Elsőként meg kell határoznunk hogy hány darab szó van benne, ehhez használjuk fel a mondatok azon tulajdonságát, hogy a szavakat elválasztjuk egymástól. Tehát csak végig kell lépkednünk a szöveg összes karakterén és megszámolni a szóközöket (egy esetleges elírás, értem úgy mint dupla szóköz, rendesen meg tudja nehezíteni a dolgunkat ebből a szempontból). Na de ha ennek a ciklusnak vége, akkor már rendelkezünk a szavak számával ami az `sz`-ben tárolunk. Az átlag megkapásához egyszerűen csak el kell osztani a szöveg méretét a szavak számával és már meg is vagyunk.

```
int
tiszta_lehet (const char *titkos, int titkos_meret)
{
```

```
  // a tiszta szoveg valszeg tartalmazza a gyakori magyar szavakat
  // illetve az átlagos szóhossz vizsgálatával csökkentjük a
  // potenciális töréseket

  double szohossz = atlagos_szohossz (titkos, titkos_meret);

  return szohossz > 6.0 && szohossz < 9.0
    && strcasestr (titkos, "hogy") && strcasestr (titkos, "nem")
    && strcasestr (titkos, "az") && strcasestr (titkos, "ha");

}
```

A `tiszta_lehet` egy létfontosságú része a kódunknak, ugyanis honnan tudná a program hogy a próbált kulcs értelmes szöveget eredményezett-e? Hát ha talál benne értelmes szavakat. De nem mindegy hogy milyen szót keresünk, ugyanis például a hiperhajtómű vagy a nukleáris tengeralattjáró aránylag ritkán fordúl elő az átlagszövegekben (kivéve itt :D). Viszont ha visszaolvassuk csak ezt a bekezdést, akkor találunk benne olyat, hogy "hogy", "nem", "az" és "ha". Ha nem is az összes bekezdésben, de sokban megtalálható mind. Pont ezért keressük benne ezeket a szavakat (az "a"-t azért nem keressük, mert az csak egy karakteres, tehát könnyen megjelenhet véletlenül is). Itt még felhasználjuk az átlag szóhosszt is, mely a magyar nyelvben 7-8 betű. Ezért folt szükség az előző függvényünkre is. Tehát a program csak akkor fogja sikeresnek nyílvánítani a törést, ha az átlag szóhossz 7-8 karakter és tartalmazza ezt a 4 gyakori magyar szót is. Itt megjegyzem hogy sok szöveget nem tört a program, pont azért mert nem volt meg benne az összes szükséges szó, de ezt most félretehetjük.

```
void
exor (const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
{

  int kulcs_index = 0;

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

      titkos[i] = titkos[i] ^ kulcs[kulcs_index];
      kulcs_index = (kulcs_index + 1) % kulcs_meret;

    }

}
```

Ez a függvény nagyon hasonló mint az amelyikkel titkosítottunk. Az elgondolás ugyan az, a titkos szöveget összekombózzuk az aktuális kulccsal amit majd a `tiszta_lehet` fog ellenőrizni a sikeresség szempontjából. Erre most Nem térnék ki túlságosan, kerülném a felesleges önismétlést (lsd. C EXOR titkosító, while ciklus bekezdés).

```
int
```

```
exor_tores (const char kulcs[], int kulcs_meret, char titkos[],
      int titkos_meret)
{

  exor (kulcs, kulcs_meret, titkos, titkos_meret);

  return tiszta_lehet (titkos, titkos_meret);

}
```

Az utolsó függvényünk a `main` előtt már csak elvégzi a piszkos munkát. Meghívja a `exor`-t majd azt ellenőrizteti a `tiszta_lehet`-tel. Ha az úgy véli hogy a szöveg rendben van, akkor visszatér igaz értékkel. Na de a `main`-ben ki is derül, hogy miért jó ez nekünk:

```
int
main (void)
{

    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char betuk[5] = {'c','i','a','l','\0'};
    char *p = titkos;
    int olvasott_bajtok;

    // titkos fajt berantasa
    while ((olvasott_bajtok =
        read (0, (void *) p,
        (p - titkos + OLVASAS_BUFFER <
            MAX_TITKOS) ? OLVASAS_BUFFER : titkos + MAX_TITKOS - p)))
    p += olvasott_bajtok;

    // maradek hely nullazasa a titkos bufferben
    for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
    titkos[p - titkos + i] = '\0';

    // osszes kulcs eloallitasa
    for (int ii = 0; ii <= 4; ++ii)
    for (int ji = 0; ji <= 4; ++ji)
        for (int ki = 0; ki <= 4; ++ki)
        for (int pi = 0; pi <= 4; ++pi)
            {
            kulcs[0] = betuk[ii];
            kulcs[1] = betuk[ji];
            kulcs[2] = betuk[ki];
            kulcs[3] = betuk[pi];

            if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
                printf
```

```
                    ("Kulcs: [%c%c%c%c]\nTiszta szoveg: [%s]\n",
                    ii, ji, ki, pi, titkos);

            // ujra EXOR-ozunk, igy nem kell egy masodik buffer
            exor (kulcs, KULCS_MERET, titkos, p - titkos);
            }

    return 0;
}
```

Fel kell vennünk a szükséges változókat, köztük az ABC-énket is. Egy while ciklussal beolvassuk a titkos szövegünket, majd a for ciklusban ürítjük azokat a helyeket a bufferből amiket nem használunk. Aztán jön a lényeg, a kulcs generálás. Négy darab for ciklussal végig próbáljuk az összes lehetőséget, kezdve a "cccc"-től az "llll"-ig minden kombinációt amíg meg nem találja a megfelelőt. Minden új kulccsal kipróbáljuk hogy tiszta szöveget eredményez-e. Ehhez if-en belül hívjuk meg az `exor_tores`-t, amely ugye igaz-hamis értéket fog visszaadni. Igaz érték esetén végeztünk, egyszerűen csak kiíratjuk a kulcsot és a szöveget. Viszont ha hamis akkor tovább kell próbálkoznunk, de egy apró probléma adódik itt, méghozzá az hogy a titkos szöveget már össze exor-oztuk a rossz kulccsal. De semmi ok a pánikra, mert mint ahogy ott is említettem, ez az eljárás ugyan az mint amivel titkosítottunk. Tehát ha meghívjuk megint akkor egyszerűen visszatitkosítja a szöveget és próbálkozhatunk tovább annélkül, hogy ehhez fent kellene tartanunk egy második buffert.

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

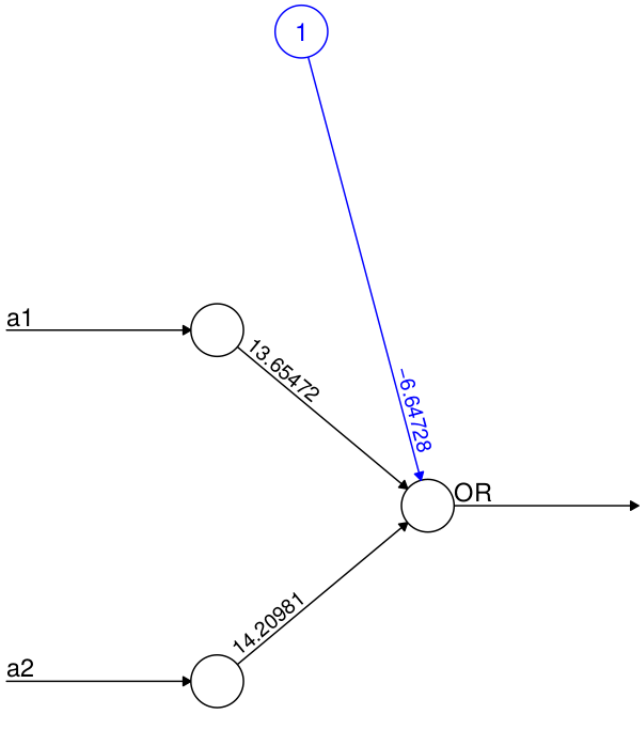
```
library(neuralnet)

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

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

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

plot(nn.or)

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


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

BAFT

```
a2    <- c(0,0,1,1)
OR    <- c(0,1,1,1)
AND   <- c(0,0,0,1)

orand.data <- data.frame(a1, a2, OR, AND)

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

plot(nn.orand)

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



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

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

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

plot(nn.exor)

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



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

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

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

plot(nn.exor)

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

Ez egy neurális háló szimulációja R nyelven. Elsőnek egy "OR" logikai kaput vizsgálunk meg. Ez ítélet logikai szempontból nem túl bonyolult, csak akkor lesz hamis, ha mindkét tagja hamis, minden egyéb esetben igaz igazságértéket kapunk. Azzal kezdjük, h ezt az egyszerű igazságtábált megtanítjuk neki. Adunk két mintasort (a1, a2) és megmondjuk, hogy ezek viszonyából milyen igazság érték származtatódik a művelet után. Ezután egy táblázatot csinálunk vele, amit aztán megettetünk egy `neuralnet()` függvényel és végül ábrázoljuk az eredményt.

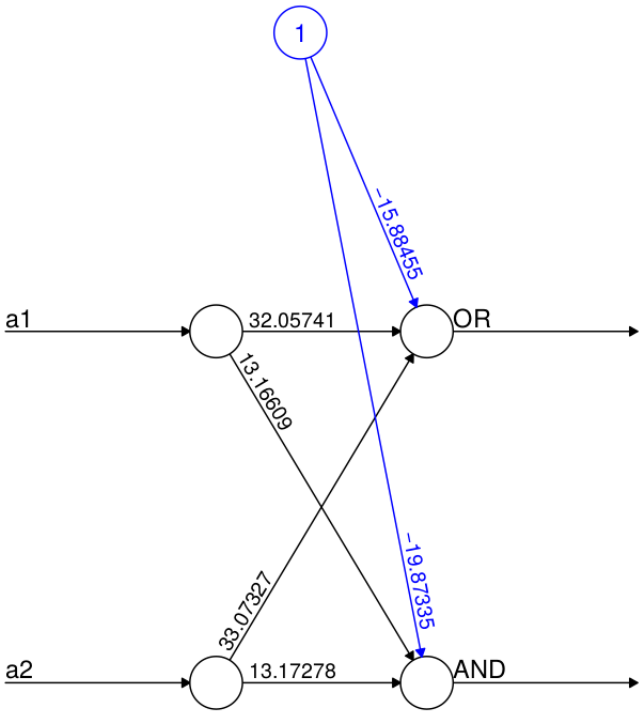
```
library(neuralnet)

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

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

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

plot(nn.or)

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



A másodikban már bonyolítunk egy kicsit és megjelenik az "AND" művelet is. Erre azért van szükség, mert a végcél egy "XOR" művelet lenne, ami egy kizáró vagy, viszont ez nem alap művelet és fel kell bontani. Viszont az "AND" még mindig nem vészes. Csak akkor igaz, ha mindkét eleme igaz. Hasonlóan járunk el

mint legutóbb, adunk mintát, 'megtanítjuk neki a műveleteket a minta alapján', abból táblázatot alkotunk, abból neurális hálót, majd végűl ábrázoljuk.

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

orand.data <- data.frame(a1, a2, OR, AND)

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

plot(nn.orand)

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

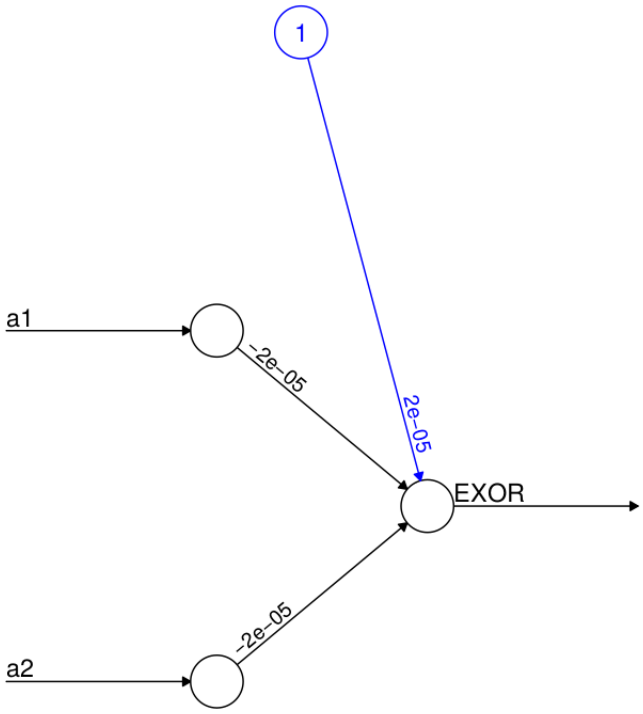


Végül eljutunk a várva várt kizáró vagyhoz. Mint azt az előbb említettem, ez már egy bonyolultabb művelet. Bár itt is megmondjuk neki, hogy mire mit várunk, valamiért a szimuláció során mégis hibába ütközünk. Az eredmények 0.5-höz közelítenek, nincs meg a szórás. Valószínűleg ha felbontjuk a műveletet és összetett

logikai probléma ként kezelnénk, akkor ez kiküszöbölhető lenne.

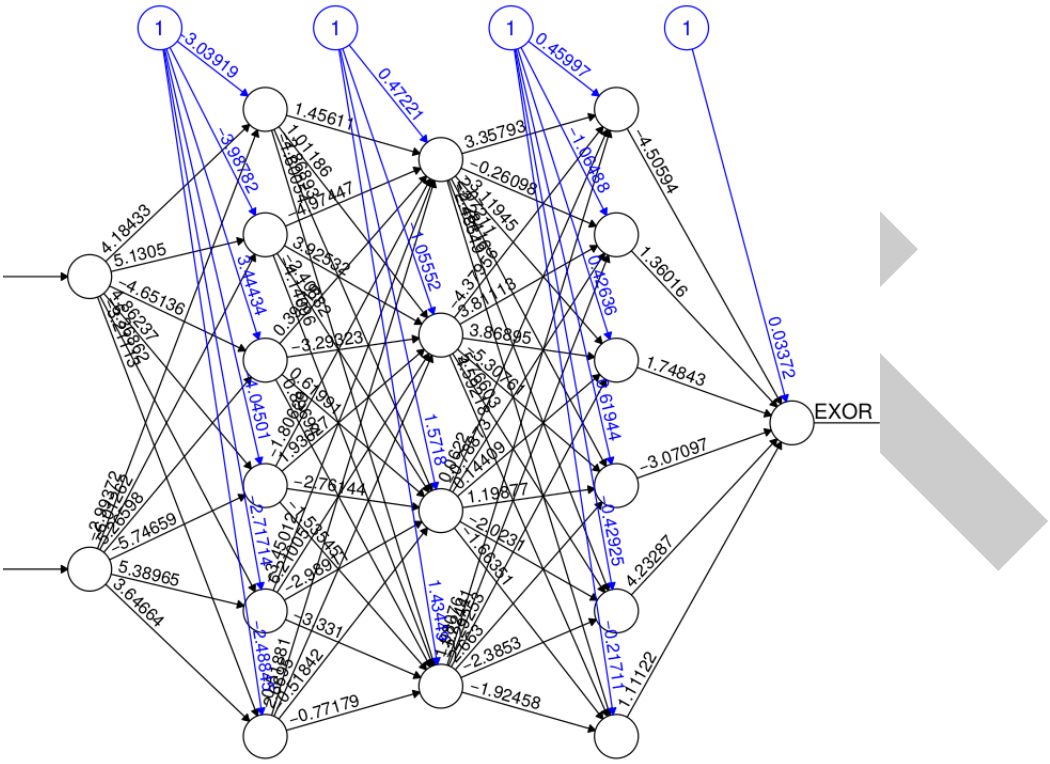
```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)

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

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

plot(nn.exor)

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



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

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

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

plot(nn.exor)

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

DRAFT



Error: 0.5 Steps: 93

DR



Error: 1e-06 Steps: 55

```
$neurons
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1


$net.result
             [,1]
[1,] 0.001295867
[2,] 0.999095698
[3,] 0.999480713
[4,] 0.999999999

dev.new(): using pdf(file="Rplots1.pdf")
$neurons
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1
```

```
$net.result
             [,1]         [,2]
[1,] 1.263070e-07 2.339449e-09
[2,] 9.999999e-01 1.220517e-03
[3,] 1.000000e+00 1.228700e-03
[4,] 1.000000e+00 9.984462e-01

dev.new(): using pdf(file="Rplots2.pdf")
$neurons
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1


$net.result
          [,1]
[1,] 0.5000062
[2,] 0.5000016
[3,] 0.5000003
[4,] 0.4999958

dev.new(): using pdf(file="Rplots3.pdf")
$neurons
$neurons[[1]]
       a1 a2
[1,] 1  0  0
[2,] 1  1  0
[3,] 1  0  1
[4,] 1  1  1

$neurons[[2]]
     [,1]        [,2]         [,3]      [,4]         [,5]         [,6]
[1,]    1 0.045686555 1.820265e-02 0.9690620 0.0172082928 0.0619692958
[2,]    1 0.758622916 7.581722e-01 0.2302292 0.6936760783 0.0001132546
[3,]    1 0.002392765 4.537795e-05 0.9998352 0.0000559165 0.9353847449
[4,]    1 0.136039995 7.615410e-03 0.9830262 0.0071801621 0.0242188339
            [,7]
[1,] 0.076671831
[2,] 0.003336195
[3,] 0.761002931
[4,] 0.113754997

$neurons[[3]]
     [,1]       [,2]       [,3]       [,4]       [,5]
[1,]    1 0.63287096 0.02913166 0.84879077 0.86566496
[2,]    1 0.03359427 0.96383097 0.01471146 0.02229534
```

```
[3,]    1 0.05988855 0.97053882 0.26776150 0.20166433
[4,]    1 0.70336901 0.02515204 0.83498484 0.86846306

$neurons[[4]]
     [,1]       [,2]       [,3]       [,4]       [,5]        [,6]          ↩
        [,7]
[1,]    1 0.98558535 0.03622016 0.03566055 0.99393396 0.004115315  ↩
   0.008706264
[2,]    1 0.02651157 0.92754571 0.98204179 0.01274759 0.993094797  ↩
   0.983011104
[3,]    1 0.04018059 0.88689361 0.97263565 0.02793090 0.982143575  ↩
   0.962935374
[4,]    1 0.98883596 0.03527228 0.02816476 0.99481645 0.003499480  ↩
   0.007311599


$net.result
             [,1]
[1,] 0.0006611321
[2,] 0.9997112758
[3,] 0.9996298113
[4,] 0.0006378277
```

4.6. Passz: Hiba-visszaterjesztéses perceptron

Ebbe a feladatba belekezdtem, de a passzolási lehetőségeimből felhasználnék egyet erre. C++

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

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

Még mielőtt nagyon belekezdünk, beszéljük át, hogy mi is az a perceptron.

A perceptron nem más mint a legegyszerűbb neurális háló. Két réteggel rendelkezik, egy bemenetivel és egy kimenetivel. Ezekben a rétegekben neuronoknk nevezett csomópontok találhatóak. A bemeneti és kimeneti réteg között úgynevezett súlyozott kontaktok helyezkednek el, melyek összekötik őket és szimulálja a köztük lévő kapcsolat erősségét.

Említettem, hogy a perceptron a legegyszerűbb neurális háló, ez azért van, mert ennek csak input és output rétege van. egy klasszikus neural network ezen kettőn kívűl még rendelkezek valamennyi "hidden" réteggel. Ezeket a rétegeken különböző logikai műveletek helyezkednek el, melyek segítségével még pontosabban meg tudjuk határozni, hogy az adott bemenet, adott szituáció esetén, milyen kimenetet eredményezzen.

Továbbiakban, itt szerepel pár érdekes videó, ahol különböző deep-learning neural network-ök, megtanulnak klasszikus játékokkal 'játszani':

Snake: https://www.youtube.com/watch?v=zIkBYwdkuTk MarI/O: https://www.youtube.com/watch?v=qv6UVC

4.7. Malmo

Megoldás videó: https://youtu.be/DX8dI04rWtk

Itt a Red Flower Hell nevű pályán barátkoztunk tovább Steve diszkrét irányításával. Ebben a feladatban az volt a cél, hogy steve képes legyen felismerni ha virágot lát és esetlegesen össze is szedni azt. Ahogy a (sajnos rossz felbontású) videónkban is látszik, ennek eleget tett. De hogyan? Alapvetőleg igen egyszerű. Steve rendelkezik egy olyan képességgel, hogy observation, azaz megfigyelés. Ezeket az adatokat egy json fájlból olvassa és elég terjedelmes mennyiségű információt tud kinyerni belőle, mint pl. a koordináták, megölt möbök száma, hotbar tartalma, körülötte lévő blokkok típusa. Szóval sok mindent. Ilyen a `LineOfSight` is amire itt nekünk szükségünk lesz. Ez egy láthatatlan sugár mely Steve fejéből kiindulva, abba a pontba mutat ahova néz és Ha ez a sugár ütközik egy objektummal akkor annak meg tudjuk határozni a nevét. Ez olyan mint a `RaycastHit` a Unity3D-ben. Egyébként ez a sugár megjeleníthető a hitboxokkal (Azok a láthatatlan keretek amikkel a sugár valóban ütközik) együtt, ha megnyomjuk az F3 + B billenytűkombinációt.

Ha már meg tudjuk határozni a objektum típusát amire nézünk, akkor már könnyű dolgunk van, mert egyszerűen csak meg kell állapítanuk, hogy az "red_flower"-e és ha igen, akkor ki kell ütni, ha kell földestől.

DRAFT

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása: `bhax/attention_raising/CUDA/mandelpngt.c++` nevű állománya.

A Mandelbrot halmaz a komplex síkon

5.1. ábra. A Mandelbrot halmaz a komplex síkon

A Mandelbrot halmazt 1980-ban találta meg Benoit Mandelbrot a komplex számsíkon. Komplex számok azok a számok, amelyek körében válaszolni lehet az olyan egyébként értelmezhetetlen kérdésekre, hogy melyik az a két szám, amelyet összeszorozva -9-et kapunk, mert ez a szám például a 3i komplex szám.

A Mandelbrot halmazt úgy láthatjuk meg, hogy a sík origója középpontú 4 oldalhosszúságú négyzetbe lefektetünk egy, mondjuk 800x800-as rácsot és kiszámoljuk, hogy a rács pontjai mely komplex számoknak felelnek meg. A rács minden pontját megvizsgáljuk a $z_{n+1}=z_n{}^2+c$, (0<=n) képlet alapján úgy, hogy a c az éppen vizsgált rácspont. A z0 az origó. Alkalmazva a képletet a

- $z_0 = 0$
- $z_1 = 0^2+c = c$
- $z_2 = c^2+c$
- $z_3 = (c^2+c)^2+c$
- $z_4 = ((c^2+c)^2+c)^2+c$
- ... s így tovább.

DRAFT

Azaz kiindulunk az origóból (z_0) és elugrunk a rács első pontjába a z_1 = c-be, aztán a c-től függően a további z-kbe. Ha ez az utazás kivezet a 2 sugarú körből, akkor azt mondjuk, hogy az a vizsgált rácspont nem a Mandelbrot halmaz eleme. Nyilván nem tudunk végtelen sok z-t megvizsgálni, ezért csak véges sok z elemet nézünk meg minden rácsponthoz. Ha eközben nem lép ki a körből, akkor feketére színezzük, hogy az a c rácspont a halmaz része. (Színes meg úgy lesz a kép, hogy változatosan színezzük, például minél későbbi z-nél lép ki a körből, annál sötétebbre).

```
#include <png++/png.hpp>
#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35
void GeneratePNG( int tomb[N][M])
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y ↩
               ], tomb[x][y]);
        }
    }
    image.write("kimenet.png");
}
struct Komplex
{
    double re, im;
};
int main()
{
    int tomb[N][M];
    int i, j, k;
    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;
    struct Komplex C, Z, Zuj;
    int iteracio;
    for (i = 0; i < M; i++)
    {
        for (j = 0; j < N; j++)
        {
            C.re = MINX + j * dx;
            C.im = MAXY - i * dy;
            Z.re = 0;
            Z.im = 0;
            iteracio = 0;
            while(Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < ↩
```

```
                        255)
                    {
                        Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
                        Zuj.im = 2 * Z.re * Z.im + C.im;
                        Z.re = Zuj.re;
                        Z.im = Zuj.im;
                    }
                    tomb[i][j] = 256 - iteracio;
                }
            }
            GeneratePNG(tomb);
            return 0;
        }
```

A Mandelbrot halmaz illeszkedik a $f_c(z) = z^2 + c$ függvény képére, és korlátos, mivel nullától iterálva nem divergál $f_c(0), f_c(f_c(0)), \ldots$ abszolútértékben.

A program elején található egy GeneratePNG() függvény, mely a kiszámolt mátrix elemeiből a libpng könyvtár segítségével felépít egy png képet. Ezt úgy teszi meg, hogy a main-ben már társítottunk a mátrix elemekhez egy értéket, mely alapján eldönti, hogy az adott elemhez tartozó pixel milyen színű lesz (hiszen a képek is csak pixel mátrixok).

Bár ez egy C++ program, hogy megjeleníthető legyen a halmaz kép formájában, de alapvetőleg C alapú a kód, így a komplex számokat nem olyan triviális meghivatkozni (C++ban a complex utasítással ezt könnyen megtehetjük, de erről a következő feladat fog szólni). Mivel egy komplex szám a legtöbb számmal ellentétben két értékkel is rendelkezik (valós és imaginárius egység), ezért egy klasszikus számtípus nem elég. A megoldás egy struktúra létrehozásában van. Ezért elkészítjük a Komplex struktúrát, mely két double értékkel fog rendelkezni.

Végül egy dupla for ciklus segítségével végig rohanunk a mátrix elemein és egyesével megvizsgájuk, hogy mennyi próbálkozás után tud kilépni a halmazból (ha ki tud). Ezután 256-ból kivonjuk a próbálkozások számát. Ez fogja adni a képünknek a színárnyalatát. Ha azonnal kilép, akkor nem része a halmaznak, és teljesen fehér lesz. Minél tovább marad bent, annál sötétebb árnyalatott vesz fel. végül, ha 256 próbálkozás alatt, sem sikerül neki, az azt jelenti, hogy része a Mandelbrot halmaznak, így teljesen fekete értéket kap.

A futtatáshoz szükséges makefile tartalma:

```
all: mandelbrot clean
mandelbrot.o: mandelbrot.cpp
    @g++ -c mandelbrot.cpp `libpng-config --cflags`
mandelbrot: mandelbrot.o
    @g++ -o mandelbrot mandelbrot.o `libpng-config -- ↩
       ldflags`
clean:
    @rm -rf *.o
    @./mandelbrot
    @rm -rf mandelbrot
```

DRAFT

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

Írj olyan C++ programot, amely kiszámolja a Mandelbrot halmazt!

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

Megoldás forrása:

A Mandelbrot halmaz pontban vázolt ismert algoritmust valósítja meg a repó `bhax/attention_raising/Mandelbrot/3.1.2.cpp` nevű állománya.

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040  ↩
   -0.019473810573093663922605855987058021128 18  ↩
   -0.019473810572541341845642648422654019668 7  ↩
   0.798505756933826860155534177465597167611 1  ↩
   0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913 1  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327099751215428 1  ↩
   0.213538705180497528912653137922461610287 4
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FORA program elején található egy  ↩
   GeneratePNG() függvény, mely a kiszámolt mátrix elemeiből a libpng
                könyvtár segítségével felépít egy png képet. Ezt úgy teszi  ↩
                   meg, hogy a main-ben már társítottunk a mátrix elemekhez
                egy értéket, mely alapján eldönti, hogy az adott elemhez  ↩
                   tartozó pixel milyen színű lesz (hiszen a képek is csak  ↩
                   pixel
                mátrixok). y of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

```
#include <iostream>
#include "png++/png.hpp"
#include <complex>

int
main ( int argc, char *argv[] )
{

  int szelesseg = 1920;
  int magassag = 1080;
  int iteraciosHatar = 255;
  double a = -1.9;
  double b = 0.7;
  double c = -1.3;
  double d = 1.3;

  if ( argc == 9 )
    {
      szelesseg = atoi ( argv[2] );
      magassag =  atoi ( argv[3] );
      iteraciosHatar =  atoi ( argv[4] );
      a = atof ( argv[5] );
      b = atof ( argv[6] );
      c = atof ( argv[7] );
      d = atof ( argv[8] );
    }
  else
    {
      std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c d ↩
         " << std::endl;
      return -1;
    }

  png::image < png::rgb_pixel > kep ( szelesseg, magassag );

  double dx = ( b - a ) / szelesseg;
  double dy = ( d - c ) / magassag;
  double reC, imC, reZ, imZ;
  int iteracio = 0;

  std::cout << "Szamitas\n";

  // j megy a sorokon
  for ( int j = 0; j < magassag; ++j )
    {
      // k megy az oszlopokon

      for ( int k = 0; k < szelesseg; ++k )
```

```
        {

          // c = (reC, imC) a halo racspontjainak
          // megfelelo komplex szam

          reC = a + k * dx;
          imC = d - j * dy;
          std::complex<double> c ( reC, imC );

          std::complex<double> z_n ( 0, 0 );
          iteracio = 0;

          while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
            {
              z_n = z_n * z_n + c;

              ++iteracio;
            }

          kep.set_pixel ( k, j,
                          png::rgb_pixel ( iteracio%255, (iteracio*iteracio ↩
                             )%255, 0 ) );
        }

      int szazalek = ( double ) j / ( double ) magassag * 100.0;
      std::cout << "\r" << szazalek << "%" << std::flush;
    }

  kep.write ( argv[1] );
  std::cout << "\r" << argv[1] << " mentve." << std::endl;

}
```

Ennek a programnak a működése nagyon hasonló az előzőhöz, így pár részlet fölött elsiklanék. Viszont ami kiemelendő, hogy itt már ki is van használva a C++ egyes előnyei a C-vel szemben. Első sorban, hogy itt már meghívásra került a Complex library, mely segítségével egyszerűbben hozhatunk létre komplex számokat, anélkül, hogy ehhez struktúrát kéne használnuk. A másik, hogy az abs() függvény segítségével a négyzetre emelést is egyszerűbb megtenni, meg a végeredmény is könnyebben érthető, mint hogy változók vannak megszorozva önmagukkal.

5.3. Biomorfok

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

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

A biomorfokra (a Julia halmazokat rajzoló bug-os programjával) rátaláló Clifford Pickover azt hitte természeti törvényre bukkant: https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--

2315_Biomorphs_via_modified_iterations.pdf (lásd a 2307. oldal aljától).

A különbség a Mandelbrot halmaz és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

Ezeket kiegészíteném azzokkal az érdekességekkel, hogy Julia halmazból végtelen sok van, melyeket magába foglal a Mandelbrot halmaz, melyből viszont csak egy létezik. Továbbá fontos lehet megjegyezni, hogy a biomorfok speciális Pickover szálak, melyek hasonlítanak biológiai alakzatokra, például sejtekre. Innen is ered a biomorf elnevezés.

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
  {
    for ( int k = 0; k < szelesseg; ++k )
      {

        // c = (reC, imC) a halo racspontjainak
        // megfelelo komplex szam

        reC = a + k * dx;
        imC = d - j * dy;
        std::complex<double> c ( reC, imC );

        std::complex<double> z_n ( 0, 0 );
        iteracio = 0;

        while ( std::abs ( z_n ) < 4 && iteracio < iteraciosHatar )
          {
            z_n = z_n * z_n + c;

            ++iteracio;
          }
```

Ezzel szemben a Julia halmazos csipetben a cc nem változik, hanem minden vizsgált z rácspontra ugyanaz.

```
// j megy a sorokon
for ( int j = 0; j < magassag; ++j )
{
    // k megy az oszlopokon
    for ( int k = 0; k < szelesseg; ++k )
    {
        double reZ = a + k * dx;
        double imZ = d - j * dy;
        std::complex<double> z_n ( reZ, imZ );

        int iteracio = 0;
        for (int i=0; i < iteraciosHatar; ++i)
        {
            z_n = std::pow(z_n, 3) + cc;
            if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
```

```
                {
                    iteracio = i;
                    break;
                }
            }
```

A bimorfos algoritmus pontos megismeréséhez ezt a cikket javasoljuk: https://www.emis.de/journals/-TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf. Az is jó gyakorlat, ha magából ebből a cikkből from scratch kódoljuk be a sajátunkat, de mi a királyi úton járva a korábbi Mandelbrot halmazt kiszámoló forrásunkat módosítjuk. Viszont a program változóinak elnevezését összhangba hozzuk a közlemény jelöléseivel:

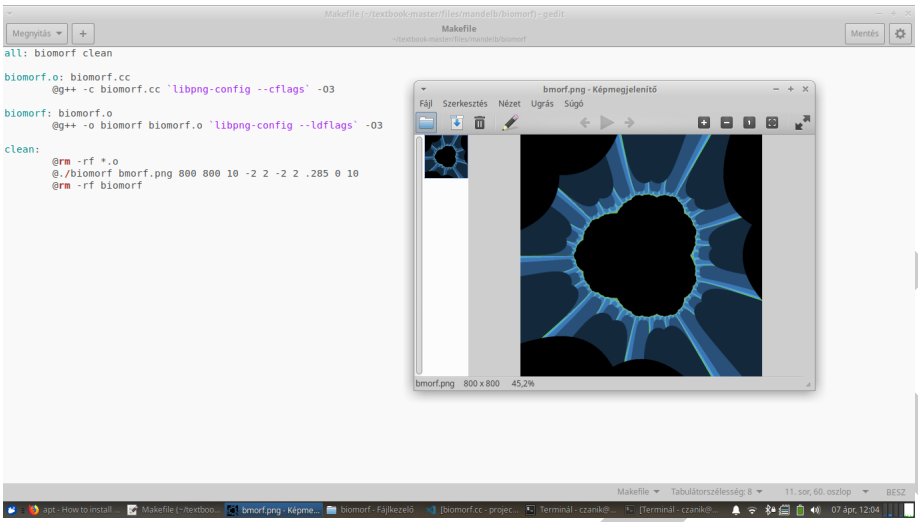
```
// Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↩
   Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

#include <iostream>
#include "png++/png.hpp"
#include <complex>
```

```
int
main ( int argc, char *argv[] )
{

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

    if ( argc == 12 )
    {
        szelesseg = atoi ( argv[2] );
        magassag =  atoi ( argv[3] );
        iteraciosHatar =  atoi ( argv[4] );
        xmin = atof ( argv[5] );
        xmax = atof ( argv[6] );
        ymin = atof ( argv[7] );
        ymax = atof ( argv[8] );
        reC = atof ( argv[9] );
        imC = atof ( argv[10] );
        R = atof ( argv[11] );

    }
    else
    {
        std::cout << "Hasznalat: ./3.1.2 fajlnev szelesseg magassag n a b c ↩
            d reC imC R" << std::endl;
        return -1;
    }

    png::image < png::rgb_pixel > kep ( szelesseg, magassag );

    double dx = ( xmax - xmin ) / szelesseg;
    double dy = ( ymax - ymin ) / magassag;

    std::complex<double> cc ( reC, imC );

    std::cout << "Szamitas\n";

    // j megy a sorokon
    for ( int y = 0; y < magassag; ++y )
    {
        // k megy az oszlopokon
```

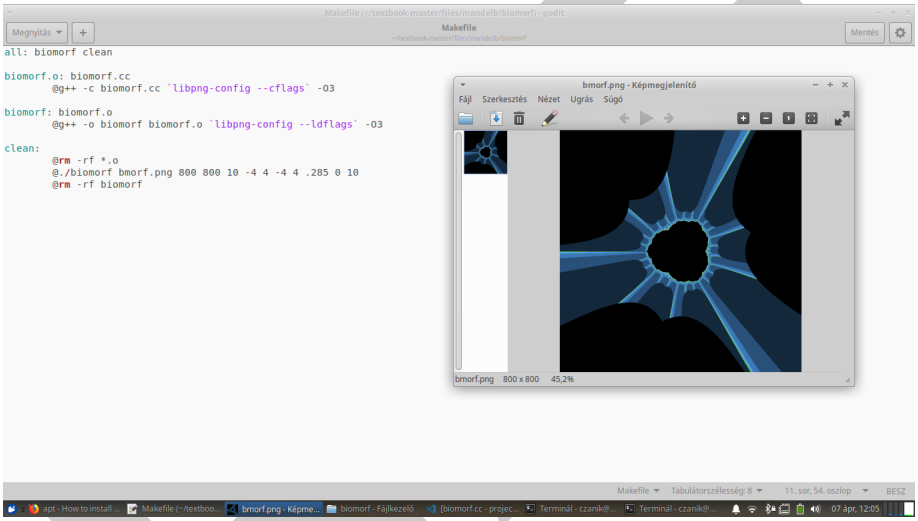
```
        for ( int x = 0; x < szelesseg; ++x )
        {

            double reZ = xmin + x * dx;
            double imZ = ymax - y * dy;
            std::complex<double> z_n ( reZ, imZ );

            int iteracio = 0;
            for (int i=0; i < iteraciosHatar; ++i)
            {

                z_n = std::pow(z_n, 3) + cc;
                //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
                if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                {
                    iteracio = i;
                    break;
                }
            }

            kep.set_pixel ( x, y,
                            png::rgb_pixel ( (iteracio*20)%255, (iteracio ↩
                               *40)%255, (iteracio*60)%255 ));
        }

        int szazalek = ( double ) y / ( double ) magassag * 100.0;
        std::cout << "\r" << szazalek << "%" << std::flush;
    }

    kep.write ( argv[1] );
    std::cout << "\r" << argv[1] << " mentve." << std::endl;

}
```

Itt is azt figyeljük, hogy mikor hagyja el a halmazt, de itt már kicsit színesítünk a dolgokon.

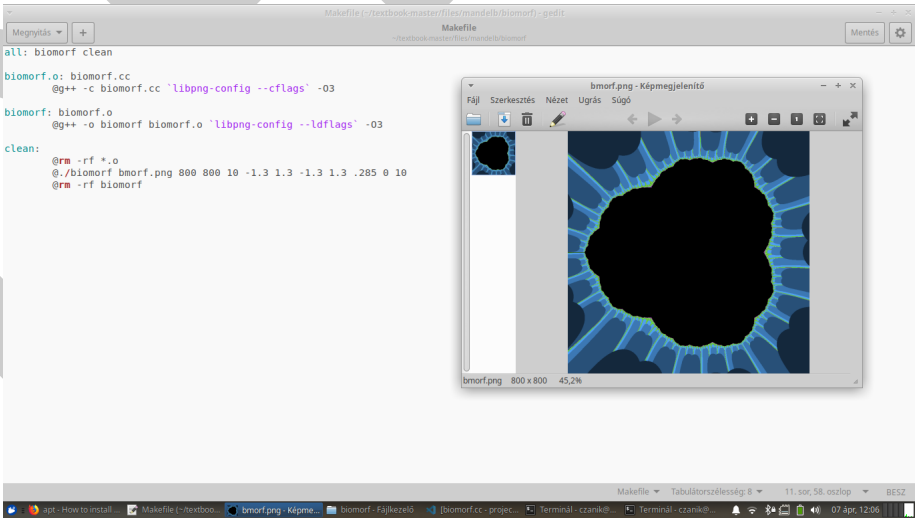
Futtatás az eredeti értékekkel:



Futtatás módosított értékekkel (1):



Futtatás módosított értékekkel (2):



5.4. A Mandelbrot halmaz CUDA megvalósítása

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

Megoldás forrása: `bhax/attention_raising/CUDA/mandelpngc_60x60_100.cu` nevű állománya.

Ez a feladat azt hivatott szemléltetni, hogy milyen előnyei lehetnek az Nvidia által fejlesztett cuda drivernek a grafikai megjelenítésben. A technológiát nagyon sokrétűen alkalmazzák, kezdve a 3D grafikától (magam is használom a rendereltetéshez, mert jóval gyorsabban számól, mint a CPU) egészen a játékokig.

Legnagyobb előnye a klasszikus OpenGL-el szemben, hogy nem a processzor használja a számításhoz, hanem a videókártyát (fontos megjegyezni, hogy ez csak az nvidia viseókártyákra vonatkozik, az amd-sek itt hátrányban vannak) , azon belül is az úgynevezett cuda magokat. Típustól függően eltérő számú mag található a VGA-ban, de minden esetben több százról van szó.

Az alábbi példában kiosztjuk a generálni kívánt kép (vagy hívhatjuk renderelésnek is) pixeleit egy-egy cuda magnak és meglátjuk mi lesz.

Eddig ha le akartunk renderelni egy Mandelbrot halmazt, az hosszú másodpercekig is eltarthat (nyílván ez nem olyan vészes, de egy komolyabb számítás esettén a szükséges idő is arányosan hatványozódik), míg a cuda render esetén ez a másodperc tört része csupán. De hogy történt ez?

Ahogy a római mondás is tartja: Divide et Impera (oszd meg és uralkodj). Ahelyett, hogy ezt a bonyolult feladatot pár darab processzor magra sózzuk (jelen esetben 4 magról van szó), inkább kiszervezzük párszáz cuda magnak. Szerintem nem kell sokat magyaráznom, hogy mi a különbség. Közösen pillanatok alatt végeznek a számítással és még csak kicsit se terhelik le a rendszer. De ne csak a levegőbe beszéljünk, jöjjenek a számok:

```
$ make
35
0.360183 sec
```

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

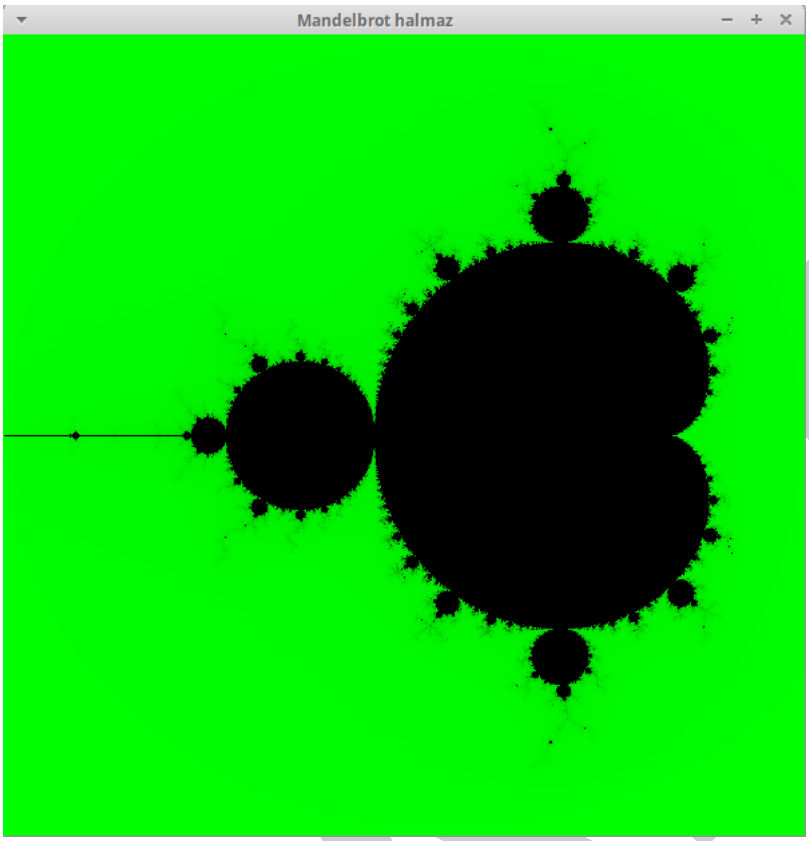
Megoldás videó: Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a_mandelbrot_halmazzal.

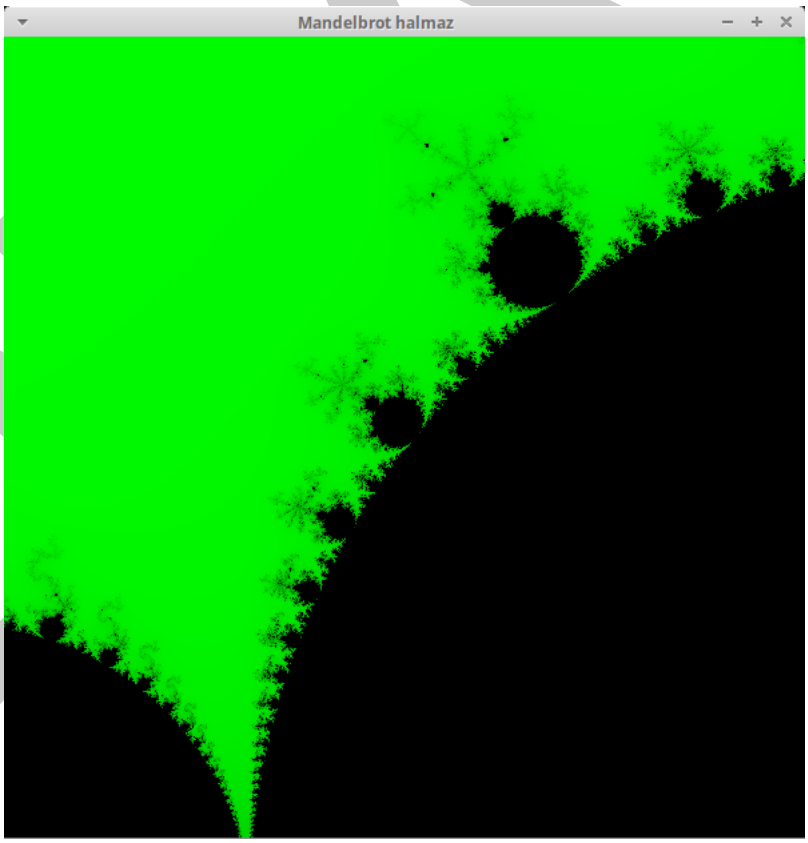
Megoldás forrása:

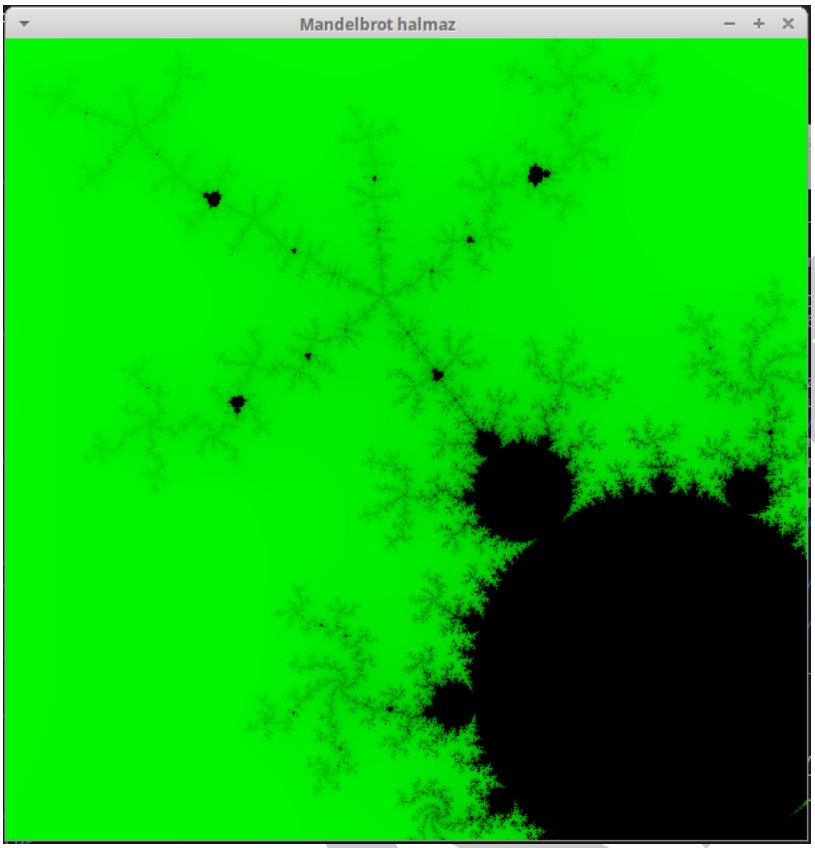
Ebben a feladatban az volt a lényeg, hogy a korábban használt Mandelbrot programunknak grafikus felhasználói felületetet (Graphic User Interface - GUI) hozzunk létre. Ehhez segítségűl hívjuk a QT program könyvtárait és objektum orientáltá alakítjuk a programunkat.

Létrehoztunk egy frakablak nevű programot, mely segítségével a terminál helyet már rendesen ablakban futtathtó a programunk, melyben megjeleníthető a mandelbrot.

DRAFT


Mandelbrot halmaz


Mandelbrot halmaz



5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás videó: https://youtu.be/Ui3B6IJnssY, 4:27-től. Illetve https://bhaxor.blog.hu/2018/09/02/ismerkedes_a

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

Az előző feladat java átírata. Ennek fordításához és futtatásához szükséges rendelkezni az openjdk8-as csomaggal.

A telepítés:

```
sudo apt-get install openjdk-8-jdk
```

A következő program Bátfai Norbert munkájának egy általam kiegészített változata (Nagyító):

```
import java.awt.*;
/*
    * MandelbrotHalmaz.java
    *
    * DIGIT 2005, Javat tanítok
    * Bátfai Norbert, nbatfai@inf.unideb.hu
    *
    */
/**
```

```
     * A Mandelbrot halmazt kiszámoló és kirajzoló osztály.
     * A programot módosította/kiegészítette: Czanik András
     *
     * @author Bátfai Norbert, nbatfai@inf.unideb.hu
     * @version 0.0.1
     */
public class MandelbrotZoom extends java.awt.Frame implements Runnable {

    protected double a, b, c, d;
    protected int szélesség, magasság;
    protected java.awt.image.BufferedImage kép;
    protected int iterációsHatár = 255;
    protected boolean számításFut = false;
    protected int sor = 0;
    protected static int pillanatfelvételSzámláló = 0;

    public static int x;
    public static int y;
    public static int mx;
    public static int my;

    public MandelbrotZoom(double a, double b, double c, double d,
            int szélesség, int iterációsHatár) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
        this.szélesség = szélesség;
        this.iterációsHatár = iterációsHatár;
        this.magasság = (int)(szélesség * ((d-c)/(b-a)));
        kép = new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
            }
        });
        addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
                    pillanatfelvétel();

                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    if(számításFut == false) {
                        MandelbrotZoom.this.iterációsHatár += 256;

                        számításFut = true;
                        new Thread(MandelbrotZoom.this).start();
                    }
```

```
            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_M) {
                if(számításFut == false) {
                    MandelbrotZoom.this.iterációsHatár += 10*256;

                    számításFut = true;
                    new Thread(MandelbrotZoom.this).start();
                }
            }
        }
    });
    addMouseListener(new java.awt.event.MouseAdapter() {
        @Override
        public void mousePressed(java.awt.event.MouseEvent me) {
            x = me.getX();
            y = me.getY();
        }

        @Override
        public void mouseReleased(java.awt.event.MouseEvent me) {
            számításFut = true;

            double dx = (b - a) / szélesség;
            double dy = (d - c) / magasság;

            double range = 60;

            double a = MandelbrotZoom.this.a+x*dx;
            double b = MandelbrotZoom.this.a+x*dx+range*dx;
            double c = MandelbrotZoom.this.d-y*dy-range*dy;
            double d = MandelbrotZoom.this.d-y*dy;

            MandelbrotZoom.this.a = a;
            MandelbrotZoom.this.b = b;
            MandelbrotZoom.this.c = c;
            MandelbrotZoom.this.d = d;

            new Thread(MandelbrotZoom.this).start();
        }
    });
    addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {

        @Override
        public void mouseMoved(java.awt.event.MouseEvent me) {
            int mx = me.getX() - x;
            int my = mx;

            repaint();
        }
    });
```

```
        setTitle("A Mandelbrot halmaz");
        setResizable(false);
        setSize(szélesség, magasság);
        setVisible(true);

        számításFut = true;
        new Thread(this).start();
    }

    public void paint(java.awt.Graphics g) {
        g.drawImage(kép, 0, 0, this);
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
    }

    public void update(java.awt.Graphics g) {
        paint(g);
    }

    public void pillanatfelvétel() {
        java.awt.image.BufferedImage mentKép =
                new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        java.awt.Graphics g = mentKép.getGraphics();
        g.drawImage(kép, 0, 0, this);
        g.setColor(java.awt.Color.BLUE);
        g.drawString("a=" + a, 10, 15);
        g.drawString("b=" + b, 10, 30);
        g.drawString("c=" + c, 10, 45);
        g.drawString("d=" + d, 10, 60);
        g.drawString("n=" + iterációsHatár, 10, 75);
        g.dispose();

        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("MandelbrotZoom_");
        sb.append(++pillanatfelvételSzámláló);
        sb.append("_");
        sb.append(a);
        sb.append("_");
        sb.append(b);
        sb.append("_");
        sb.append(c);
        sb.append("_");
        sb.append(d);
        sb.append(".png");
```

```
        try {
            javax.imageio.ImageIO.write(mentKép, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }



        public void run() {

        double dx = (b-a)/szélesség;
        double dy = (d-c)/magasság;
        double reC, imC, reZ, imZ, ujreZ, ujimZ;
        int rgb;
        int iteráció = 0;

        for(int j=0; j<magasság; ++j) {
            sor = j;
            for(int k=0; k<szélesség; ++k) {
                reC = a+k*dx;
                imC = d-j*dy;
                reZ = 0;
                imZ = 0;
                iteráció = 0;

                while(reZ*reZ + imZ*imZ < 4 && iteráció < iterációsHatár) {
                    ujreZ = reZ*reZ - imZ*imZ + reC;
                    ujimZ = 2*reZ*imZ + imC;
                    reZ = ujreZ;
                    imZ = ujimZ;

                    ++iteráció;

                }
                iteráció %= 256;
                rgb = (255-iteráció)|
                        ((255-iteráció) << 8) |
                        ((255-iteráció) << 16);
                kép.setRGB(k, j, rgb);
            }
            repaint();
        }
        számításFut = false;
    }
    public static void main(String[] args) {
        new MandelbrotZoom(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```

Daraboljuk fel és nézzük meg a függvényeket részenként. Itt már a kommenteket és a licenset kihagyom helytakarékosság célján:

```
public class MandelbrotHalmaz extends java.awt.Frame implements Runnable {
    protected double a, b, c, d;
    protected int szélesség, magasság;
    protected java.awt.image.BufferedImage kép;
    protected int iterációsHatár = 255;
    protected boolean számításFut = false;
    protected int sor = 0;
    protected static int pillanatfelvételSzámláló = 0;
}
```

Itt még nagy csoda nem történik. Az osztályon belül létrehozzuk a szükséges változókat.

```
public MandelbrotHalmaz(double a, double b, double c, double d,
        int szélesség, int iterációsHatár) {
    this.a = a;
    this.b = b;
    this.c = c;
    this.d = d;
    this.szélesség = szélesség;
    this.iterációsHatár = iterációsHatár;
    this.magasság = (int)(szélesség * ((d-c)/(b-a)));
    kép = new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);

    addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
            setVisible(false);
            System.exit(0);
        }
    });

    addKeyListener(new java.awt.event.KeyAdapter() {
        public void keyPressed(java.awt.event.KeyEvent e) {
            if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
                pillanatfelvétel();

            else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                if(számításFut == false) {
                    MandelbrotHalmaz.this.iterációsHatár += 256;
                    számításFut = true;
                    new Thread(MandelbrotHalmaz.this).start();
```

```
                }

            } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_M) {
                if(számításFut == false) {
                    MandelbrotHalmaz.this.iterációsHatár += 10*256;
                    számításFut = true;
                    new Thread(MandelbrotHalmaz.this).start();
                }
            }
        }
    });
    setTitle("A Mandelbrot halmaz");
    setResizable(false);
    setSize(szélesség, magasság);
    setVisible(true);
    számításFut = true;
    new Thread(this).start();
}
```

Az első függvényünk egyből maga fő függvény. A változóknak átadjuk a paraméterként kapott értékeket, a `magassag` esetén kiszámoljuk azt. Az `addWindowListener`-ben létrehozzuk a `windowClosing` függvényt ami egyszerűen csak az ablak bezárásáért felel. Ezután jön egy kicsit érdekesebb rész a `addKeyListe` ben, ahol konkrétan billenytűket fogunk figyelni hogy különböző utasításokat hajtsunk végre. Ha megnyomjuk az "s"-t azzal meghívjuk a `pillanatfelvétel`-t ami screenshotot készít majd a halmazról. Amikor az "n" gombot ütjük le, akkor az iterációs határt megnöveljük 256-al és újraszámoltatjuk a halmazt egy pontosabb eredményért. Hasonlóan jár el az "m" billenytű is, azzal a különbséggel, hogy itt nem 256-al növelünk, hanem a 2560-al, azaz a tízszeresével. Végül pedig beállítjuk az ablak tulajdonságait, úgyis mint a címét, megtiltjuk az átméretezést, meghatározzuk a méretet és láthatóvá tesszük (ez utóbbi tulajdonságát megvonjuk a bezárás előtt).

```
addMouseListener(new java.awt.event.MouseAdapter() {
    @Override
    public void mousePressed(java.awt.event.MouseEvent me) {
        x = me.getX();
        y = me.getY();
    }

    @Override
    public void mouseReleased(java.awt.event.MouseEvent me) {
        számításFut = true;

        double dx = (b - a) / szélesség;
        double dy = (d - c) / magasság;

        double range = 60;

        double a = MandelbrotZoom.this.a+x*dx;
        double b = MandelbrotZoom.this.a+x*dx+range*dx;
```

```
            double c = MandelbrotZoom.this.d-y*dy-range*dy;
            double d = MandelbrotZoom.this.d-y*dy;

            MandelbrotZoom.this.a = a;
            MandelbrotZoom.this.b = b;
            MandelbrotZoom.this.c = c;
            MandelbrotZoom.this.d = d;

            new Thread(MandelbrotZoom.this).start();
        }
});
addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {

    @Override
    public void mouseMoved(java.awt.event.MouseEvent me) {
        int mx = me.getX() - x;
        int my = mx;

        repaint();
    }
});
```

Ezt a részt külön venném, azon egyszerű okból, hogy ez az én hozzáaadott értékem a programhoz. Az eredeti verzió sajnos nem tudott nagyítani, ami a feladat kritériuma is. Pont ezért ebben a rövid részletben megoldottam ezt, hogy az egérrel kijelölt területbe zoomoljunk bele. Az elv egy az egyben ugyan az mint a C++ verzióban. Tehát bekérjük az egér pozícióját mikor lenyomjuk és mikor elengedjük. Közben mikör mozog az egér, a `my` és `mx`-ben eltároljuk az így kijelölt négyzet fő pontjait, majd felengedéskor ezeket átadjuk a konstruktornak, hogy ezalapján legyen szíves újraszámolni a halmazt.

```
public void paint(java.awt.Graphics g) {
    g.drawImage(kép, 0, 0, this);
    if(számításFut) {
        g.setColor(java.awt.Color.RED);
        g.drawLine(0, sor, getWidth(), sor);
    }
}

public void update(java.awt.Graphics g) {
    paint(g);
}

public void pillanatfelvétel() {
    java.awt.image.BufferedImage mentKép =
            new java.awt.image.BufferedImage(szélesség, magasság,
            java.awt.image.BufferedImage.TYPE_INT_RGB);
    java.awt.Graphics g = mentKép.getGraphics();
    g.drawImage(kép, 0, 0, this);
    g.setColor(java.awt.Color.BLUE);
```

```
    g.drawString("a=" + a, 10, 15);
    g.drawString("b=" + b, 10, 30);
    g.drawString("c=" + c, 10, 45);
    g.drawString("d=" + d, 10, 60);
    g.drawString("n=" + iterációsHatár, 10, 75);
    g.dispose();

    StringBuffer sb = new StringBuffer();
    sb = sb.delete(0, sb.length());
    sb.append("MandelbrotHalmaz_");
    sb.append(++pillanatfelvételSzámláló);
    sb.append("_");

    sb.append(a);
    sb.append("_");
    sb.append(b);
    sb.append("_");
    sb.append(c);
    sb.append("_");
    sb.append(d);
    sb.append(".png");

    try {
        javax.imageio.ImageIO.write(mentKép, "png",
                new java.io.File(sb.toString()));
    } catch(java.io.IOException e) {
        e.printStackTrace();
    }
}
```

A következő szakasz három függvényt tartalmaz. Az első a `paint` Ő felel azért hogy a mandelbrot halmaz meg is jelenjen a képernyőkön. Másik extra feature, hogy ha újra számoltatjuk a halmazt esetlegesen egy nagyobb iterációban, akkor kirajzoltatunk vele egy piros vonalat, ami abban a sorban fog elhelyezkedni, ahol a számítás is tart. Semmi ördöngősség, csak egy kis apróság, mely sokat dob a program megjelenésén és segít vizualizálni, hogy hol tart az újraszámítás. Ezt követi az `update`, mely egyszerűen csak folyamatosan meghívja a `paint`-et, azaz frissíti a halmazunk állapotát, ahogy a neve is sugalja. Végül pedig a `pillanatkep`. Az előző bekezdésben említettem, hogy ezzel szeretnénk screenshot-okat kimenteni png formátumba. Ez elsőre bonyultan hangozhat, de nem olyan vészes. Létrehozunk egy új pufferelt képet, kirajzoltatjuk a képet, majd rá kékkel az iteráció szempontjából fontosabb változókat. Ha a kép megvan, akkor már csak el kéne nevezni valahogy. Ehhez kell egy új StringBuffer amiben ezt tároljuk. Ez tartalmazza a nevét, a kép sorszámát, a fontosabb változókat (a, b, c és d), majd végül ugye a kiterjesztést. Ezt mind szépen összefűzzük. És már adjuk is át a javax.imageio.ImageIO.write(mentKép, "png", new java.io.File(sb.toString()));-nak hogy elkészítse magát a képet.

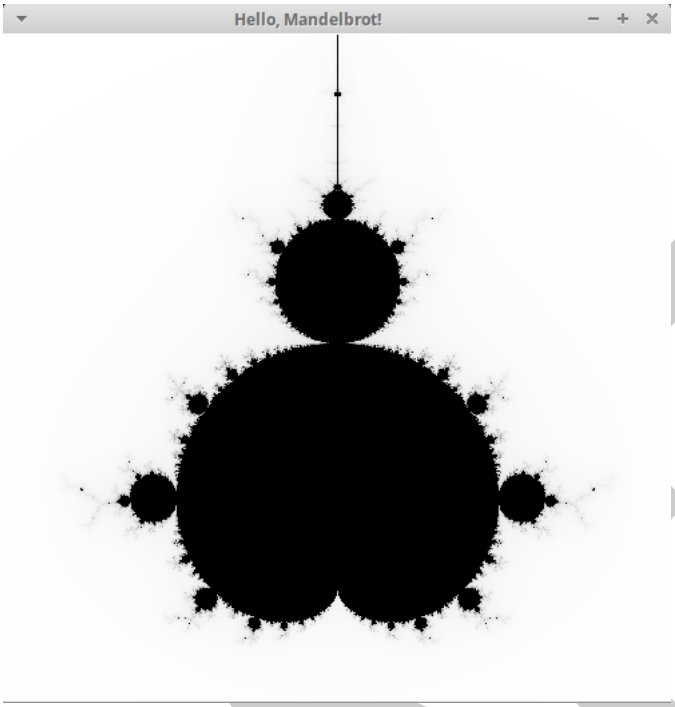
```
public void run() {
    double dx = (b-a)/szélesség;
    double dy = (d-c)/magasság;
    double reC, imC, reZ, imZ, ujreZ, ujimZ;
```

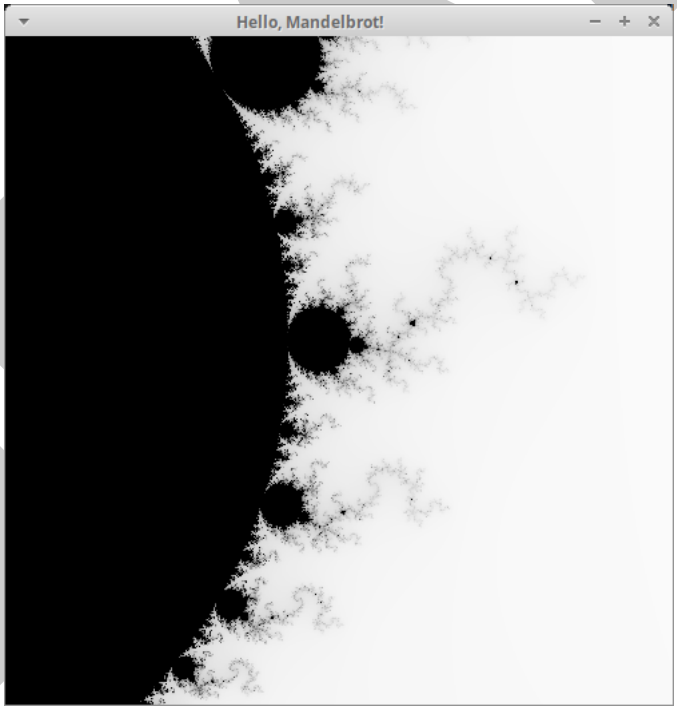
```
    int rgb;
    int iteráció = 0;
    for(int j=0; j<magasság; ++j) {
        sor = j;
        for(int k=0; k<szélesség; ++k) {
            reC = a+k*dx;
            imC = d-j*dy;
            reZ = 0;
            imZ = 0;
            iteráció = 0;
            while(reZ*reZ + imZ*imZ < 4 && iteráció < iterációsHatár) {

                ujreZ = reZ*reZ - imZ*imZ + reC;
                ujimZ = 2*reZ*imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;

                ++iteráció;

            }
            iteráció %= 256;
            rgb = (255-iteráció)|
                    ((255-iteráció) << 8) |
                    ((255-iteráció) << 16);
            kép.setRGB(k, j, rgb);
        }
        repaint();
    }
    számításFut = false;
}

public static void main(String[] args) {
    new MandelbrotHalmaz(-2.0, .7, -1.35, 1.35, 600, 255);
}
```

Végül pedig itt a lényeg, a `run`, ahol a varázslat végbemegy. Felvesszük a szükséges változókat, köztük a komplex számoknak is. Aztán jön egy klasszikus sorfolytonos mátrix bejárás, hogy képünk minden pixelét meghatározzuk. Itt már a felvett változókból elkészítjük a valódi komplex számainkat és utánna megvizsgáljuk, hogy hány iteráció alatt sikerül elhagynia a halmazt, ha sikerül persze. Az elv ugyan az mint a korábbi verzióknál, ezt nem részletezném különösebben. Majd azalapján hogy ez mennyi időbe telt neki, kap egy szín értéket. Soronként újrahívjuk a `repaint` függvényt, hogy nyomonkövethető legyen a folyamat, majd ha az utolsóval is végzett, akkor egyszerűen hamisra állítjuk a `számításFut`-ot ezzel jelezvén más függvényeknek, hogy végeztünk. Végül pedig a `void` ami egyszerűen csak megfelelően felparaméterezve meghívja a `MandelbrotHalmaz`-t.

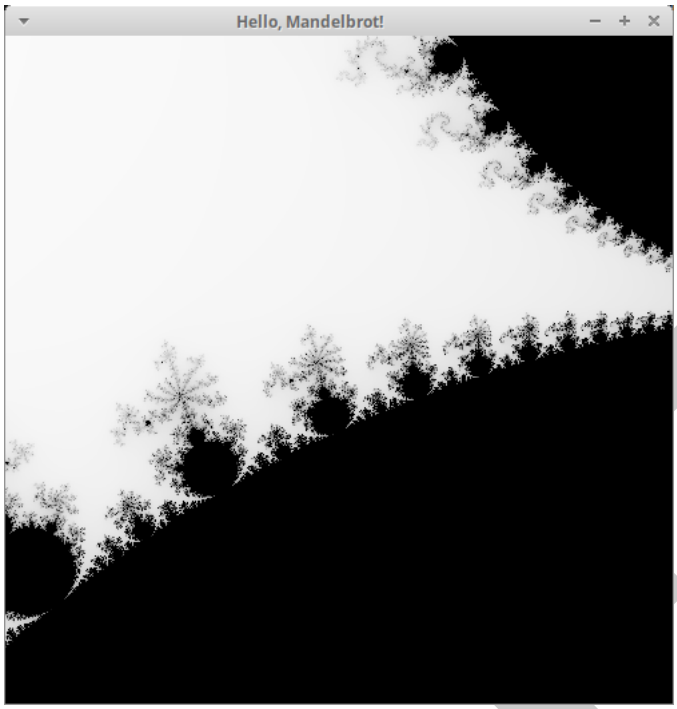
Fordítás és futtatás:

```
javac MandelbrotHalmaz.java
java MandelbrotHalmaz
```





DRAFT



5.7. Malmo: láváig fel, majd vissza le

https://youtu.be/zO6cNp8L4-Q

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 elspájzolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltéve kiszámolt szám.

Megoldás videó:

Megoldás forrása:

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

Átvett kódcsipet
A következő kódcsipetet Bátfai Norbert munkája.

Java verzió:

```
public class PolarGenerator {
    boolean nincsTarolt = true;
    double tarolt;
    public PolarGenerator() {

        nincsTarolt = true;

    }
    public double kovetkezo() {
        if(nincsTarolt) {
            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
```

DRAFT

```
                u2 = Math.random();

                v1 = 2*u1 - 1;
                v2 = 2*u2 - 1;

                w = v1*v1 + v2*v2;

            } while(w > 1);

            double r = Math.sqrt((-2*Math.log(w))/w);

            tarolt = r*v2;
            nincsTarolt = !nincsTarolt;

            return r*v1;

        } else {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }

    public static void main(String[] args) {
        PolarGenerator g = new PolarGenerator();
        for(int i=0; i<10; ++i)
            System.out.println(g.kovetkezo());
    }
}
```

Ez pedig a C++ verzió, ami már önálló munkám az előző alapján:

```
#include <iostream>
#include <math.h>
#include <stdlib.h>
#include <time.h>

using namespace std;

class PolarGenerator
{
    public:
        bool nincsTarolt = true;
        double tarolt;


    void PolGen()
    {
        nincsTarolt = true;
    }
```

```
    double kovetkezo()
    {
        if (nincsTarolt)
        {
            double u1, u2, v1, v2, w;
            do
            {
                u1 = ((double) rand() / (RAND_MAX));
                u2 = ((double) rand() / (RAND_MAX));

                v1 = 2 * u1 - 1;
                v2 = 2 * u2 - 1;

                w = v1 * v1 + v2 * v2;

            } while (w > 1);

            double r = sqrt ((-2 * log (w)) / w);

            tarolt = r * v2;
            nincsTarolt = !nincsTarolt;

            return r * v1;
        }
        else
        {
            nincsTarolt = !nincsTarolt;
            return tarolt;
        }
    }
};

int main(int argc, const char **argv)
{
    PolarGenerator g;
    srand (time(NULL));
    for (int i = 0; i < 10; ++i)
    {
        cout << g.kovetkezo() << endl;
    }

    return 0;
}
```

A polártranszformációs algoritmus véletlen szám generálásra alkalmazható. Ha megnézzük a két programot akkor láthatjuk, hogy elég hasonlók. Igazából csak a nyelvi sajátosságokban különböznek. Az egyik fő különbség, hogy míg Javaban egy könyvtárat se hívtunk meg, addig a C++-hoz kapásból 4-et (egyet a kimenetre íráshoz, kettőt a random szám generáláshoz és mégegyet a matematikai függvények használatához).

Őszintén nem vagyok egy nagy Java-s, nem nagyon próbálgattam korábban, mert nem éreztem szükségét OO környezetnek, kivéve unity-ben, ott viszont C#-ban dolgoztam. Viszont a most szerzet tapasztalataim alapján azt mondhatom, hogy OO programozáshoz sokkal jobban tetszik a Java mint a C++. Persze ez ízlés kérdés elsősorban. Ami a legjobban tetszik, hogy letisztultabb, egyszerűbb, igaz nem kis programokhoz való, mert ott elég overkill. Dolgokat egyszerűbben tudok megvalósítani (elég a random szám generálásra gondolni : Math.random() VS srand (time(NULL)) + ((double) rand() / (RAND_MAX))). Viszont C++-ban jóval több rutinom van, ezért alapvetőleg egyszerűbb azzal dolgoznom, mert nem kell minden lépésem előtt google-zni. De mindenképp szeretném jobban megismerni ezt a nyelvet is (meg a python-t is).

Próbáljuk is ki:

```
0.500010678804642
0.7466265624656746
1.0803216647807399
-0.32064099460970763
-2.5477451034196923
0.6811730798994344
-0.44975361547907916
-0.9422083605110528
0.49015177151970096
2.058535110772562
```

Ahogyan azt vártuk, kaptunk tíz darab normalizált véletlen számot. Az OO megvalósításnak köszönhetően a kódunk sokkal olvashatóbb, átláthatóbb és a generálás matematikai hátterével se kell különösen foglalkoznunk.

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

Átvett kódcsipet
A következő kódcsipet Bátfai Norbert munkája.

Ezt a programot from skrech módon, a koronavírus miatti távoktatás során írtuk, Bátfai Norbert stream-jei alapján.

A bineáris fa építése a következő algoritmus alapján zajlik: Amennyiben 0-ás elemet kapunk, meg kell vizsgálnunk, hogy az aktuális csomópontunknak van-e nullás gyermeke. Azesetben, ha még nincs akkor létrehozunk egyet, majd visszalépünk a gyökérre. Viszont ha van akkor arra lépünk rá és olvassuk tovább a bemenetet. Hasonló képpen járunk el 1-es bemenet esetén is, csak ott értelemszerűen az egyes gyermeket vizsgáljuk.

Nézzünk rá egy példát (a bemenet: 01111001001001000111):

DRAFT

```
---------0 3 0
------0 2 0
---------1 3 0
---/ 1 0
---------------0 5 0
------------0 4 0
---------0 3 0
------1 2 0
---------1 3 0
------------1 4 0
```

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

A megoldás nem túl bonyolult. Az előző programunk rekuríz print függvényét kell csak módosítanunk. De még előtte pár szóban arról, hogy mi is az az inorder, preorder és postorder:

- Inorder: Először a bal oldali közvetlen részfát járjuk be, majd a gyökeret és végül a jobb oldalit.
- Preorder: Először a gyökeret, majd a bal aztán a jobb oldali közvetlen részfát járjuk be.
- Postorder: Először a bal aztán a jobb oldali közvetlen részfát járjuk be és végül a gyökeret.

Az előbb már néztünk példát az inorder fára, de most akkor nézzünk meg egy preodert majd egy postordert is.

Preorder:

```
---/ 1 0
------0 2 0
---------0 3 0
---------1 3 0
------1 2 0
---------0 3 0
------------0 4 0
---------------0 5 0
---------1 3 0
------------1 4 0
```

Postorder:

```
---------0 3 0
---------1 3 0
------0 2 0
---------------0 5 0
------------0 4 0
---------0 3 0
------------1 4 0
---------1 3 0
------1 2 0
---/ 1 0
```

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:

Ebben az esetben felhasnzáljuk a már megírt LZWBinFa programot (z3a7.cpp), mely eleve úgy lett megírva, hogy a gyökér kompozícióban van a fával.

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:

Itt ugyan úgy felhasználjuk, viszont itt már át kell dolgoznunk kitcsit, hogy a gyökér elemünkből mostmár mutató legyen.

A feladat nem olyan nehéz mint ahogy gondoluk. Először is a gyökér csomópont helyénél helyet kell foglalnunk a gyökérnek. Majd az így kapott gyökér mutatónkat behelyettesítjük oda ahol eddig a gyökér változóra hivatkoztunk (ez csak annyit tesz, hogy kiszedjük előle a címképző operátort).

6.6. Mozgató szemantika

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

Megoldás videó:

DRAFT

Megoldás forrása:

Ezt a feladatot, a második feladathoz hasonló módon valósítjuk meg. Egész pontosan annak a programnak a továbbfejlesztéséről van szó Bátfai Norbert vezénylése alatt.

Az LZWTree osztályunk úgy épül fel, hogy az `LZWTree` osztályon belül megtalálhatóak beágyazott `Csomopont` osztályú objektumok, ezek alkotják a fát. Ebből következik, hogy a fát úgy tudjuk másolni, hogy ezeket a beágyazott csomópontokat másoljuk, rekurzívan.

```
        BinTree(const BinTree & old)
{
    std::cout << "BT copy ctor" << std::endl;

    root = cp(old .root, old.treep);
}

Node * cp(Node *node, Node *treep)
{
    Node * newNode = nullptr;

    if(node)
    {
        newNode = new Node(node -> getValue());

        newNode -> leftChild(cp(node -> leftChild(), treep));
        newNode -> rightChild(cp(node -> rightChild(), treep));

        if (node == treep)
        {
            this -> treep = newNode;
        }
    }

    return newNode;
}

BinTree & operator=(const BinTree & old)
{
    std::cout << "BT copy assign" << std::endl;

    BinTree tmp{old};
    std::swap(*this, tmp);
    return *this;
}
BinTree(BinTree && old)
{
    std::cout << "BT move ctor" << std::endl;

    root = nullptr;
    *this = std::move(old);
}
BinTree & operator=(BinTree && old)
```

```
        {
            std::cout << "BT move assign" << std::endl;

            std::swap(old.root, root);
            std::swap(old.treep, treep);

            return *this;
    }
```

6.7. Malmo: 5x5x5

https://youtu.be/aoyZWakqNGY

DRAFT

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:

Ebben a feladatban a hangyák mozgását fogjuk szimulálni. Fontos megjegyezni, hogy a hangyák nem csak össze-vissza mozognak, hanem tájékozódnak. Méghozzá fejlett szaglásuk segítségével követik egymás feromon nyomát. Ennek főleg akkor van szerepe, ha valamelyik hangya talál valamit (pl. ételt), akkor a megfelelő szagminta kibocsátásával oda csalhatja a társait, hogy segítsenek. Ennek a legnagyobb szerepe ott van, hogy a hangyák mozgásának a tanulmányozásával megfigyelhetjük, hogy hogyan találják meg így a legrövidebb útvonalat. Ezek az ismeretek sokat segíthetnek azútkereső algoritmusoknak is (lásd. navmesh).

Kezdésnek nézzük meg a hangyáinkat:

```
 #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;

    }

};
```

RAFT

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

#endif
```

A main.cpp-ben első sorban lekezeljük a lehetséges argumentumokat, a program hívásához (úgy mint az ablak méretét, a hangyák sebességét, számát, stb). Ezeket pedig átadjuk az ablak elkészítéséhez. Itt még szerepel a qsrand, ami a véletlenszerű irány meghatározásához fog használni a hangyánk.

Nézzük meg az antwin.cpp-t:

```
void AntWin::paintEvent ( QPaintEvent* )
{
    QPainter qpainter ( this );

    grid = grids[gridIdx];

    for ( int i=0; i<height; ++i ) {
        for ( int j=0; j<width; ++j ) {

            double rel = 255.0/max;

            qpainter.fillRect ( j*cellWidth, i*cellHeight,
                                cellWidth, cellHeight,
                                QColor ( 255 - grid[i][j]*rel,
                                         255,
                                         255 - grid[i][j]*rel)  ↩
                                   );

            if ( grid[i][j] != min )
            {
                qpainter.setPen (
                    QPen (
                        QColor ( 255 - grid[i][j]*rel,
                                 255 - grid[i][j]*rel, 255),
                        1 )
                );

                qpainter.drawRect ( j*cellWidth, i*cellHeight,
                                    cellWidth, cellHeight );
            }



            qpainter.setPen (
                QPen (
                    QColor (0,0,0 ),
                    1 )
            );

            qpainter.drawRect ( j*cellWidth, i*cellHeight,
```

```
                                        cellWidth, cellHeight );

                }
            }

            for ( auto h: *ants) {
                qpainter.setPen ( QPen ( Qt::black, 1 ) );

                qpainter.drawRect ( h.x*cellWidth+1, h.y*cellHeight+1,
                                    cellWidth-2, cellHeight-2 );

            }

            qpainter.end();
        }
```

Itt a `paintEvent` van kiemelve, mely a GUI újrarajzolásáért felel (például mikor az update függvény lefut). Amit csinál az alapvetőleg egyszerű, mert csak kirajzolja a világot, a feromon nyomot és a hangyáinkat.

Folyt. köv.

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Az életjáték (Game of Life) John Conway Brit matematikus munkája. Játék ként van említve, de a passzív játék megnevezés talán ponstosabb, ugyanis a játékosnak a kezdő alakzat megrajzolásán túl nem sok szerepe van a játékban, inkább megfigyelőként vsz részt. Érdekes, hogy egy időben ez egy aránylag népszerű játék volt Amerikában.

A játék menete: Adott egy tábla (négyzetháló) ahova a játékos pontokat, úgynevezett sejteket helyezhet el. A játék körökből áll, melyek generációk ként van meghivatkozva. Ezekben a körökben a sejtünk vagy túlél, vagy elpusztúl, vagy szaporodik. Hogy az adott körben melyik törénik azt három egyszerű szabály határozza meg, a szomszédok számának függvényében (egy sejtnek 8 szomszéda lenne):

- Egy sejt 2 vagy 3 élő szomszéd esetében életben marad.
- Egy sejt elpusztul ha 2-nél kevesebb vagy 3-nál több élő szomszédja van
- Egy új sejt születik, ha pontosan 3 élő szomszédja van.

Ezekkel az egyszerű szabályokkal viszont lehetőségünk van olyan speciális alakzatokra is mint például a sikló (glider) melynek a különlegessége, hogy pár lépésen belül vissza alakúl saját magává, viszont pár egységgel odébb, ezáltal mozog.

DRAFT

A szabályok és a speciális alakzatok kihasználásával pedig létrehozhatunk akár komplex automatákat. Egy ilyet mutat be az alábbi program is. Ennek az automatának a neve Glider Gun (siklóágyú), melyet Bill Gosper alkotott meg és remekűl reprezentálja, hogy mennyi mindent lehet csinálni még egy ilyen egyszerű játékkal is.

7.3. Extra: Sejtautomaták a játékiparban:

Lehet pár embernek meglepő, de a játékiparban sok olyan algoritmust használnak, melyről nem is gondolnánk hogy erre is alkalmazható. Ilyenek a sejtautomaták is, amilyen Conway életjátéka is. Ebben a példában Sebastian Lague mutatja be, hogy hogyan is lehet ezt az algoritmust felhasználni procedurálisan generált barlangok készítéséhez. Érdemes lehet megnézni más videóit is, mert nagyon igényes és érdekes tartalmakat készít. A videó: https://youtu.be/v7yyZZjF1z4 A Github repoja: https://github.com/SebLague/-Procedural-Cave-Generation

Ebből a hangyabolyból...

...ez a szép barlang lett:

DRAFT

Mivel ez egy nyílvános repó és tutoriál anyag, ezért remélem nem gond, ha megosztom a forráskódot elemzés céljára. Minden jog fenttartva Sebastian részére.

```
using UnityEngine;
using System.Collections;
using System;

public class MapGenerator : MonoBehaviour {

  public int width;
  public int height;

  public string seed;
  public bool useRandomSeed;

  [Range(0,100)]
  public int randomFillPercent;

  int[,] map;

  void Start() {
    GenerateMap();
  }

  void Update() {
    if (Input.GetMouseButtonDown(0)) {
      GenerateMap();
    }
  }

  void GenerateMap() {
    map = new int[width,height];
```

```
    RandomFillMap();

    for (int i = 0; i < 5; i ++) {
      SmoothMap();
    }
  }


  void RandomFillMap() {
    if (useRandomSeed) {
      seed = Time.time.ToString();
    }

    System.Random pseudoRandom = new System.Random(seed.GetHashCode());

    for (int x = 0; x < width; x ++) {
      for (int y = 0; y < height; y ++) {
        if (x == 0 || x == width-1 || y == 0 || y == height -1) {
          map[x,y] = 1;
        }
        else {
          map[x,y] = (pseudoRandom.Next(0,100) < randomFillPercent)? 1: 0;
        }
      }
    }
  }

  void SmoothMap() {
    for (int x = 0; x < width; x ++) {
      for (int y = 0; y < height; y ++) {
        int neighbourWallTiles = GetSurroundingWallCount(x,y);

        if (neighbourWallTiles > 4)
          map[x,y] = 1;
        else if (neighbourWallTiles < 4)
          map[x,y] = 0;

      }
    }
  }

  int GetSurroundingWallCount(int gridX, int gridY) {
    int wallCount = 0;
    for (int neighbourX = gridX - 1; neighbourX <= gridX + 1; neighbourX  ↩
       ++) {
      for (int neighbourY = gridY - 1; neighbourY <= gridY + 1; neighbourY  ↩
         ++) {
        if (neighbourX >= 0 && neighbourX < width && neighbourY >= 0 &&  ↩
           neighbourY < height) {
          if (neighbourX != gridX || neighbourY != gridY) {
```

```
            wallCount += map[neighbourX,neighbourY];
          }
        }
        else {
          wallCount ++;
        }
      }
    }

    return wallCount;
  }


  void OnDrawGizmos() {
    if (map != null) {
      for (int x = 0; x < width; x ++) {
        for (int y = 0; y < height; y ++) {
          Gizmos.color = (map[x,y] == 1)?Color.black:Color.white;
          Vector3 pos = new Vector3(-width/2 + x + .5f,0, -height/2 + y+.5f ↩
             );
          Gizmos.DrawCube(pos,Vector3.one);
        }
      }
    }
  }

}
```

7.4. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

A játék alapmechanikáját már az előző feladatban tárgyaltuk, ezért itt már nem térnék ki rá. Inkább nézzük meg részenként. Kezdjük a sejtablak.cpp-vel:

```
SejtAblak::SejtAblak(int szelesseg, int magassag, QWidget * ↩
   parent)
: QMainWindow(parent)
{
setWindowTitle("A John Horton Conway-féle életjáték");

this->magassag = magassag;
this->szelesseg = szelesseg;
```

```
        cellaSzelesseg = 6;
        cellaMagassag = 6;

        setFixedSize(QSize(szelesseg*cellaSzelesseg, magassag* ↩
           cellaMagassag));

        racsok = new bool**[2];
        racsok[0] = new bool*[magassag];
        for(int i=0; i<magassag; ++i)
            racsok[0][i] = new bool [szelesseg];
        racsok[1] = new bool*[magassag];
        for(int i=0; i<magassag; ++i)
            racsok[1][i] = new bool [szelesseg];

        racsIndex = 0;
        racs = racsok[racsIndex];

        // A kiinduló racs minden cellája HALOTT
        for(int i=0; i<magassag; ++i)
            for(int j=0; j<szelesseg; ++j)
            racs[i][j] = HALOTT;
            // A kiinduló racsra "ELOlényeket" helyezünk
            //siklo(racs, 2, 2);

            sikloKilovo(racs, 5, 60);

        eletjatek = new SejtSzal(racsok, szelesseg, magassag, 120, this ↩
           );

        eletjatek->start();

        }
```

Itt alapozzuk meg az ablakunkat. Meghatározzunk olyanokat, mint az ablak dimenzióit és a cellaméretet. Létrehozzuk a két rácsunkat. A rácspontokat beállítjuk halott sejteknek, mivel majd csak ezután helyezzük el az automatánkat. Ezután meghívjuk a `sikloKilovo` függvényt, amely egyszerűen csak meghatározza, hogy a sikló ágyú kiinduló állapotához, mely sejteknek kell élőnek lennie és ezeket a rácspontokat halottról élőre állítja. Végül elindítjuk magát a játékot.

Itt még található egy `paintEvent` ami ugyan úgy mint a hangyaszimuláció esetén, itt is a GUI kirajzolásáért és frissitésenkénti újrarajzolásáért felel.

```
        void SejtAblak::paintEvent(QPaintEvent*) {
            QPainter qpainter(this);

            // Az aktuális
            bool **racs = racsok[racsIndex];
            // racsot rajzoljuk ki:
            for(int i=0; i<magassag; ++i) { // végig lépked a sorokon
              for(int j=0; j<szelesseg; ++j) { // s az oszlopok
```

```
                // Sejt cella kirajzolása
                if(racs[i][j] == ELO)
              qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                        cellaSzelesseg, cellaMagassag, Qt::black);
              else
                qpainter.fillRect(j*cellaSzelesseg, i*cellaMagassag,
                          cellaSzelesseg, cellaMagassag, Qt::white);
                qpainter.setPen(QPen(Qt::gray, 1));

              qpainter.drawRect(j*cellaSzelesseg, i*cellaMagassag,
                        cellaSzelesseg, cellaMagassag);
              }
            }

            qpainter.end();
          }
```

Ezen túl van még egy destruktorunk és egy vissza függvény ami csak lépteti a játékot és frissití az update függvénnyel, ezáltal kikényszerítve az újrarajzolást.

```
SejtAblak::~SejtAblak()
{
delete eletjatek;

for(int i=0; i<magassag; ++i) {
    delete[] racsok[0][i];
    delete[] racsok[1][i];
}

delete[] racsok[0];
delete[] racsok[1];
delete[] racsok;


}

void SejtAblak::vissza(int racsIndex)
{
this->racsIndex = racsIndex;
update();
}
```

A sejtszal.cpp tartalmazza lényegében az alapszabályokat. Itt a `szomszedokSzama` meghatározza a szomszédok számát, mely segítségével eldönthetjük, hogy az adott körben a sejt életben marad-e vagy szaporodik-e vagy esetleg elpusztul.

```
int SejtSzal::szomszedokSzama(bool **racs,
                  int sor, int oszlop, bool allapot) {
    int allapotuSzomszed = 0;
```

```
                // A nyolcszomszédok végigzongorázása:
                for(int i=-1; i<2; ++i)
                    for(int j=-1; j<2; ++j)
                        // A vizsgált sejtet magát kihagyva:
                        if(!((i==0) && (j==0))) {
                    // A sejttérből szélének szomszédai
                    // a szembe oldalakon ("periódikus határfeltétel")
                    int o = oszlop + j;
                    if(o < 0)
                        o = szelesseg-1;
                    else if(o >= szelesseg)
                        o = 0;

                    int s = sor + i;
                    if(s < 0)
                        s = magassag-1;
                    else if(s >= magassag)
                        s = 0;

                    if(racs[s][o] == allapot)
                        ++allapotuSzomszed;
                }

            return allapotuSzomszed;
        }
```

Az `IdoFejlodes` pedig meghatározza, hogy a szomszéd szám alapjáán ténylegesen mi lesz a sejttel. Ugye az előbbi bekezdés alapján 3 állapot álhat elő.

```
        void SejtSzal::idoFejlodes() {

            bool **racsElotte = racsok[racsIndex];
            bool **racsUtana = racsok[(racsIndex+1)%2];

            for(int i=0; i<magassag; ++i) { // sorok
                for(int j=0; j<szelesseg; ++j) { // oszlopok

                    int elok = szomszedokSzama(racsElotte, i, j,  ↩
                       SejtAblak::ELO);

                    if(racsElotte[i][j] == SejtAblak::ELO) {
                        /* Élő élő marad, ha kettő vagy három élő
                     szomszedja van, különben halott lesz. */
                        if(elok==2 || elok==3)
                            racsUtana[i][j] = SejtAblak::ELO;
                        else
                            racsUtana[i][j] = SejtAblak::HALOTT;
                    } else {
                        /* Halott halott marad, ha három élő
                     szomszedja van, különben élő lesz. */
```

```
                        if(elok==3)
                            racsUtana[i][j] = SejtAblak::ELO;
                        else
                            racsUtana[i][j] = SejtAblak::HALOTT;
                    }
                }
            }
            racsIndex = (racsIndex+1)%2;
        }
```

7.5. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Ennek a játéknak a lényege, hogy játékosok, esportolók kognitív képességeit hivatott felmérni. A játék 10 percig tart és a feladat egyszerű. A samu nevű entitáson kell tartanunk a mutót amíg csak tudjuk. Viszont samu mozog és bizonyos időközönként új entitások jelennek meg, hogy megzavarják a játékost. És ha ez nem lenne elég, ugye a játék 10 percig tart, így ez is kihívást jelenthet a játékosok számára.

A játék legfontosabb mechanikája a BrainBWin.cpp-ben található, méghozzá az `updateHeroes` függvény.

```
void BrainBWin::updateHeroes ( const QImage &image, const int &x, const int &y )
{

    if ( start && !brainBThread->get_paused() ) {

        int dist = ( this->mouse_x - x ) * ( this->mouse_x - x ) + ( this->mouse_y - y ) * ( this->mouse_y - y );

        if ( dist > 121 ) {
            ++nofLost;
            nofFound = 0;
            if ( nofLost > 12 ) {

                if ( state == found && firstLost ) {
                    found2lost.push_back ( brainBThread->get_bps() );
                }

                firstLost = true;
```

```
                                        state = lost;
                                        nofLost = 0;
                                        //qDebug() << "LOST";
                                        //double mean = brainBThread-> ↩
                                           meanLost();
                                        //qDebug() << mean;

                                        brainBThread->decComp();
                                }
                        } else {
                                ++nofFound;
                                nofLost = 0;
                                if ( nofFound > 12 ) {

                                        if ( state == lost && firstLost ↩
                                            ) {
                                                lost2found.push_back (  ↩
                                                   brainBThread-> ↩
                                                   get_bps() );
                                        }

                                        state = found;
                                        nofFound = 0;
                                        //qDebug() << "FOUND";
                                        //double mean = brainBThread-> ↩
                                           meanFound();
                                        //qDebug() << mean;

                                        brainBThread->incComp();
                                }

                        }

                }
                pixmap = QPixmap::fromImage ( image );
                update();
        }
```

Ha a játék el van indítva és nincs szüneteltetve akkor folyamatosan figyeli ez a függvény, hogy a játékos épp samura mutat-s vagy elvesztette és ha igen, akkor hányszor, továbbá, hogy hászor találja meg elvesztés után. Azt hogy elvesztette-e samut azt úgy hatűrozza meg, hogy figyeli samu és a kurzor közti távolságot. Ha ez meghaladja a 121 egységet akkor elveszítjük samut.

7.6. Malmo

Megoldás videó: https://youtu.be/WRDt8ljy_hI

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 (/tensorflow-0.9.0/tensorflow/exa
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

DRAFT

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Kezdésként mi is az a Lisp? Ugye rengeteg programozási nyelv van (olyanok is mint a beef, chicken vagy a brainfuck) és nehéz is mindet számontartani. De azért a lisp nem egy vicc nyelv, mert komoly szerepe volt az első körös mesterséges intelligencia kutásban (ez mondjuk inkább a terveőjén, John McCarthy-n múlott), mikor az első aranykorát élte az 50-60as években.

A nyelven magán eléggé érződik, hogy az 50es években született, matematikai beállítottsága miatt. Viszont van pár tulajdonsága ami a mai felhasználót könnyen elijeszthet. Ilyen például, hogy a Lisp adatstruktúrája láncolt lista, ezért minden összetartozó elemet, mint argumentumot, zárójelek kötnek össze. Ez a számítógépnek egyszerűen értelmezhető, sajnos a felhasználónaek egy hosszabb utasítás esetén kevésbé. Másik apró furcsaság lehet, hogy a programnyelv prefix-jelölést használ, ellentétben az általunk megszokott infix-jelöléstől. Magyarra fordítva ez azt jelenti, hogy például a műveleti jelek a számok előtt vannak (lsd. (* f (+ i 1))).

A feladat megoldása Besenczi Renátó videója alapján készült. https://youtu.be/3jcDl4pHFMI

```
#!/usr/bin/clisp

(format t "Give a number: ~%")

(setq n (read))

(defun factorial_iterative (n)
    (let ((f 1))
        (dotimes (i n)
        (setf f (* f (+ i 1))))
        f
    )
)

(defun factorial_recursive (n)
    (if (= n 0)
        1
```

DRAFT

```
            (* n (factorial_recursive(- n 1)))
        )
    )

    (format t "Recursive: ~%")

    (loop for i from 0 to n
        do (format t "~D! = ~D~%" i (factorial_recursive i)))

    (format t "Iterative: ~%")

    (loop for i from 0 to n
        do (format t "~D! = ~D~%" i (factorial_iterative i)))
```

A program maga nem túl bonyolúlt, bekérünk egy számot a felhasználótól, majd elvégezzük a műveletet iteratív és rekurzív függvénnyel is. Az iteratív esetén egyszerűen addig szorozgatjuk össze a számokat, míg el nem érjük a kívánt számot. Ehhez használjuk a `dotimes` makro függvényt, melyben az f értékét folyamatosan (f * (i + 1)-re állítjuk. Ehhez felhasználtuk a faktoriális függvény formális definícióját.

$$n! = \prod_{k=1}^{n} k \quad \text{minden } n \geq 0 \text{ egész számra.}$$

Talán a rekurziós eljárás mégegyszerűbb, hiszen ott a barátságosabb rekurzív képletet alkalmaztuk, azaz n! = n * (n - 1)!. Ugye a rekurzív függvények lényege, hogy meghívják saját magukat, így lehet egyértelmű, hogy mindig érdemes egy kilépési feltétellel kezdeni, mivel ha kimarad, akkor könnyen úgy járhatunk mint a ciklusok esetén, hogy egy végtelen iterációba kerülünk. A mi feltételünk itt az, hogy ha n = 0, akkor a függvény újboli meghívása helyett, egyszerűen csak kilépünk az 1-el. Ezt ugye nem kell részleteznem, a 0! definíció szerint 1. Ha pedig az n nem 1, akkor lefut a képlet része, mely újra és újra meghívja magát, míg az n el nem éri a nullát.

Futtatás után:

```
Give a number:
5
Recursive:
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
Iterative:
0! = 1
1! = 1
```

```
2! = 2
3! = 6
4! = 24
5! = 120
```

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

Megoldás forrása:

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

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

Tanulságok, tapasztalatok, magyarázat...

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

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Malmo

Ez a továbbfejlesztett verziója annak a kódnak, amivel az RFIII-ra neveztünk Tutor Tündével és tabellásak lettünk.

```
            from __future__ import print_function
#  ↩
   ------------------------------------------------------------------------------------

# Copyright (c) 2016 Microsoft Corporation
#
# Permission is hereby granted, free of charge, to any person obtaining a  ↩
   copy of this software and
# associated documentation files (the "Software"), to deal in the Software  ↩
   without restriction,
```

DRAFT

```
# including without limitation the rights to use, copy, modify, merge, publish, distribute,
# sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all copies or
# substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
# NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
# NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
# DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
# ------------------------------------------------------------------------------------------------


# Tutorial sample #2: Run simple mission using raw XML

# Added modifications by Norbert Bátfai (nb4tf4i) batfai.norbert@inf.unideb.hu, mine.ly/nb4tf4i.1
# 2018.10.18, https://bhaxor.blog.hu/2018/10/18/malmo_minecraft
# 2020.02.02, NB4tf4i's Red Flowers, http://smartcity.inf.unideb.hu/~norbi/NB4tf4iRedFlowerHell


from builtins import range
import MalmoPython
import os
import sys
import time
import random
import json
import math

if sys.version_info[0] == 2:
    sys.stdout = os.fdopen(sys.stdout.fileno(), 'w', 0)  # flush print output immediately
else:
    import functools
    print = functools.partial(print, flush=True)

# Create default Malmo objects:
```

```
agent_host = MalmoPython.AgentHost()
try:
    agent_host.parse( sys.argv )
except RuntimeError as e:
    print('ERROR:',e)
    print(agent_host.getUsage())
    exit(1)
if agent_host.receivedArgument("help"):
    print(agent_host.getUsage())
    exit(0)

# -- set up the mission -- #
missionXML_file='nb4tf4i_d.xml'
with open(missionXML_file, 'r') as f:
    print("NB4tf4i's Red Flowers (Red Flower Hell) - DEAC-Hackers Battle ↩
       Royale Arena\n")
    print("NB4tf4i vörös pipacsai (Vörös Pipacs Pokol) - DEAC-Hackers ↩
       Battle Royale Arena\n\n")
    print("The aim of this first challenge, called nb4tf4i's red flowers, ↩
       is to collect as many red flowers as possible before the lava flows ↩
       down the hillside.\n")
    print("Ennek az első, az nb4tf4i vörös virágai nevű kihívásnak a célja ↩
       összegyűjteni annyi piros virágot, amennyit csak lehet, mielőtt a ↩
       láva lefolyik a hegyoldalon.\n")
    print("Norbert Bátfai, batfai.norbert@inf.unideb.hu, https://arato.inf. ↩
       unideb.hu/batfai.norbert/\n\n")
    print("Loading mission from %s" % missionXML_file)
    mission_xml = f.read()
    my_mission = MalmoPython.MissionSpec(mission_xml, True)
    my_mission.drawBlock( 0, 0, 0, "lava")


class Hourglass:
    def __init__(self, charSet):
        self.charSet = charSet
        self.index = 0
    def cursor(self):
        self.index=(self.index+1)%len(self.charSet)
        return self.charSet[self.index]

hg = Hourglass('|/-\|')

class Steve:
    def __init__(self, agent_host):
        self.agent_host = agent_host
        self.x = 0
        self.y = 0
        self.z = 0
        self.yaw = 0
```

```
        self.pitch = 0

    def pickUp(self):
        self.agent_host.sendCommand( "attack 1" )
        time.sleep(.23)

    def calcNbrIndex(self):
        if self.yaw >= 180-22.5 and self.yaw <= 180+22.5 :
            self.front_of_me_idx = 1
            self.front_of_me_idxr = 2
            self.front_of_me_idxl = 0
            self.right_of_me_idx = 5
            self.left_of_me_idx = 3
        elif self.yaw >= 180+22.5 and self.yaw <= 270-22.5 :
            self.front_of_me_idx = 2
            self.front_of_me_idxr = 5
            self.front_of_me_idxl =1
            self.right_of_me_idx = 8
            self.left_of_me_idx = 0
        elif self.yaw >= 270-22.5 and self.yaw <= 270+22.5 :
            self.front_of_me_idx = 5
            self.front_of_me_idxr = 8
            self.front_of_me_idxl = 2
            self.right_of_me_idx = 7
            self.left_of_me_idx = 1
        elif self.yaw >= 270+22.5 and self.yaw <= 360-22.5 :
            self.front_of_me_idx = 8
            self.front_of_me_idxr = 7
            self.front_of_me_idxl = 5
            self.right_of_me_idx = 6
            self.left_of_me_idx = 2
        elif self.yaw >= 360-22.5 or self.yaw <= 0+22.5 :
            self.front_of_me_idx = 7
            self.front_of_me_idxr = 6
            self.front_of_me_idxl = 8
            self.right_of_me_idx = 3
            self.left_of_me_idx = 5
        elif self.yaw >= 0+22.5 and self.yaw <= 90-22.5 :
            self.front_of_me_idx = 6
            self.front_of_me_idxr = 3
            self.front_of_me_idxl = 7
            self.right_of_me_idx = 0
            self.left_of_me_idx = 8
        elif self.yaw >= 90-22.5 and self.yaw <= 90+22.5 :
            self.front_of_me_idx = 3
            self.front_of_me_idxr = 0
            self.front_of_me_idxl = 6
            self.right_of_me_idx = 1
            self.left_of_me_idx = 7
        elif self.yaw >= 90+22.5 and self.yaw <= 180-22.5 :
```

```
            self.front_of_me_idx = 0
            self.front_of_me_idxr = 1
            self.front_of_me_idxl = 3
            self.right_of_me_idx = 2
            self.left_of_me_idx = 6
        else:
            print("There is great disturbance in the Force...")

    def run(self):
        world_state = self.agent_host.getWorldState()
        # Loop until mission ends:
        i = 0
        for i in range(30):
            self.agent_host.sendCommand( "jumpmove 1" )
            time.sleep(0.1)
            self.agent_host.sendCommand( "move 1" )
            time.sleep(0.1)

        self.agent_host.sendCommand( "turn 1" )
        time.sleep(0.2)
        self.agent_host.sendCommand( "look 1" )
        self.agent_host.sendCommand( "look 1" )

        while world_state.is_mission_running:

            #print(">>> nb4tf4i arena ----------------------------------\n ↩
                ")
            self.action(world_state)
            #print("nb4tf4i arena >>> ----------------------------------\n ↩
                ")

            world_state = self.agent_host.getWorldState()

    def action(self, world_state):

        for error in world_state.errors:
            print("Error:", error.text)
        if world_state.number_of_observations_since_last_state == 0:
            #print("    NO OBSERVATIONS NO ACTIONS")
            return False

            #if world_state.number_of_observations_since_last_state != 0:
        print("--- nb4tf4i arena ----------------------------------\n")

        sensations = world_state.observations[-1].text
        print("    sensations: ", sensations)
        observations = json.loads(sensations)
        nbr= observations.get("nbr3x3", 0)
        print("    3x3x3 neighborhood of Steve: ", nbr)
```

```
        if "Yaw" in observations:
            self.yaw = int(observations["Yaw"])
        if "Pitch" in observations:
            self.pitch = int(observations["Pitch"])
        if "XPos" in observations:
            self.x = int(observations["XPos"])
        if "ZPos" in observations:
            self.z = int(observations["ZPos"])
        if "YPos" in observations:
            self.y = int(observations["YPos"])

        print("    Steve's Coords: ", self.x, self.y, self.z)
        print("    Steve's Yaw: ", self.yaw)
        print("    Steve's Pitch: ", self.pitch)

        '''
                if "LineOfSight" in observations:
                    LineOfSight = observations["LineOfSight"]
                    self.lookingat = LineOfSight["type"]
                print("    Steve's <): ", self.lookingat)

                if self.lookingat == "red_flower":
                    print("    VIRAG!!")
        '''

        self.calcNbrIndex()

                #checking corners

        if nbr[self.front_of_me_idx+9] == "dirt" and nbr[self. ↩
           left_of_me_idx+9] == "dirt":
            self.agent_host.sendCommand( "turn 1" )
            time.sleep(0.2)

        else:
            print("There is no corner")

                #checking lava

        if nbr[self.left_of_me_idx+18]=="flowing_lava" or nbr[self. ↩
           front_of_me_idx+9]=="flowing_lava" or nbr[self.front_of_me_idx ↩
           +18]=="flowing_lava":
            self.agent_host.sendCommand( "strafe 1" )
            time.sleep(0.1)
            self.agent_host.sendCommand( "strafe 1" )
            time.sleep(0.1)

                #checking traps

        if nbr[self.front_of_me_idx+9]=="dirt" and nbr[self.right_of_me_idx ↩
```

```
            +9]=="dirt" and nbr[self.left_of_me_idx+9]=="dirt":
                print("     IT'S A TRAAAAP")
                self.agent_host.sendCommand( "jumpmove 1" )
                time.sleep(0.1)

        if nbr[self.front_of_me_idx+9]=="dirt" and nbr[self.left_of_me_idx ↩
            +9]=="air":
                self.agent_host.sendCommand( "turn 1" )
                time.sleep(0.2)
                self.agent_host.sendCommand( "jumpstrafe 1" )
                time.sleep(0.1)
                self.agent_host.sendCommand( "jumpstrafe 1" )
                time.sleep(0.1)

        if nbr[self.front_of_me_idx+9]=="red_flower":
                print("    VIRAGOT SZEDEK!!")
                #self.agent_host.sendCommand( "move 1" )
                #time.sleep(0.2)
                self.pickUp()
                time.sleep(0.2)
                self.agent_host.sendCommand( "jumpstrafe 1" )
                time.sleep(0.1)
                self.agent_host.sendCommand( "jumpstrafe 1" )
                time.sleep(0.1)

        if nbr[self.front_of_me_idx+9] == "air" and nbr[self. ↩
            front_of_me_idx] == "dirt":
                self.agent_host.sendCommand( "move 1" )
                time.sleep(0.05)

        if nbr[self.front_of_me_idx+9] == "air" and nbr[self. ↩
            front_of_me_idx] == "air":
                self.agent_host.sendCommand( "move 1" )
                time.sleep(0.05)

                #self.agent_host.sendCommand( "look -1" )
                #self.agent_host.sendCommand( "look -1" )



num_repeats = 1
for ii in range(num_repeats):

    my_mission_record = MalmoPython.MissionRecordSpec()

    # Attempt to start a mission:
    max_retries = 6
    for retry in range(max_retries):
        try:
```

```
            agent_host.startMission( my_mission, my_mission_record )
            break
        except RuntimeError as e:
            if retry == max_retries - 1:
                print("Error starting mission:", e)
                exit(1)
            else:
                print("Attempting to start the mission:")
                time.sleep(2)

    # Loop until mission starts:
    print("   Waiting for the mission to start ")
    world_state = agent_host.getWorldState()

    while not world_state.has_mission_begun:
        print("\r"+hg.cursor(), end="")
        time.sleep(0.15)
        world_state = agent_host.getWorldState()
        for error in world_state.errors:
            print("Error:",error.text)

    print("NB4tf4i Red Flower Hell running\n")
    steve = Steve(agent_host)
    steve.run()

print("Mission ended")
# Mission has ended.
```

Ez pedig c++ átírata:

```
            //  ↩
               ------------------------------------------------------------------

            //  Copyright (c) 2016 Microsoft Corporation
            //
            //  Permission is hereby granted, free of charge, to any person ↩
                obtaining a copy of this software and
            //  associated documentation files (the "Software"), to deal in ↩
                the Software without restriction,
            //  including without limitation the rights to use, copy,  ↩
               modify, merge, publish, distribute,
            //  sublicense, and/or sell copies of the Software, and to  ↩
               permit persons to whom the Software is
            //  furnished to do so, subject to the following conditions:
            //
            //  The above copyright notice and this permission notice shall ↩
                be included in all copies or
            //  substantial portions of the Software.
            //
```

```
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
// --------------------------------------------------------------------------------------------


// Malmo:
#include <AgentHost.h>
#include <ClientPool.h>
#include <boost/property_tree/json_parser.hpp>
#include <boost/property_tree/ptree.hpp>
#include <boost/foreach.hpp>
using namespace malmo;

// STL:
#include <cstdlib>
#include <exception>
#include <iostream>
using namespace std;

#include <boost/property_tree/ptree.hpp>
#include <boost/property_tree/json_parser.hpp>

vector<string> GetItems(WorldState world_state,std::stringstream & ss, boost::property_tree::ptree & pt)
{
    vector<string> nbr3x3;

    ss.clear();
    pt.clear();

    ss << world_state.observations.at(0).get()->text;
    boost::property_tree::read_json(ss, pt);
    BOOST_FOREACH(boost::property_tree::ptree::value_type &v, pt.get_child("nbr3x3"))
    {
        assert(v.first.empty());
        nbr3x3.push_back(v.second.data());
    }

    return nbr3x3;
}
```

```
            void calcNbrIndex();
                int front_of_me_idx = 0;
                int front_of_me_idxr = 0;
                int front_of_me_idxl = 0;
                int right_of_me_idx = 0;
                int left_of_me_idx = 0;

                int yaw = 0;

        int main(int argc, const char **argv)
        {

            int virag = 0;
            int y = 0;

            AgentHost agent_host;

            try
            {
                agent_host.parseArgs(argc, argv);
            }
            catch( const exception& e )
            {
                cout << "ERROR: " << e.what() << endl;
                cout << agent_host.getUsage() << endl;
                return EXIT_SUCCESS;
            }
            if( agent_host.receivedArgument("help") )
            {
                cout << agent_host.getUsage() << endl;
                return EXIT_SUCCESS;
            }

            std::ifstream xmlf{"nb4tf4i_d.xml"};
            std::string xml{std::istreambuf_iterator<char>(xmlf), std:: ↩
               istreambuf_iterator<char>()};

            MissionSpec my_mission{xml, true};


            MissionRecordSpec my_mission_record("./saved_data.tgz");


            int attempts = 0;
            bool connected = false;
            do {
                try {
                    agent_host.startMission(my_mission,  ↩
                       my_mission_record);
                    connected = true;
```

```
            }
            catch (exception& e) {
                cout << "Error starting mission: " << e.what() <<  ↩
                   endl;
                attempts += 1;
                if (attempts >= 3)
                    return EXIT_FAILURE;    // Give up after three  ↩
                       attempts.
                else
                    boost::this_thread::sleep(boost::posix_time:: ↩
                       milliseconds(1000));   // Wait a second and  ↩
                       try again.
            }
        } while (!connected);

        WorldState world_state;

        cout << "Waiting for the mission to start" << flush;
        do {
            cout << "." << flush;
            boost::this_thread::sleep(boost::posix_time:: ↩
               milliseconds(100));
            world_state = agent_host.getWorldState();
            for( boost::shared_ptr<TimestampedString> error :  ↩
               world_state.errors )
                cout << "Error: " << error->text << endl;
        } while (!world_state.has_mission_begun);
        cout << endl;


        for (int i = 0; i < 30; i++)
        {
            agent_host.sendCommand("jumpmove 1");
            boost::this_thread::sleep(boost::posix_time:: ↩
               milliseconds(100));
            agent_host.sendCommand("move 1");
            boost::this_thread::sleep(boost::posix_time:: ↩
               milliseconds(100));
        }
        agent_host.sendCommand("turn 1");
        boost::this_thread::sleep(boost::posix_time::milliseconds ↩
           (200));
        agent_host.sendCommand("look 1");
        agent_host.sendCommand("look 1");

        // main loop:
        do {

            if(world_state.number_of_observations_since_last_state  ↩
               != 0)
```

```
                    {
                        std::stringstream ss;
                        ss << world_state.observations.at(0).get()->text;
                        boost::property_tree::ptree pt;
                        boost::property_tree::read_json(ss, pt);

                        vector<std::string> nbr3x3;

                        nbr3x3 = GetItems(world_state,ss,pt);
                        for(vector< boost::shared_ptr< TimestampedString ↩
                           >>::iterator it = world_state.observations.begin ↩
                           ();it !=world_state.observations.end();++it)
                        {
                        boost::property_tree::ptree pt;
                        istringstream iss((*it)->text);
                        boost::property_tree::read_json(iss, pt);

                        //string x =pt.get<string>("LineOfSight.type");
                        //string LookingAt =pt.get<string>("XPos");
                        //y = pt.get<double>("YPos");
                        yaw = pt.get<double>("Yaw");
                        //cout<<" Steve's Coords: "<<y<<" "<<yaw<<" "<<"RF ↩
                           :"<<virag;
                        }

                        calcNbrIndex();

                        //checking corners

                        if (nbr3x3[front_of_me_idx+9] == "dirt" && nbr3x3[ ↩
                           left_of_me_idx+9] == "dirt")
                        {
                            agent_host.sendCommand("turn 1");
                            boost::this_thread::sleep(boost::posix_time:: ↩
                               milliseconds(300));
                        }
                        else
                            cout << "\nThere is no corner";

                        //checking lava
                        if (nbr3x3[left_of_me_idx+18] == "flowing_lava" ||  ↩
                           nbr3x3[front_of_me_idx+9] == "flowing lava"
                            || nbr3x3[front_of_me_idx+18] == "flowing_lava" ↩
                               )
                        {
                            agent_host.sendCommand("strafe 1");
                            boost::this_thread::sleep(boost::posix_time:: ↩
                               milliseconds(100));
                            agent_host.sendCommand("strafe 1");
                            boost::this_thread::sleep(boost::posix_time:: ↩
```

```
            milliseconds(100));
    }

    //checking traps
    if (nbr3x3[front_of_me_idx+9] == "dirt" && nbr3x3[ ↩
       right_of_me_idx+9] == "dirt"
        && nbr3x3[left_of_me_idx+9] == "dirt")
    {
        cout << "\nIt's a TRAAAAP";
        agent_host.sendCommand("jumpmove 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(100));
    }

    if (nbr3x3[front_of_me_idx+9] == "dirt" && nbr3x3[ ↩
       left_of_me_idx+9] == "air")
    {
        agent_host.sendCommand("turn 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(200));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(100));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(100));
    }

    //finding flower
    if (nbr3x3[front_of_me_idx+9] == "red_flower")
    {
        cout << "\nVIRÁGOT SZEDEK!!";
        agent_host.sendCommand("attack 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(230));
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(200));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(100));
        agent_host.sendCommand("jumpstrafe 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
            milliseconds(100));
    }

    if (nbr3x3[front_of_me_idx+9] == "air" && nbr3x3[ ↩
       front_of_me_idx] == "dirt")
    {
        agent_host.sendCommand("move 1");
        boost::this_thread::sleep(boost::posix_time:: ↩
```

```
                        milliseconds(50));
                }


                if (nbr3x3[front_of_me_idx+9] == "air" && nbr3x3[ ↩
                   front_of_me_idx] == "air")
                {
                    agent_host.sendCommand("move 1");
                    boost::this_thread::sleep(boost::posix_time:: ↩
                       milliseconds(50));
                }
            }

            world_state = agent_host.getWorldState();
            for( boost::shared_ptr<TimestampedString> error :  ↩
               world_state.errors )
                cout << "Error: " << error->text << endl;


        } while (world_state.is_mission_running);

        cout << "Mission has stopped." << endl;

        return EXIT_SUCCESS;

    }

    void calcNbrIndex()
        {
            if (yaw >= 180-22.5 and yaw <= 180+22.5)
            {
                front_of_me_idx = 1;
                front_of_me_idxr = 2;
                front_of_me_idxl = 0;
                right_of_me_idx = 5;
                left_of_me_idx = 3;
            }
            else if (yaw >= 180+22.5 and yaw <= 270-22.5)
            {
                front_of_me_idx = 2;
                front_of_me_idxr = 5;
                front_of_me_idxl =1;
                right_of_me_idx = 8;
                left_of_me_idx = 0;
            }
            else if (yaw >= 270-22.5 and yaw <= 270+22.5)
            {
                front_of_me_idx = 5;
                front_of_me_idxr = 8;
                front_of_me_idxl = 2;
```

```
                right_of_me_idx = 7;
                left_of_me_idx = 1;
            }
            else if (yaw >= 270+22.5 and yaw <= 360-22.5)
            {
                front_of_me_idx = 8;
                front_of_me_idxr = 7;
                front_of_me_idxl = 5;
                right_of_me_idx = 6;
                left_of_me_idx = 2;
            }
            else if (yaw >= 360-22.5 or yaw <= 0+22.5)
            {
                front_of_me_idx = 7;
                front_of_me_idxr = 6;
                front_of_me_idxl = 8;
                right_of_me_idx = 3;
                left_of_me_idx = 5;
            }
            else if (yaw >= 0+22.5 and yaw <= 90-22.5)
            {
                front_of_me_idx = 6;
                front_of_me_idxr = 3;
                front_of_me_idxl = 7;
                right_of_me_idx = 0;
                left_of_me_idx = 8;
            }
            else if (yaw >= 90-22.5 and yaw <= 90+22.5)
            {
                front_of_me_idx = 3;
                front_of_me_idxr = 0;
                front_of_me_idxl = 6;
                right_of_me_idx = 1;
                left_of_me_idx = 7;
            }
            else if (yaw >= 90+22.5 and yaw <= 180-22.5)
            {
                front_of_me_idx = 0;
                front_of_me_idxr = 1;
                front_of_me_idxl = 3;
                right_of_me_idx = 2;
                left_of_me_idx = 6;
            }
            else
                cout<<"There is great disturbance in the Force...";
        }
```

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

A számítógépek programozására szolgáló programozási nyelveket, három típusba/szintbe soroljuk:

- gépi nyelv (amin a számítógép beszél)
- assembly nyelv (egyfajta átmenete a két másik szintnek)
- magas szintű nyelv (ez már valamivel közelebb áll az általunk beszélt emberi nyelvhez)

A magas szintű programozási nyelven megírt programokat forrásprogramnak vagy forrásszövegnek (vagy akár forráskódnak) hívják. Ezt viszont nem írhatjuk meg akárhogy. Minden magas szintű programozási nyelvnek megvan a maga szintaktikája (a nyelvre vonatkozó formai és "nyelvtani" megszoritások) melyeket be kell tartania a programozónak, különben a fordítás során szintaktikai hibát kapunk. Ezen túl vannak még úgynevezett szemantikai szabályok is, melyek a tartalmiés értelmezési szabályok együttese. Tehát egy magas szintű nyelvet a szintaktikai és a szemantikai szabályok határoznak meg, különböztetnek meg a többitől.

Ha megírtuk a forrásprogramot, abból még nem lesz futtathó program, hiszen a processzur nem beszéli a magas szintű nyelveket, ezért a programunkat a processzor számára is értelmezhetővé kell tenni, azaz lefordítani. Itt lépnek be a fordítók, mely adott magas szintű nyelvből képes futtatható állományt (tárgyprogramot) előállítani. Ehhez a következő lépéseket hajtja végre:

- lexikális elemzés (lexikális egységekre darabolja a forrásszöveget)
- szintaktikai elemzés (szintaktikai megkötéseknek teljseülésének ellenőrzése)
- szemantikai elemzés (szemantikai megkötéseknek teljseülésének ellenőrzése)
- kódgenerálás (a forrásprogramból gépi nyelvű kód készítése)

DRAFT

Általában egy fordító program egy tetszőlege magas szintű nyelvről egy tetszőleges gépi nyelvre fordít. Ezen felől léteznek olyan nyelvek (pl.: C), melyek tartalmaznak nem nyelvi elemeket is. Ilyenkor előfordítóra is szükségünk van.

Minden programozási nyelv rendelkezik egy úgynevezett hivatkozási nyelvvel, mely a programnyelv szabványa (itt kerülnek definiálásra a szemantikai és szintaktikai megszorítások). Ezen felül léteznek implementációk, mely a megszorításokat változtatva és/vagy kiegészítve platformspecifikussá (platformhoz kötötté, azaz pl.: egy processzor típust vagy egy operációs rendszert támogat, míg a többin nem képes a működésre) teszi a programot. Bár vannak úgynevezett crossplatform (platformok közti átjárás) megoldások, a teljes átjárást máig nem sikerült megoldani.

A programnyelvek (rövid) osztályozása

Imperatív nyelvek:

- Algoritmikus nyelv
- Utasítások sorozatából áll (a processzor sorrol sorra hajtja végre a parancsokat)
- Fő programozási eszköze a változó (adatok tárolására szolgál és rajta keresztül közvetlen elérhetőek a memória egységek)
- Neumann-architektúra alapú
- Alcsoportjai:
- Eljárásorientált
- Objektumorientált (ma már inkább ez az általános)

Dekleratív nyelvek:

- Nem algoritmikus
- A programozó csak a problémát veti fel, a megoldást a nyelv implementációi adják
- Korlátozott a memóriaműveletek lehetősége (szinte nincs is)
- Alcsoportjai:
- Funkcionális nyelvek
- Logikai nyelvek

Karakterkészlet

Ahogyan az írott nyelvnek, úgy a programoknak is az alapkövei a karakterek. Tekinthetjük atomi egységeinek is, hiszen nem tudjuk tovább bontani. Karakterekből épül fel minden. Viszont nem mindegy hogy milyen karaktert mikor és hol használunk, ugyanis vannak karakterek melyeket nem támogat egy nyelv, vannak speciális tulajdonsággal felruházott karakterek , stb. A félreértések elkerülése véget a karakterek alatt nem csak a betűket kell érteni, hanem:

DRAFT

- betűk
- számok
- egyéb karakterek

A legtöbb nyelv nem támogatja a nemzeti nyelveket, kizárólag az angol abc betűit (általánosan csak a nagybetűket támogatja az összes nyelv, kisbetűk esetén már vannak eltérések és ezen felül is vannak, melyek nem különböztetik meg a kis- és nagybetűket, míg mások igen)

A számjegyek helyzete már jóval egyszerűbb, ugyanis a decimális számrendszer az általánosan elterjedt.

Az egyéb karakterek közé tartoznak az elhatároló jelek, a műveleti jelek, az írásjelek és a speciális karakterek. Ezeknek általában külön szerepük van a programnyelvben.

Lexikális egységek

Röviden azok az elemei a forráslódnak melyet a fordító a lexikális elemzés során azonosít és tokenizál. Fajtái:

- Többkarakteres szimbólum: karaktersorozat melynek az adott nyelv tulajdonít jelentést. Gyakran elhatároló vagy operátori funkciót töltenek be.
- Szimbólikus nevek: lehet azonosító, kulcsszó vagy standard azonosító
- Címke: utasítások megjelölésére szolgál objektumorientált környezetben, későbbi hivatkozás céljából.
- Megjegyzés: megjegyzéseket írhatunk a kódunk későbbi olvasóinak (akár magunknak is) az egyszerűbb értelmezés céljából. Ezeket a sorokat a fordító figyelmen kívűl hagyja.
- Literál: másnéven konstansok, tehát olyan értékek amik nem változnak

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

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...

DRAFT

IV. rész

Irodalomjegyzék

DRAFT

11.3. Általános

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

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 , 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.

DRAFT