Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert Ã©s Puskas, Csaba Zsolt	2019. május 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-05-1	Feladatok kidolgozása.	PuskásCsaba

Ajánlás

„To me, you understand something only if you can program it. (You, not someone else!) Otherwise you don't really understand it, you only think you understand it."

—Gregory Chaitin, *META MATH! The Quest for Omega*, [METAMATH]

Tartalomjegyzék

I. Bevezetés 1

1. Vízió 2
1.1. Mi a programozás? . . . 2
1.2. Milyen doksikat olvassak el? . . . 2
1.3. Milyen filmeket nézzek meg? . . . 2

II. Tematikus feladatok 3

2. Helló, Turing! 5
2.1. Végtelen ciklus . . . 5
2.2. Lefagyott, nem fagyott, akkor most mi van? . . . 6
2.3. Változók értékének felcserélése . . . 7
2.4. Labdapattogás . . . 8
2.5. Szóhossz és a Linus Torvalds féle BogoMIPS . . . 10
2.6. Helló, Google! . . . 10
2.7. 100 éves a Brun tétel . . . 12
2.8. A Monty Hall probléma . . . 13

3. Helló, Chomsky! 16
3.1. Decimálisból unárisba átváltó Turing gép . . . 16
3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen . . . 17
3.3. Hivatkozási nyelv . . . 17
3.4. Saját lexikális elemző . . . 18
3.5. l33t.l . . . 19
3.6. A források olvasása . . . 20
3.7. Logikus . . . 22
3.8. Deklaráció . . . 22

4. Helló, Caesar! 24
4.1. double ** háromszögmátrix . . . 24
4.2. C EXOR titkosító . . . 25
4.3. Java EXOR titkosító . . . 26
4.4. C EXOR törő . . . 27
4.5. Neurális OR, AND és EXOR kapu . . . 30
4.6. Hiba-visszaterjesztéses perceptron . . . 31

5. Helló, Mandelbrot! 32
5.1. A Mandelbrot halmaz . . . 32
5.2. A Mandelbrot halmaz a `std::complex` osztállyal . . . 34
5.3. Biomorfok . . . 34
5.4. A Mandelbrot halmaz CUDA megvalósítása . . . 38
5.5. Mandelbrot nagyító és utazó C++ nyelven . . . 38
5.6. Mandelbrot nagyító és utazó Java nyelven . . . 39

6. Helló, Welch! 40
6.1. Első osztályom . . . 40
6.2. LZW . . . 42
6.3. Fabejárás . . . 45
6.4. Tag a gyökér . . . 46
6.5. Mutató a gyökér . . . 47
6.6. Mozgató szemantika . . . 48

7. Helló, Conway! 49
7.1. Hangyaszimulációk . . . 49
7.2. Java életjáték . . . 52
7.3. Qt C++ életjáték . . . 53
7.4. BrainB Benchmark . . . 53

8. Helló, Schwarzenegger! 55
8.1. Szoftmax Py MNIST . . . 55
8.2. Minecraft-MALMÖ . . . 55

9. Helló, Chaitin! 56
9.1. Iteratív és rekurzív faktoriális Lisp-ben . . . 56
9.2. Weizenbaum Eliza programja . . . 56
9.3. Gimp Scheme Script-fu: króm effekt . . . 56
9.4. Gimp Scheme Script-fu: név mandala . . . 56
9.5. Lambda . . . 57
9.6. Omega . . . 57

10. Helló, Gutenberg! 58
10.1. Programozási alapfogalmak . . . 58
10.2. Programozás bevezetés . . . 59
10.3. Programozás . . . 60

III. Második felvonás 61

11. Helló, Arroway! 63
11.1. A BPP algoritmus Java megvalósítása . . . 63
11.2. Java osztályok a Pi-ben . . . 63

IV. Irodalomjegyzék 64
11.3. Általános . . . 65
11.4. C . . . 65
11.5. C++ . . . 65
11.6. Lisp . . . 65

Ábrák jegyzéke

2.1. Brun-tétel 13

4.1. Neuron (Forrás http://project.mit.bme.hu/mi_almanach/books/aima/ch20s05) 31

5.1. Mandebrot Kimenet 33
5.2. Biomorf Kimenet 37
5.3. Mandebrot nagyító 39

7.1. Hangyaszimuláció 52
7.2. Sejtautomata 53
7.3. BrainB benchmark 54

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

Bátf41 Haxor Stream
A feladatokkal kapcsolatos élő adásokat sugároz a https://www.twitch.tv/nbatfai csatorna, melynek permanens archívuma a https://www.youtube.com/c/nbatfai csatornán található.

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: clips.twitch.tv/csabi333

Megoldások forrásai:

1)Egy magot 100%-ra: elsőnek a feladat legegyszerűbb részét oldjuk meg, használunk egy egyszerű, minden fordító által ismert " for(;;); " végtelen ciklust.

```
#include <stdio.h>

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

2) Egy magot 0%-ra: Ahhoz hogy ezt elérjuk a " sleep(); " függvényt fogjuk használni ami 'altatja' azt a szálat amin éppen fut, egy paraméterkent megadott ideig. Ezt egy végtelen ciklusba rakva, határozatlan ideig futtathatjuk így a programunk 0% CPU-t fog használni miközben végtelen ciklusban van. Hogy ezt a függvényt használhassuk include-olni kell a unistd.h header file-t.

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

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

3)Az összes magot 100%-ra: ehhez a feladathoz ismernünk kell hogyan fordítsunk le úgy egy programot hogy az majd párhuzamosan fusson több magon. Kevés internetes segitség után rájövünk hogy "openmp" módszer a legegyszerűbb mivel az első feladathoz csak egy sort kell hozzá adni. El ne felejtsük hogy ezt a programot máshogy kell fordítani, a "gcc -fopenmp vegtelen.c" parancsot használjuk.

```
#include <stdio.h>

int main()
{
#pragma omp parallel
for(;;);
return 0;
}
```

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

Megoldás videó: clips.twitch.tv/csabi333

Megoldás forrása:

```
            #include <stdio.h>
      int main()
      {
        int  x, y;
        printf("Kerek ket szamot:");
        scanf("%d%d",&x,&y);
        printf("x=%d \ny=%d\n",x,y);
            x = x + y;
            y = x - y;
            x = x - y;
        printf("Csere utan:\nx=%d  y=%d\n",x,y);
      return 0;
      }
```

Tanulságok, tapasztalatok, magyarázat...

2.4. Labdapattogás

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

Megoldás videó: clips.twitch.tv/csabi333

Megoldás forrása: If-el https://progpater.blog.hu/2011/02/13/megtalaltam_neo_t

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

int main()
{
  WINDOW *ablak;
  ablak = initscr();

  int x=0; int y=0; //A labda helyzetenek hosszusaga es szelessege
  int deltax=1; int deltay=1;  //Be tudjuk allitani milyen szogben ↩
     pattogjon a labda

  int mx; int my;  //Az ablak hosszusaga es szelessege

curs_set(0);  //Elrejtjuk a cursor-t hogy a labda mellett ne villogjon

  for(;;){    //vegtelen ciklussal nem all meg a labda pattogasa
    getmaxyx(ablak,my,mx);  //Valahanyszor helyzetet valt a labda a program ↩
        megnezi
                //hogy jelen esetben milyen nagy az ablak
                //igy akar a program futtatasa kozben is //valtoztathatjuk  ↩
                   az ablak meretet
```

```
    mvprintw(y,x,"O");    //ez a fuggveny kirakja az adott koordinatakra az ↩
                        //altalunk megadott karaktert
    refresh();
    usleep(50000);  //allithatjuk a labda sebesseget
    clear();    //ha nem tisztitanank az ablakot akkor folyamatos csikot  ↩
       huzna
    x=x+deltax;   //elmozditjuk a labdat az x tengelyen
    y=y+deltay;   //az y tengelyen is

//if-ekkel valtoztatjuk az iranyat ha az ablak szelere er
    if(x<=0) { deltax=deltax * -1;}
    if(x>=mx-1) { deltax=deltax * -1;}
    if(y<=0) {deltay=deltay * -1;}
    if(y>=my-1) {deltay=deltay * -1;}
  }
  return 0;
}
```

Megoldás forrása: If nélkül; fontos program részlet: https://progpater.blog.hu/2011/02/13/megtalaltam_neo_t

```
        for (;;){
        xj = (xj - 1) % mx;
        xk = (xk + 1) % mx;
        yj = (yj - 1) % my;
        yk = (yk + 1) % my;

        clear ();
mvprintw (0, 0,
    " ↩
       ------------------------------------------------------------------------
       ");
mvprintw (24, 0,
    " ↩
       ------------------------------------------------------------------------
       ");
        mvprintw (abs ((yj + (my - yk)) / 2),
                  abs ((xj + (mx - xk)) / 2), "X");
        refresh ();
        usleep (50000);

    }
```

Tanulságok, tapasztalatok, magyarázat...

If-el a program eredménye, a labda pattogása, futás közben változik az ablak mérete függvényében, tehát szabadon változtathatjuk az ablak méretét a labda mindig az ablak szélét érinteni fogja. Az if nélküli megoldásban ez nem lehetséges mert nem tudjuk ellenőrizni, hogy elért-e a labda az ablak szélére ezért egy előre meghatározott négyzetben mozoghat csak.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: clips.twitch.tv/csabi333

Megoldás forrása:

```
#include <stdio.h>

int main (void)
{
    int h=0;
    int n=0x01;

    do
    h++;
    while(n <<=1);

    printf("A szohossz ezen a gepen: %d bites\n",h);

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Megszámolhatjuk egy INT típusó változó hosszát ha azt binárisan 1-re rakjuk majd bitenként shifteljuk balra addig amíg az nulla nem lesz, így annyi lépéssel fogjuk azt lenullázni ahány bitből áll az INT típus.

2.6. Helló, Google!

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

Megoldás forrása: https://progpater.blog.hu/2011/02/13/bearazzuk_a_masodik_labort

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

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

    int i;

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

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

    int i;
```

```
    double osszeg=0;

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

void pagerank(double T[4][4]){
    double PR[4] = { 0.0, 0.0, 0.0, 0.0 }; //ebbe megy az eredmény
    double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0}; //ezzel szorzok

    int i, j;

    for(;;){
        // ide jön a mátrix művelet
        for (i=0; i<4; i++){
            PR[i]=0.0;
            for (j=0; j<4; j++){
                PR[i] = PR[i] + T[i][j]*PRv[j];
            }
        }
        if (tavolsag(PR,PRv,4) < 0.0000000001)
        break;

        // ide meg az átpakolás PR-ből PRv-be

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

    kiir (PR, 4);
}

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}
    };


    printf("\nAz eredeti mátrix értékeivel történő futás:\n");
    pagerank(L);


    printf("\n");

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A pagerank algoritmust Google találta fel arra hogy vele egyszerűen lehessen keresi az interneten. Egy web-oldal pagerank-ja egy szám ami egy aszerint kiszámolt érték hogy hány másik oldal mutat erre a bizonyos oldalra, mutatva ennek a fontosságát és információ értékét.

Az algoritmus megkap egy kapcsolat gráfot ami tartalmazza melyik oldal melyik másik oldalra mutat. A program a példában 4 oldallal dolgozik, ezt egy 4x4-es mátrixban határozzuk meg. A sor és oszlop metszésével kapjuk meg a kívánt pagerank értékeket. A mátrixszorzást az L és a PRv tömb között végezzük el, az eredmény lesz a négy oldal pagerank-ja.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

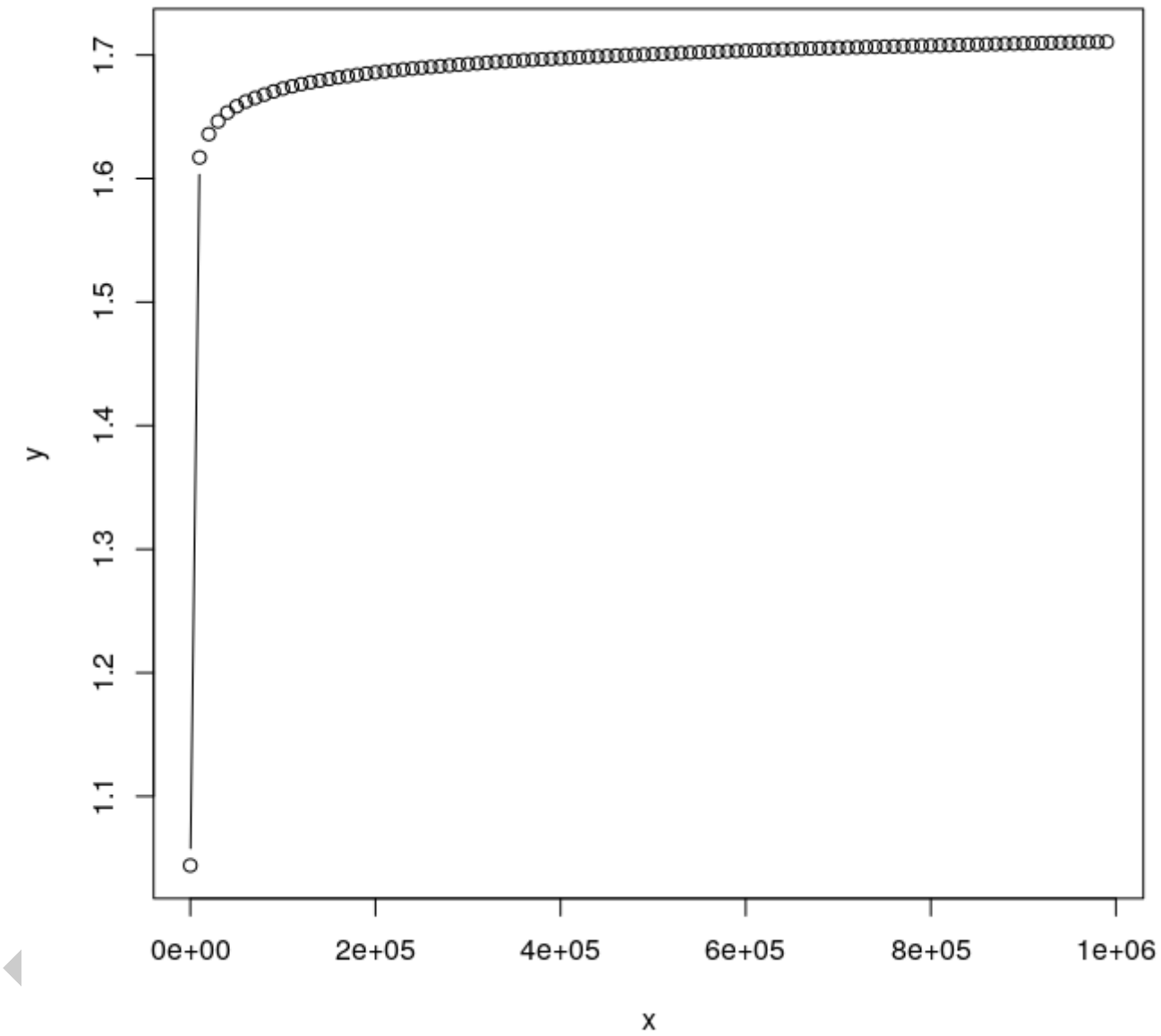
Brun konstansnak nevezzük azt a határt amihez az ikerprímszámok reciprokából képzett összeg konvergál. A prímszámokkal ellentétben az ikerprímek nem tartanak a végtelen felé.

```
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 függvény kér egy számot és a bementet továbbadja a 'primes' függvénynek. A megkapott számig minden prímet kiszámol és beletesz egy vektorba. A 'diff' nevű vektorban a felsorolt, egymás melleti prímek külömbségét tároljuk, amelyre a képlet szemléltetésénél szükségünk lesz. Ezen eredmények indexeit is el kell tárolni (idx függvény), de csak azokra van szükség amelyek egyenlők kettővel. Ezután amely párok különbsége 2 fel lesznek használva a Brun-tétel alapján úgy hogy vesszük ezek reciprokait és ezeket össze adjuk. A függvény rajzolásához meg kell adnunk egy x és egy y értéket:

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

A ’seq’ függvény mondja meg hogy milyen intervallumban vehet fel értéket az X és azon belül is milyen lépésközzel. Minden y-hoz hozzárendeljük az ’stp(x)’ értékét. A ’plot()’ függvényt pedig rajzolásra használják ami a jelen esetben arra kell, hogy az (x,y) értékeket egy grafikonon megjelenítse. A következő ábrát kapjuk futtatás után:



2.1. ábra. Brun-tétel

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

Tanulságok, tapasztalatok, magyarázat...

A Monty Hall probléma röviden megfogalmazva: Tegyük fel létezik egy olyan játék ahol a játékos elé három ajtó egyikének a választási lehetőségét teszik, ezen ajtók közül csak egy szolgál nyereménnyel. A játékosnak választania kell egy ajtót majd a műsorvezető kinyit egy üreset azok közül amit nem választott. Ezután feltevődi ka kérdés a játékosnak, hogy meg szeretné-e változtatni a döntését. Ilyenkor jön képbe a Monty Hall probléma, ugyanis a nyerés esélye megváltozik ha a játékos változtat a döntésén.

A játékos első választása esetén 1/3 eséllyel rendelkezik a nyertes ajto kiválasztásához. Ez nem változik ha az egyik ajtó kinyitása után nem változtat a döntésén a játékos, viszont ha igen akkor a játékos nyerési esélye is megváltozik 2/3-ra. Tehát a Monty Hall probléma szerint a váltás mondhatni megduplázza a nyerési esélyeinket. A következő program ezt hivatott bebizonyítani.

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

Meg kell határoznunk hány kísérletet hajtunk végre.A sample() függvényt használva szimuláljuk a véletlenszerűséget. A játékos tömbben a játékos tippje van. A műsorvezető tömbjét csak a játékos és a nyeremény függvényében tudjuk majd elkészíteni.

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

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

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

    }else{

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

    }

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

}
```

A for ciklust a kísérletek számáig léptetjük. Az if-ben vizsgáljuk azt hogy eltalálta-e a játékos a nyertes ajtót. A játékvezető csak az az ajtót válatszhatja ami nem nyertes és nem volt a játékos által választva.

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

A nemvaltoztatesnyer vektorba a which() függvénnyel feltöltjük azokat a poziciókat amelyekben a kisérket és a játékos tömmben is megegyeznek. Ezután létrehozuk a valtoztat vektort, ami olyan hosszú mint a kísérletek száma.

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

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

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

A for ciklus belseje azt a szerepet kapja hogy ha a játékos új ajtót választ megnézzük hol egyezik meg az indexe az értékeknek a kiserlet és a valtoztat tömbben és ezeket a helyeket betöltjük a valtoztatesnyer vektorba. Ezek utan nincs más dolgunk csak kiirni a valtoztatesnyer és a nemvaltoztatesnyer vektor értékeit. Itt bizonyosodunk meg arról hogy ha a játékos változtat tényleg nagyobb eséllyel nyer.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

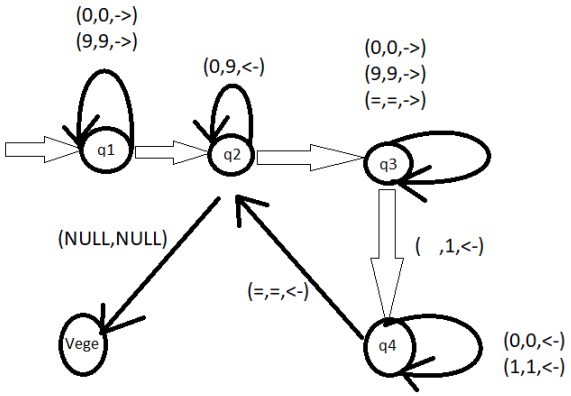
Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:

```
#include <stdio.h>

int main()
{
    int a;
    printf("Kerem a szamot:");
    scanf("%d",&a);
    printf("A szam unarisban:");

        while(a){
            a-=1;
            printf("1");
        }
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Decimálisból unárisba átváltani nem jelent mást mint a decimális szám értékével egyenlő számú 1-es számjegyet írunk ki a képernyőre. A Turing gép ezt úgy oldja meg hogy addig von le 1-et a decimális számból amíg az 0 nem lesz.



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

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

```
1.
S->abc;S->aXbc;Xb->bX;Xc->Ybcc;bY->Yb;aY->aaX;aY->aa

Lépések:
S->aXbc
Xb->bX
Xc->Ybcc
bY->Yb
aY->aa
```

```
2.
A->aAB;A->aC;CB->bCc;cB->Bc;C->bc

Lépések:
A (A->aAB)
aAB (A->aC)
aaCB (CB->bCc)
aabCc (C->bc)
aabbcc
```

Tanulságok, tapasztalatok, magyarázat... A képzett nyelvek a szabályos stringekből konstansokat képeznek.

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 forrása:

```
#include <stdio.h>

int main(){

for(int i =0; i<5;i++)
printf("teszt");
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Többek közt például a for ciklusban régi fordítok nem engedték meg hogy új változót inicializáljunk, ezt új fordítókkal már megtehetjük. A "gcc 3_3.c -std=c89" paranccsal a forditü egy régebbi szabály szerint probál fordítani de egyből észrevesszük hogy hibát fog adni:

```
3_3.c:5:1: error: ‘for’ loop initial declarations are only allowed in C99 ↩
   or C11 mode
 for(int i =0; i<5;i++)printf("teszt");
 ^~~
```

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 forrása:

```
            %{
#include <stdio.h>
int realnumbers = 0;
%}

digit   [0-9]

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

Tanulságok, tapasztalatok, magyarázat...

Egy szűrőt használva if-ként ellenőrizzük hogy tartalmaz-e a bement olyan rész-karakterláncot amely csak számokból áll, ezeket a megtalált karakterláncrészleteket majd kiemeljuk és ezeket kapjuk eredményként. A % jelek elkülönitésre szolgálnak, felosztva kódot harom részre. A első a header file-okat tartalmazza, a második részbe vannak a szabályok, az ultsó rész hívja meg a lexikális elemzőt.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása:

```
      %{
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <ctype.h>
#define L337SIZE (sizeof l337d1c7 / sizeof (struct cipher))
struct cipher {
  char c;
  char *b[5];
} l337d1c7 [] = {
{'a',"/-\\"},
{'b',"|3"},
{'c',"<"},
{'d',"|]"},
{'e',"3"},
{'f',"|="},
{'g',"[+"},
{'h',"|-|"},
{'i',"!"},
{'j',"_/"},
{'k',"1<"},
{'l',"1"},
{'m',"44"},
{'n',"/\\/"},
{'o',"()"},
{'p',"/o"},
{'q',"9"},
{'r',"12"},
{'s',"$"},
{'t',"7"},
{'u',"|_|"},
{'v',"\\/"},
{'w',"VV"},
{'x',")("},
{'y',"y"},
{'z',"z"},
```

```
  {'0',"D"},
  {'1',"I"},
  {'2',"Z"},
  {'3',"E"},
  {'4',"A"},
  {'5',"S"},
  {'6',"b"},
  {'7',"7"},
  {'8',"X"},
  {'9',"g"}
  };
%}
%%
.    {
       int found = 0;
       for(int i=0; i<L337SIZE; ++i)
       {
         if(l337d1c7[i].c == tolower(*yytext))
         {
             printf("%s", *l337d1c7[i].b);
           found = 1;
           break;
         }
       }
       if(!found)
          printf("%c", *yytext);
     }
%%
int
main()
{
  yylex();
  return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A leet cipher kódolás csak annyit jelent hogy a betűket kicseréljük egy hozzá hasonló karakterrel vagy karakterlánccal. A forráskódban jól kiolvasható hogy mit mivel cserélunk ki. A feladat az volt hogy ezt a programot a lex segitségével oldjuk meg, ami azt jelentette hogy a bemenetet a lex szintaxisával kezeljük minden egyes karakter bevitele után.

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 nem lett kezelve, akkor ezután se legyen kezelve, ha pedig fel lett dolgozva akkor a jelkezelő függvény is vegye figyelembe.

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

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

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

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

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

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

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

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

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

Megoldás:

i. Minden jel figyelmen kívül lesz hagyva kivéve a ctrl+z "SIGTSTP" signal-t amit semmilyen program nem irhat felül.

ii. Egy for ciklus amely 5 ismétlést fog végrehajtani.

iii. Egy for ciklus amely 5 ismétlést fog végrehajtani. Másként növeli a 'i' változó értékét de ebben az esetben ez nem fogja befolyásolni a ciklust.

iv. Ez a ciklus beletenne 0-tol 4-ig a számokat egy tömbbe, a probléma az hogy nem ismerjük a parancsok végrehajtásának sorrendjét.

v. Értékadó utasítát használunk összehasonlítás helyett, ez okozza itt a problémát.

vi. A függvénynek adott két int-nek a kiértékelési sorrendje nem tisztázott, ez okozza a problémát.

vii. Hiba nélkül kiirjuk az 'a' változó, függvénnyel módosított és a kezdő értékét is.

viii. Nem tudjuk eldönteni mit fog kiírni, kétszer a függvény által módosított értéket vagy egyszer azt és egyszer módosulatlan értéket.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

A formulák értelmezéséhez szükség van a külömböző függvények értelmének tudásához.

Az univerzális kvantort a forall jelöli; Az exist az egzisztenciális kvantort; A wedge az implikációt; A supset pedig a konjunkciót fogja jelölni.

Most, hogy ezeket tudjuk, már csak le kell fordítani a emberi nyelvre a fentebb látható formulákat.

Megoldás:

1. Minden x esetén van olyan y ami nagyobb x-nél és y prím.

2. Minden x esetén van olyan y ami nagyobb x-nél és y prím, és ha y prím, akkor annál kettővel nagyobb szám is prím.

3. Létezik olyan y, aminél minden x kisebb és prím.

4. Létezik olyan y, aminél minden x nagyobb és x nem prím.

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

Megoldás forrása:

```
#include <stdio.h>

int main(){

int a;                          //'a' nevű egész típusú változó
int *b = &a;                    //mutató amiben az előző változónk ↩
   memóriacíme van
int &r=a;                       //'r' nevezetű mutató amiben az 'a' változó ↩
    értéke van
int c[5];                       //Öt darab egész számot tartalmazó tömb
int (&tr)[5]=c;                 //Öt elemű tömbre mutató pointer ami az el ↩
   őző tömbre mutat
int *d[5];                      //Öt elemű tömb, ami egyben mutató is
int *h();                       //Függvény ami mutatót add vissza
int *(*l) ();                   //Egész számra mutató pointert visszaadó ↩
   függvényre mutató pointer
int (*v (int c)) (int a,int b); //Egésszel visszatérő és két egészet kapó ↩
   függvényre mutató pointer ami visszaadja a függvényre mutatót
int (*(*z) (int)) (int,int);    //Egészet visszaadó és két egészet kapó ↩
   függvényre mutató pointert visszaadó, egészet kapó függvény
    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A g++ forditot kell hasznaljuk a referencia miatt.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó és forrása:https://youtu.be/1MRTuKwRsB0

Tanulságok, tapasztalatok, magyarázat...

A háromszögmátrixok valójában négyzetes mátrixok, melyeknek sorainak és oszlopainak száma megegyezik viszont csak a főátlójuk felett vagy csak a főátlójuk alatt tartalmaznak értékeket, tehát beszélhetünk alsó vagy felső háromszögmátrixról.

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

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

Az int típusú nr nevű változó tartalmazza hogy milyen méretű a mátrixunk. Még itt deklaráljuk a tm nevű pointert is.

```
printf("%p\n", &tm);

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

printf("%p\n", tm);
```

Kiíratjuk tm memóriacímét majd a malloc függvény használatával lefoglalunk 5x8 byte helyet. Ha ez a művelet sikertelen akkor az if miatt kilép a program mert nem tud tovább lépni, ha nem mutat sehova a mutatónk.

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

}
printf("%p\n", tm[0]);
```

Ahhoz hogy egy háromszögmátrixnak foglaljunk helyet a memóriában még minden soron végig megyünk és rendre 1 2 3 4 és 5-ször 8 byteot foglalunk le a for ciklussal.

```
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 ("%d, ", tm[i][j]);
    printf ("\n");
}
```

Feltöltjük a háromszögmátrixunkat amely egy alsó háromszögmátrix, majd a mátrix elemeit kiirjuk.

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

free (tm);

return 0;
}
```

A program utolsó részében fel kell szabadítanunk a program által használt memóriát.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: :https://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia

Az EXOR titkosító a logikai vagyra épül, a logikai XOR műveletre. A XOR művelete bitenként összehasonlítja a két operandust, ha a két összehasonlított bit megegyezik akkor nullát, különben egyet ad vissza. Az EXOR titkosító esetén a két operandus a titkosítandó szöveg és a titkosító kulcs. Az lenne az ideális ha két operandus mérete megegyezne, evvel szinte feltörhetetlen kódot kapnánk, ha a kulcs rövidebb mint a titkosítandó szöveg akkor a kulcs ismétlődni fog ami biztonsági hibát eredményez, kevésbé lesz megbízható a titkosítás.

```
#define MAX_KULCS 100
#define BUFFER_MERET 256
```

A kulcs és a buffer méretét konstansba tároljuk ezek módosíthatatlanok.

```
int
main (int argc, char **argv)

char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];

int kulcs_index = 0;
int olvasott_bajtok = 0;

int kulcs_meret = strlen (argv[1]);
strncpy (kulcs, argv[1], MAX_KULCS);
```

```
while ((olvasott_bajtok = read (0, (void *) buffer,  ↩
   BUFFER_MERET)))
{

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

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

    }

write (1, buffer, olvasott_bajtok);

}
```

A while ciklus csak a megadott mennyiségű bájtok beolvasása után áll le csak. Végigmegyünk bufferben eltárolt karaktereken és össze xorozzuk a kulcs megfelelő elemével, majd növeljük a kulcs_index-et 1-el ameddig el nem érjük a kulcs_meret-et, ha elértük ezt akkor lenullázzuk. A végén kiírjuk a buffer tartalmát.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása: :https://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia

```
public class ExorTitkosító {

public ExorTitkosító(String kulcsSzöveg,
        java.io.InputStream bejövőCsatorna,
        java.io.OutputStream kimenőCsatorna)
```

```
            throws java.io.IOException {

        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;

        while((olvasottBájtok =
                bejövőCsatorna.read(buffer)) != -1) {

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

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

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

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

        } catch(java.io.IOException e) {
            e.printStackTrace();}}
```

A feladat megegyezik az első feladattal csak a megoldás módszere más. A kizáró vagyos titkosítás során a titkosítandó szöveg bájtjait lefedjük a titkosító kulcs bájtjaival és az egymás alá eső biteken végrehajtunk egy kizáró vagy műveletet. A kizáró vagy 1 értéket ad, ha a két bit különböző és 0 értéket, ha megegyező. A külső while ciklus buffer tömbönként addig olvassa a bemenetet, amíg csak tudja. A belső for ciklusban helyezzük rá a kulcsot a beolvasott bájtokra a kulcsIndex változó segítségével, majd végrehajtjuk a kizáró vagy műveletet, az eredmény a buffer tömbben keletkezik, amit végül a kimenetre írunk.

4.4. C EXOR törő

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

Megoldás forrása: :https://progpater.blog.hu/2011/02/15/felvetelt_hirdet_a_cia

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
#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 *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 <= '9'; ++ii)
        for (int ji = '0'; ji <= '9'; ++ji)
        for (int ki = '0'; ki <= '9'; ++ki)
        for (int li = '0'; li <= '9'; ++li)
        for (int mi = '0'; mi <= '9'; ++mi)
            for (int ni = '0'; ni <= '9'; ++ni)
            for (int oi = '0'; oi <= '9'; ++oi)
            for (int pi = '0'; pi <= '9'; ++pi)
            {
                kulcs[0] = ii;
                kulcs[1] = ji;
                kulcs[2] = ki;
                kulcs[3] = li;
                kulcs[4] = mi;
                kulcs[5] = ni;
                kulcs[6] = oi;
                kulcs[7] = pi;

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

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

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

                        return 0;
                }
```

A feladat megoldásához használt módszer a 'Brute Force' módszer ami nem jelent mást mint az összes lehetőség kipróbálása egészen addig amíg megoldást nem találunk. Ez a lehető leglassúbb és legtöbb erőforrást igénylő módszer mégis jelen esetünkben ez a legkézenfekvőbb.

A program elején a kulcs méretét adjuk meg, ezt fontos meghatározni mert kiküszöböljük evvel ennek hosszának is a kitalálását. A exor függvény ugyan azt csinálja mint a tikosító mert ha valamit kétszer exorozunk akkor az az eredeti értékét veszi fel. Előállítjuk az összes lehetséges kulcsot és ennek a függvénynek adjuk át ezeket egyenként és mindegyikről eldöntjük hogy feltörte-e a titkos szöveget.

Tutoralt: Ranyhoczki Mariann

4.5. Neurális OR, AND és EXOR kapu

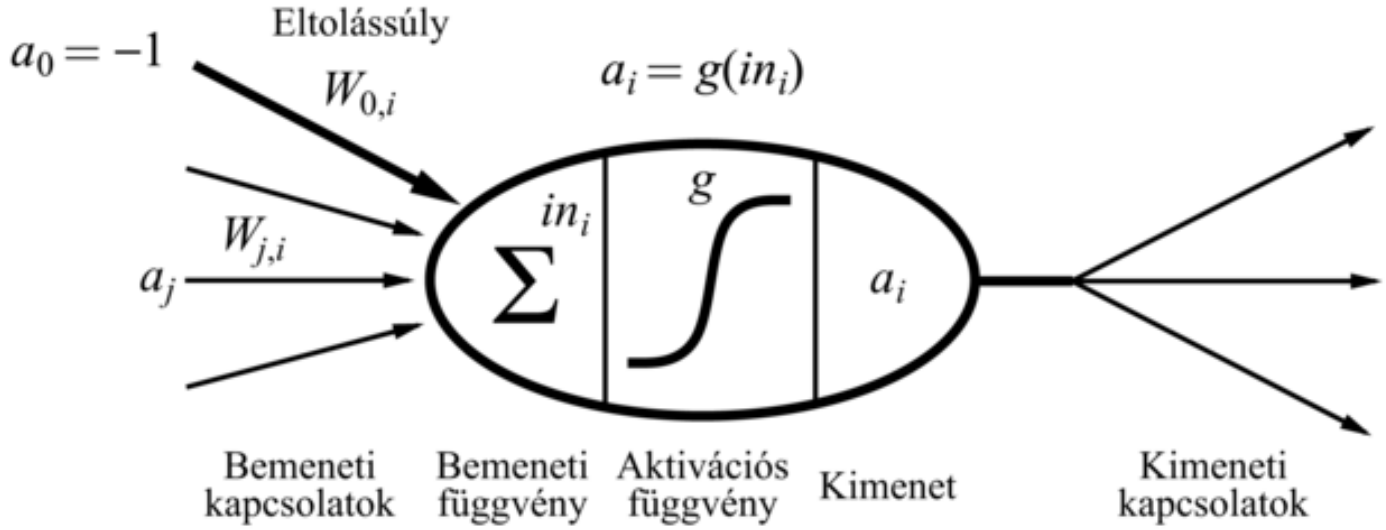
R

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

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

Tanulságok, tapasztalatok, magyarázat...

Ebben a feladatban újra visszatérünk a Monty Hall problémánál megismert R nyelvhez. Segítségével neurális hálózatot fogunk létrehozni, mely képes "tanulni", és megközelíteni az általunk megadott megfelelő értékeket. A hálózat a nevét a neuronról kapta, mely agyunk egy sejtje. Feladata az elektromos jelek összegyűjtése, feldolgozás és szétterjesztése. Az a feltételezés, hogy az agyunk információfeldolgozási képességét ezen sejtek hálózata adja. Éppen emiatt a mesterséges intelligencia kutatások során ennek a szimulálást tűzték ki célul. A neuron matematikai modeljét McCulloch és Pitts alkotta meg 1943-ban. Ezt mutatja a következő ábra:



4.1. ábra. Neuron (Forrás http://project.mit.bme.hu/mi_almanach/books/aima/ch20s05)

A lényeg, hogy a neuron akkor fog tüzelni, ha a bemenetek súlyozott összege meghaladnak egy küszöböt. Az aktivációs függvény adja meg a kimenet értékét.

Ezt a modellt fogjuk implementálni egy R programba. Az R-hez hozzá kell adnunk a neuralnetwork libraryt. Minta feladatot kell megadnunk a programnak amely alapján tanulni fog, ekkor a program előre tudja mi lesz a bemenet és a kimenet. Hogy ezeket az értékeket kapjuk a program magától választ súlyokat. A program logikai kapukat vizsgál meg igazságuk szerint, ezek a kapuk a logikai 'és' 'vagy' 'kizáró vagy' műveletek. Ha növeljük a neuronok számát akkor pontosabb értéket kapunk továbbá kevesebb lépés szükséges ahhoz hogy eredményt kapjunk és több mintát készít.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: https://github.com/nbatfai/samu/blob/master/ql.hpp

Tanulságok, tapasztalatok, magyarázat...

A perceptron egy olyan algoritmus ami megtanítja a számítógépnek a bináris osztályozásra. Három részből áll, az első bemeneti jeleket fogad ami nagyrészt az igen és a nem. A második része az úgynezvezett asszociatív cella, összegzik a beérkező jeleket és csatlakozik a másik két cellához. A harmadik a döntő cella ami a percetronok kimenetéért felelős.

5. fejezet

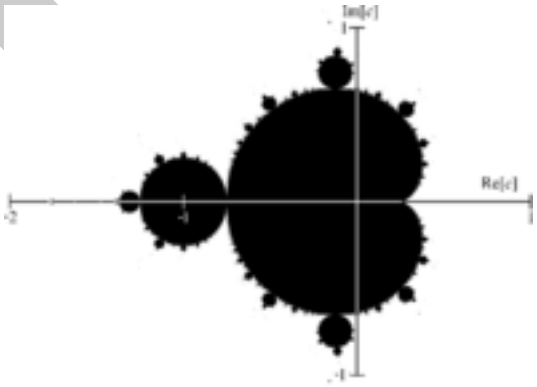
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás forrása: :https://sourceforge.net/p/udprog/code/ci/master/tree/source/binom/Batfai-Barki/frak/

```
#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;
}
```



5.1. ábra. Mandebrot Kimenet

Tanulságok, tapasztalatok, magyarázat...

A matematikában a Mandelbrot-halmaz azon c komplex számokból áll (a „komplex számsík” azon pontjainak mértani helye, halmaza), melyekre az alábbi (komplex szám értékű) Z(n) rekurzív sorozat:

```
                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;
                }
```

nem tart végtelenbe, azaz abszolút értékben (hosszára nézve) korlátos. A Mandelbrot-halmazt a komplex számsíkon ábrázolva, egy nevezetes (és hasonnevű) fraktálalakzat adódik. Jelen esetünkben a mandebrot halmaz ábrázolásához szükségünk lesz ’png++/png.hpp’ header file-ra továbbá az evvel járó szintaxisok használatára. A halmazt egy 500x500 mátrixban tároljuk majd ezt bejárva generáljuk a képünket róla a GeneratePNG() void segítségével. A matematikai háttér miatt létre kell hoznunk egy komplex struktúrát amely a tömb elemei lesznek.

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

Az előző feladatot fogjuk megoldani újra csak annyi különbséggel, hogy most a std::complex könyvtárat fogjuk használni. Ennek köszönhetően a komplex számokat nem két változóban tároljuk.

struct Komplex{ double re, im; };

E helyett a struktúra helyett std::complexdouble Z (reZ, imZ); szintaxissal majd ennek a változónak használatával az előző programunk csak annyiban változik hogy máshogy hivatkozunk a komplex számokat tartalmozó változókra.

5.3. Biomorfok

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

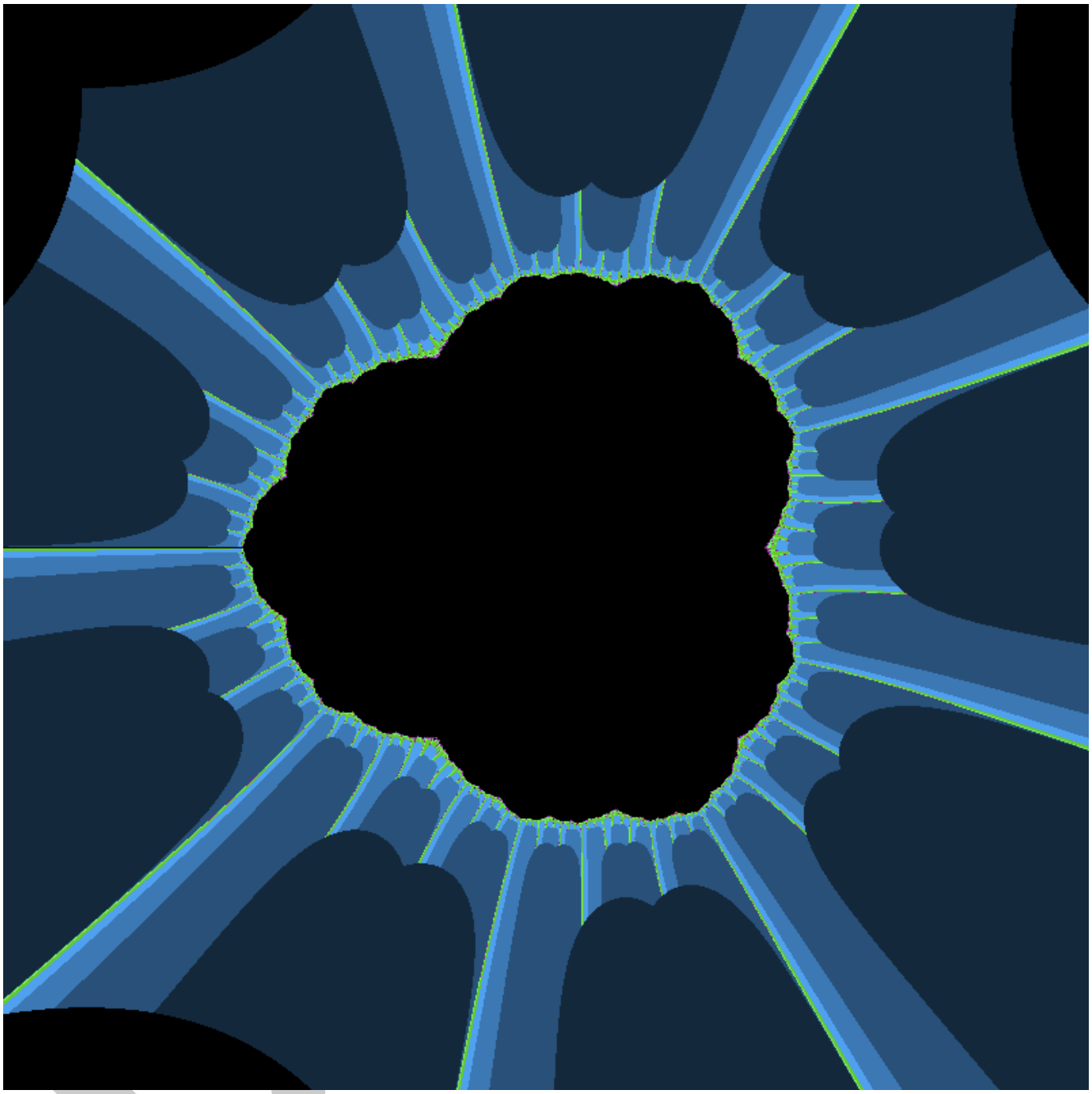
Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/Biomorf

Megoldás forrása:

```
#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;

}

              }
```



5.2. ábra. Biomorf Kimenet

Tanulságok, tapasztalatok, magyarázat...

A Mandelbrot halmazban megtalálható az összes Júlia halmaz. A Júlia halmaz esetén a C konstans, és a rácsot a z-vel járjuk be. A Mandelbrot halamzban ez a c konstans helyett változóként szerepel melyhez z értékeket rendelünk. Szóval mindig új Júlia halmazt számolunk ki vele. Mikor egy Bug-os programmal Clifford Pickover rátalált a biomorfokra, akkor azt hitte természeti törvényre bukkant.

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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

A CUDA egy többszálas, párhuzamos feldolgozási platform és API, amit az nvidia grafikus kártyák használnak. Ez a GPU-t használja a CPU helyett számításokra.

Egy GPU esetében sokszorosan hatékonyabb párhuzamosítást érhetünk el a CPU-val szemben. Elérhetjük hogy a kép készítése során minden pixel egyszerre készüljön el, mert minden pixelnek megfelel egy szál, miközben a CPU-s verzió csak egy szállal rendelkezik.

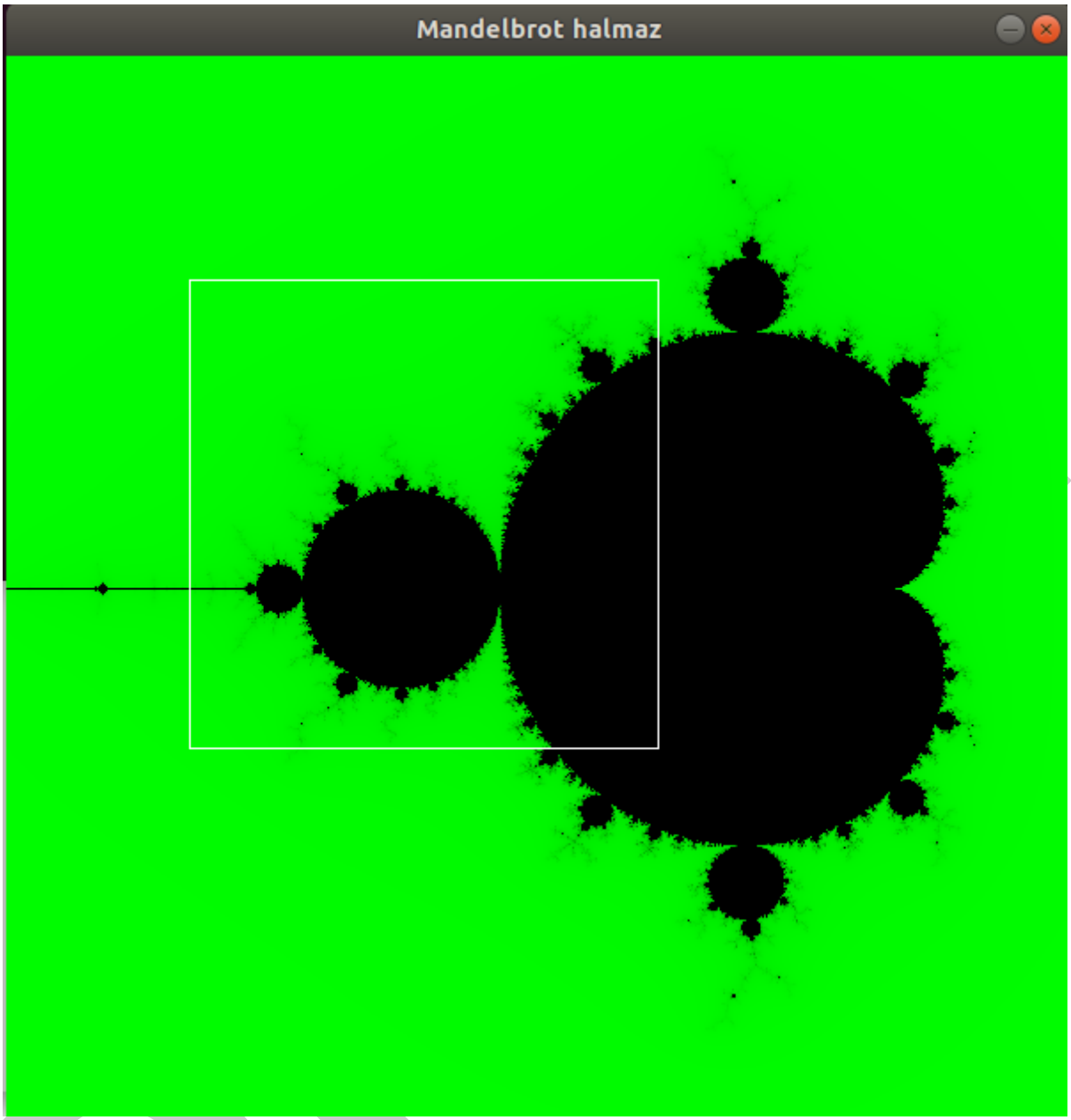
5.5. Mandelbrot nagyító és utazó C++ nyelven

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

Megoldás forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/binom/Batfai-Barki/frak/

Használata: Telepíteni: sudo apt-get install libqt4-dev

A program a QT GUI-t használja, ennek segítségével tudjuk elkészíteni a programunkat. Ez a GUI az egyik legelterjedtebb garfikus interfésze a C++-nak. Ezt telepítenünk kell előre. Fordítás: A fájloknak egy mappában kell lennie. A mappában futtatni kell a qmake frak.pro parancsot. Ez létre fog hozni egy Make fájlt. ezután ki adjuk a 'make' parancsot, mely létrehoz egy bináris fájlt. Ezt pedig a szokásos módon futtatjuk.



5.3. ábra. Mandebrot nagyító

5.6. Mandelbrot nagyító és utazó Java nyelven

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás videó:

Megoldás forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf

A feladat az objektum orientált programozásról szól. A feladatbana a polártranszformációt használjuk random számok generálásához.

```
class PolarGen {

public:

PolarGen(); //konstruktor
~PolarGen(){} //destruktor
double kovetkezo(); //random lekérés

private:

bool nincsTarolt;
double tarolt; //random értéke

};
```

A 'public' rész elemei elérhetőek a class-on kívül amíg a 'private' rész elemi csak azon belül. A konstruktor dolga, hogy létrehozza a PolarGen típusú objektumot. A konstruktor a program futása során csak egyszer hajtódik végre hiába a 'public' része. A destructor csak a program futása végén lesz használva, szintén csak egyszer. Ha class-on belül foglalunk le tárhelyet akkor feltétlen szükséges a destructor használata a lefutás után.

```
#include "polargen.h"
```

```
double PolarGen::kovetkezo(){
    if (nincsTarolt){
        double u1, u2, v1, v2, w;
        do{
            u1 = std::rand() / (RAND_MAX + 1.0);
            u2 = std::rand() / (RAND_MAX + 1.0);
            v1 = 2*u1-1;
            v2 = 2*u2-1;
            w = v1*v1+v2*v2;
        }
        while (w > 1);

double r = std::sqrt ((-2*std::log (w)) / w);
tarolt = r*v2;
nincsTarolt = !nincsTarolt;
return r*v1;
}
else{
    nincsTarolt = !nincsTarolt;
    return tarolt;
    }

}
```

A class-ban szereplő 'kovetkezo()' egy algoritmust tartalmazó függvény, ami generál két random számot ha még nem rendelkezünk vele, az egyiket vissza adja a másikat eltárolja. A main függvény tartalma szemlélteti a class használatát abban az esetben ha a feladat egyszerűen az, hogy standard inputra akarunk egy random számot kiíratni ennek a classnak a használatával.

```
int main()
{

    PolarGen rnd;

    std::cout << rnd.kovetkezo() << std::endl; //random szám ↩
       generálása

}
```

Megoldás forrása: https://arato.inf.unideb.hu/batfai.norbert/UDPROG/deprecated/Prog1_5.pdf

A java forrás a következő:

```
    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();
                    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());
            }
        }
    }
```

Java-ban az egész program egy class részeként szerepel. Minden elemről külön meg kell határozni hogy a public vagy a private része.

6.2. LZW

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

Megoldás forrása: https://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg

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

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

typedef struct binfa
{
  int ertek;
  struct binfa *bal_nulla;
  struct binfa *jobb_egy;

} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
  BINFA_PTR p;

  if ((p = (BINFA_PTR) malloc (sizeof (BINFA))) == NULL)
    {
      perror ("memoria");
      exit (EXIT_FAILURE);
    }
  return p;
}

extern void szabadit (BINFA_PTR elem);


int
main (int argc, char **argv)
{
  char b;

  BINFA_PTR gyoker = uj_elem ();
  gyoker->ertek = '/';
  gyoker->bal_nulla = gyoker->jobb_egy = NULL;
  BINFA_PTR fa = gyoker;

  while (read (0, (void *) &b, 1))
    {
//      write (1, &b, 1);
      if (b == '0')
  {
    if (fa->bal_nulla == NULL)
      {
        fa->bal_nulla = uj_elem ();
        fa->bal_nulla->ertek = 0;
        fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
        fa = gyoker;
      }
    else
```

```
      {
        fa = fa->bal_nulla;
      }
  }
      else
  {
    if (fa->jobb_egy == NULL)
      {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
      }
    else
      {
        fa = fa->jobb_egy;
      }
  }
    }

  printf ("\n");
  kiir (gyoker);
  szabadit (gyoker);
}




void
szabadit (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      szabadit (elem->jobb_egy);
      szabadit (elem->bal_nulla);
      free (elem);
    }
}
```

Az LZW algoritmus egy bináris fát épít a bemeneti egyesekből es nullásokból. Ezt úgy hajtja végre, hogy ellenőrizzük hogy van-e már 0-ás vagy 1-es gyereke az éppen soron következő elemnek, ha nincs akkor létrehozunk egyet és visszaugrunk a gyökérre, ha van akkor a 0-ás vagy 1-es gyerekre lépünk ezt addig ismételjük amig nem jutunk el oda hogy már nincs több gyerek, ekkor lértehozunk egyet ismét.

A megvalósításhoz létrehozzuk a szükséges struktúrát amiben egy érték és két mutató szerepel. Egy fa elemének nemcsak a saját értékét kell tartalmaznia hanem a rákövetkező elem vagy elemek elérhetőségeit kivéve ha ez az elem a fa szélén van.

Az 'ujelem()' függvény akkor lesz használva mikor egy új elemet hozunk létre amelyek BINFA típusúak. A függvény visszaszolgáltatja az elemre mutató pointert.

6.3. Fabejárás

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

Tutoriált Ignéczi Tibor

Pre_order bejárás Megoldás forrása:

```
void
kiir_preorder (BINFA_PTR elem)
{
  if (elem !=NULL)
  {
    ++melyseg;
    if (melyseg > max_melyseg){ max_ max_melyseg=melyseg;}

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

Post_order bejárás Megoldás forrása:

```
void
kiir_postorder (BINFA_PTR elem)
{
  if (elem !=NULL){

    melyseg++;
    if (melyseg > max_melyseg){max_melyseg=melyseg;}
    kiir_postorder(elem->bal_nulla);
    kiir_postorder(elem->jobb_egy);

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

    melyseg--;


  }
}
```

A postorder és preorder fabejárások között az a külömbség,hogy az eljárásban hogyan hívjuk meg rekurzívan a kiír eljárást. A preoder kiírás algoritmusa először a fa gyökeréhez közeli elemeket írja ki megfelelő

sorrendben majd onnan indul lefelé, az postorder pedig ennek az ellentettje, a fát a gyökérhez legtávolabb lévő elemmel kezdi bejárni, tehát az eredményt nézve két teljesen eltérő ereményt kapunk ugyanazon fa bejárásával.

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 forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/vedes/elso/z3a7.cpp

Az előző bináris fával ellentétben a gyökér nem egy pointer hanem egy '/' karaktert tartalmazó objektum.A védett tagok között lesz a csomopont gyoker, a fa pedig pointer lesz ami az éppen épülő fa azon csomópontjára mutat amit éppen az LZW fa építő algoritmusa határoz meg.

```
void operator<< (char b)
{
    if (b == '0')
    {

        if (!fa->nullasGyermek ())
        {

            Csomopont *uj = new Csomopont ('0');
            fa->ujNullasGyermek (uj);
            fa = &gyoker;
        }
        else
        {

            fa = fa->nullasGyermek ();
        }
    }

    else
    {
        if (!fa->egyesGyermek ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyesGyermek (uj);
            fa = &gyoker;
        }
        else
        {
            fa = fa->egyesGyermek ();
        }
    }
}
```

Túlterheljük az operátort:

```
void operator<< (char b)
```

C++ ban lehetőségünk van túlterhelni a már meglévő operátorokat, a túlterhelés azt jelenti hogy definiáljuk az operátor számára hogy hogyan működjön, milyen utasításokat hajtson végre, ha például a saját magunk által megalkotott típusokra használjuk.

```
    void kiir (void)
    {


        melyseg = 0;

        kiir (&gyoker, std::cout);
    }
```

A kiír függvény az ebben az esetben az alapértelmezett outputra írja ki a fát.

6.5. Mutató a gyökér

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

Az előző feladatban részletesen taglaltuk, hogy hogyan is épül fel a C++-os LZW algoritmust alkalmazó program, most ezt kell módosítani, mivel az előzőben a gyoker tagja volt az osztálynak. Ebben a feladatban pointerre fogjuk átírni, ami nem is annyira nehéz feladat, csak néhány dologra oda kell figyelni.

```
                Csomopont *gyoker;
```

Ha most próbálkozunk a fordítással akkor rengeteg hibát fogunk kapni, ezeket kell kijavítani a következő lépésekben. A konstruktor módosítása következik-

```
LZWBinFa ()
    {
        gyoker = new Csomopont ('/');
        fa = gyoker;
    }
```

Mivel itt a gyoker mutatót egy újonnan lefoglalt tárterületre ráálítjuk ezt majd fel is kell szabadítani. Így néz kia destruktora-

```
~LZWBinFa ()
    {
        szabadit (gyoker->egyesGyermek ());
        szabadit (gyoker->nullasGyermek ());
        delete(gyoker);
    }
```

Meg kell hívnunk a szabadit függvényt a a gyoker nullás és egyes elemére is mert mostmár a gyökér is rendelkezik ezekkel. A gyoker mutató által mutatott területet pedig a delete() függvénnyel szabadítjuk fel.

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása: https://sourceforge.net/p/udprog/code/ci/master/tree/source/binom/Batfai-Barki/vedes/-z3a9.cpp

A megoldáshoz azt as programot vesszük alapul amelyben a gyökér tag volt. Ezt vesszük alapul és pár módosítás után már egy mozgató konstruktorral rendelkezünk.

Létre kell hoznunk a mozgató kontruktort és az értékadást. A paraméterként átadott fa gyökerének az elemeit átküldjük az üres fának majd az eredeti fát úgy töröljük hogy kinullázzuk.

```
    LZWBinFa binFa2 = std::move(binFa);

    kiFile << binFa2;
    kiFile << "depth = " << binFa2.getMelyseg () << std::endl;
    kiFile << "mean = " << binFa2.getAtlag () << std::endl;
    kiFile << "var = " << binFa2.getSzoras () << std::endl;
```

Még az egyes elemeket kell átmozgatnunk a move függvénnyel, utána kiírjuk a fát.

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: gitlab.com/nbatfai/bhax

A program célja az, hogy szimulálja a hangyák feromonokkal való tájékozódását. Ezt úgy éri el hogy az ablakot felosztja egy négyzetrácsra. A négyzetrácsban a hangyák megkeresik a hozzájuk azt a legközelebb lévő szomszédot amelyiknek a legerősebb a feromon szintje. A négyzetek feromon szintjei folyamatosan csökkennek, kivéve mikor egy hangya belép oda, mert akkor megnő. Ezeket az értékeket a parancssori argumentumokkal adjuk meg.

A forráskód azon részeire érdemes kitérni amelyek a feladat megoldásához egyediek. Az első ilyen az 'Ant' osztály ameleyet az 'ant.h' header tartalmazza itt csak annyi van hogy a hangyák koordinátáit meghatározzuk az x és y koordináták segítségével, továbbá az irányukat is itt számoljuk random értékkel, amit a 'dir' változó tárol.

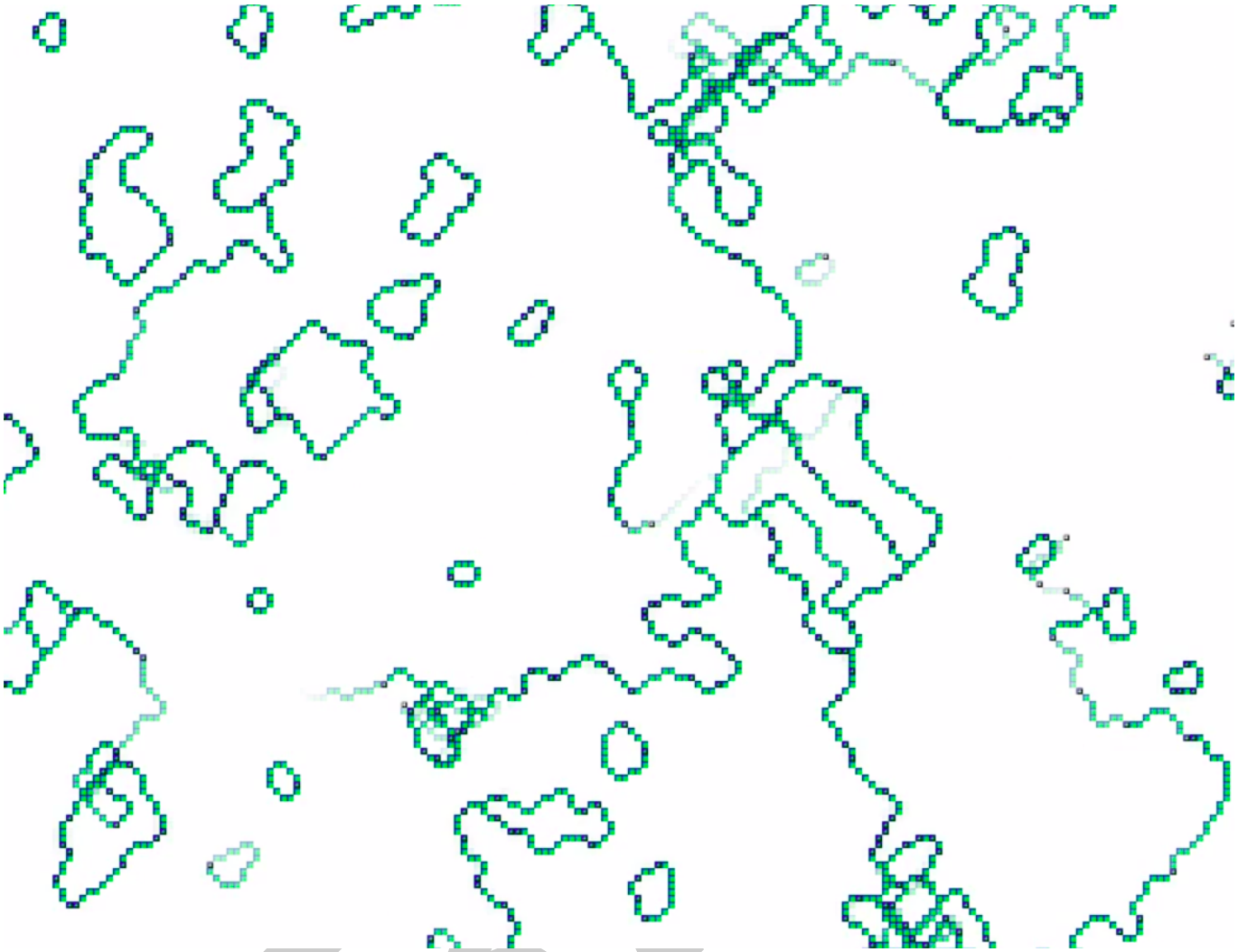
```
class Ant
{
public:
    int x;
    int y;
    int dir;

    Ant(int x, int y): x(x), y(y) {

        dir = qrand() % 8;

    }

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

A Qt funkciói használatához szükségesek a további header file-ok. A 'QMainWindow' például arra szolgál hogy megnyit egy ablakot amiben a hangyaszimuláció végbemegy.

```
#include <QThread>
#include "ant.h"

class AntThread : public QThread
{
    Q_OBJECT

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

    ~AntThread();

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

    void pause()
    {
        paused = !paused;
    }

    bool isRunnung()
    {
        return running;
    }

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

```
                int delay;

                void timeDevel();

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

            signals:
                void step ( const int &);

            };
```

A Qthread osztály szükséges a program szálai kezeléséhez. Az AntThread osztály komstruktora megkapja az AntWIn osztályban megadott bemeneti értékeket, majd elkezdi mozgatni a hangyákat. A feromonszintek és a cellák sűrűségének függvényében kiszámolja a hangyák irányát és ezeket az számításokat visszaszolgátatlja az AntWin osztálynak.



7.1. ábra. Hangyaszimuláció

7.2. Java életjáték

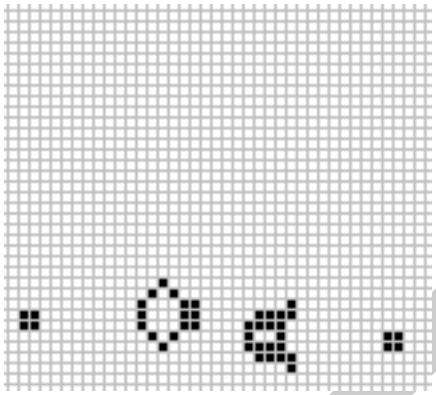
Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

Megoldás videó:

Megoldás forrása: tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/

John Horton Conway angol matematikus nevéhez fűződik az életjáték. A játékoznak kizárólag csak a kiinduló pontot kell meghatároznia. Matematikai meghatározásában a neve ennek a játéknak a sejtautomata melyben a cellákat a négyzetrácsok és a sejteket pedig korongok jelképezik. Minden sejtnek nyolc szomszédos cellája van, ha egy generációban egy sejtnek van kettő vagy három szomszédja ami szintén sejt akkor az a következő generációban is élni fog, minden más esetben kihal. Ha egy üres cellának pontosan három élő sejtüszomszédja van akkor ott új sejt keletkezik. Ezek a szabályok az időFejlődés() függvényben vannak jelen. Két időiterációt két rács fogja tárolni a t_n és a t_n+1. Ahol az első az a jelenlegi állapot a második az ebből keletkezett következő lépést tárolja. A sejttérbe időnként 'siklóágyúkat' helyezünk ezek 'élölényeket'

jelképeznek. Az addKeyListener, addMouseListener, addMouseMotionListener függvényekben meghatározzuk azokat a billentyűlenyomásokat és egéreseményeket, amelyekkel tudjuk a futó Életjáték-program eseményeinek alakulását befolyásolni.



7.2. ábra. Sejtautomata

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

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

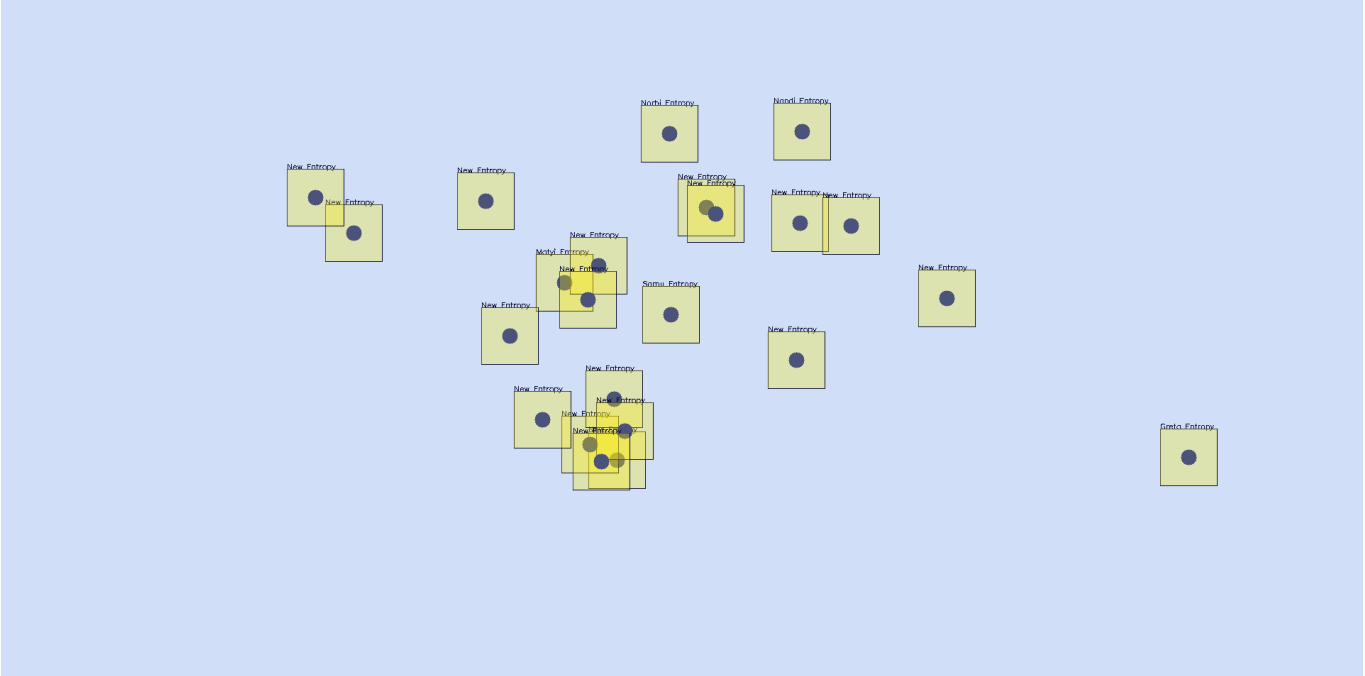
Az előző feladatot fogjuk megnézni C++ környezetben, méghozzá ismét a Qt keretrendszer segítségével. A Java és a C++ hasonlóságainak köszönhetően nem az előző feladat forráskódját vehetjük támaszpontnak. A forrás lefordításához a fájlokat egy mappába kell helyezni és ebben a mappában futtatni kell a 'qmake Tool' parancsot, miután ez létrehoz egy eletjatek.pro és egy Makefile nevű fájlt, A Makefile használatával kaounk egy futtatható állományt, amit a terminálon keresztül indítunk el.

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása: https://gitlab.com/lbalazs96/bhax/tree/master/attention_raising/Source/BrainB

A program célja az eSport teheségek keresése. A program ezt úgy akarja elérni hogy azt a feladatot adja a felhasználónak, hogy kövessen egy Samu Entropy nevű formán. A program több szempontból vizsgálja a játékos teljesítményét. Az első az hogy mennyi időbe telik neki újra megtalálni az alakot miután azt esetleg elveszti a követésében. A második az hogy mennyire volt bonyolult az alak követése ennek elvesztésének pillanatában. A alakzatok bonyolódása lelassul ha elhagyjuk a követni akart Samu Entropy-t. A program egy fáljban értékeli a játékos teljesítményét, ez lesz a kimenete. A program a Qt grafikus felületét használja.



7.3. ábra. BrainB benchmark

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

Megoldás forrása: https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Tanulságok, tapasztalatok, magyarázat...

Ez a python program a TensorFlow könyvtárt használva ismer fel számjegyeket. Ehhez először egy MNIST adatbázist vesz alapul a tanuláshoz. Az MNIST erre a célra hozták lérte, hogy sok olyan kézzel írott számjegyeket tartalmaz amelyeket gépi tanulásra használnak többek közt.

A program futtatásához szükséges a a python3 developer csomag továbbá a tensorflow csomag is.

8.2. Minecraft-MALMÖ

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

Ezt a feladatot a SMINST for humans alkalmazásban elért lvl 6-os eredménnyel passzolom.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

9.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

9.3. Gimp Scheme Script-fu: króm effekt

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

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

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

Tanulságok, tapasztalatok, magyarázat...

9.4. Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

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

Tanulságok, tapasztalatok, magyarázat...

9.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9.6. Omega

Megoldás videó:

Megoldás forrása:

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

Juhász István - Magas szintű programozási nyelvek 1

Mikor programnyelvekről beszélünk akkor fontos ezeket osztályozni. Beszélhetünk magas szintű, assembly és gépi nyelvről. A magasszintű nyelven megírt program összeállítását a fordító program végzi szintaktikai és szemantikai szabályok alapján, gépi kódot készít belőle.

Programnyelvek osztályozása

Vannak imperatív nyelvek és deklaratív nyelvek. Az imperatív nyelvekben két alcsoportot különböztethetünk meg, eljárásorientált és objektumorienltált nyelveket. A deklaratív nyelvek nem algoritmikus nyelvek, a programozó csak a problémát oldja meg, a programozónak nincs lehetősége memóriaműveletre. Ennek alcsoportjai a funkcionális és logikai nyelvek.

Karakterkészlet

Minden programnyelv legkisebb alkotórészei a karakterek. Ezeket három kategóriába soroljuk, ezek a betűk, a számjegyek és az egyéb karakterek. A lexikális egységeket a fordító a lexikális elemzés során felismeri és tokenizálja.

Adattípusok

Minden adattípusnak megvan a saját neve, ami egyben az azonosítója is. Minden adatípusnak megvan az ábrázolási módja, itt van meghatározva hogy egyes elemek a típusból hány byte-ot foglalnak a memóriában. A nyelvek megengedik a saját típus létrehozását, ha ennek megadjuk a műveleteit, tartományát és reprezentációját. Léteznek nevesítettt konstansok amelyeknek három része van: név, típus, érték amit mindig deklarálni kell, ez az érték attól fogva nem változtatható meg. A változóknak mindaddig nincs értéke amíg az nincs meghatározva, ezt egyes programnyelvek alapértelmezett nullával javítják de mindaddig amíg nem deklaráltuk saját kézből ezek így ne használjuk.

Kifejezések

A kifejezések olyan szintaktikai eszközök amelyek operandusokból operátorokból és kerek zárójelekből állnak. Az operandus lehet literál, konstans, változó vagy függvényhívás. Az operátorok műveleti jelek. A kerek zárójelek a műveletek végrehajtási sorrendjét befojásolják, rendundásan alkalmazhatóak.

Utasítások

Az utasítások segítségével készíti el a fordító program a tárgyprogramot. Két részből állnak az utasítások: deklarációs és végrehajtó utasítások. A deklarációs utasítások közölik a fordítóprogrammal milyen üzemmódot állítson be és milyen szolgáltatást kérnek. A végrehajtó utasítások tartalmazzák a tárgykód lényegét. Ez lehet: értékadó utasítás, ugró, elágazó, ciklusszervező, hívó, vezérlésátadó,I/O utasítások és más utasítások.

A programok szerkezete

Egy program szövege programegységekből áll össze. A programegység lehet alprogram, blokk, csomag, taszk. Az alprogram az eljárásorientált programozási nyelvek alapja. Az alprogram az újrafelhasználásra tökéletesen alkalmas. Felépítése: név, formális paraméter lista, törzs, környezet. A név az azonosító és a fej része a formális paraméterekkel együtt. A törzsben deklarációs és végrehajtható utasítások szerepelnek. Az alprogramnak két fajtája van: eljárás és függvény. Az eljárás feladatokat hajt végre és ennek az eredményét használjuk fel. A függvény egyetlen értéket határoz meg.Egyes nyelvekben az alprogramnak meg lehet adni másodlagos belépési pontot íigy ne mcsak a fejen keresztül lehet meghívni.

A folyamatot mikor az alprogram hívásánál egymáshoz rendelődnek a formális és aktuális paraméterek paraméterkiértékelésnek nevezzük. A blokk olyan programegység amely egy másik programegységbe van beágyazva. A blokknak nincs paramétere és bárhol elhelyezhető.

Paraméterkiértékelés

A paraméterkiértékelődés során egymáshoz rendelődnek a formális és az aktuális paraméterek. Mindig az aktuális paramétereket rendeljük a formálisakhoz mert a formálisok az alprogram specifikációját tartalmazzák. Az aktuálisokból annyi van anyányszor meghíjuk az alprogramot. Két féle rendelődés létezik: sorrendi és név szeritni rendelődés. A sorrendi az aktuális és formalis paramétereket a felsorolás sorrendjében rendeli egymáshoz. A név szerinti az egymáshoz kötést a paraméterlistában határozzuk meg.

Paraméterátadás

Paraméterátadási módok: érték, cím, eredmény, név és cím szerinti. Érték szerinti átadás esetén az alprogram az aktuális paraméterek értékeit kapják meg csak, ezek értkéke nem változik az alprogramon kívül. A hívott program semmit nem tud a hívóról csak. Az információ áram egyirányú. A címszeritni paraméterátadáskor a formális paramétereknek nincs nincs címkomponensük, az aktuális paramétereknek viszont feltétlen rendelkezniük kell vele. Az információátadás itt kétirányú, a hívó program az aktuális címkomponensén kresztül adhat és vehet át értékeket. Az alprogramok formális paramétereinek három csoportja van: input, output és input-output paraméterek.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

Vezérlési szerkezetek

A kifejezések utasításokká válnak ha zokat pontosvessző követi. A kapcsos zárójelek között deklarációkat és utasitásokat helyezhetünk el. Ha kapcsos zárójelek között csak egy utasítás szerepel akkor az egyenértékű lesz. A következő vezérlési szerkezeteket külömbötethetünk meg: az if-else utasítás döntést írunk le. A switch utasítás nevezhető ennek a bővített formájának, mivel a switch a döntést több kimenetre is szét tudja ágaztatni, miközben az if utasítás ugyan ezt csak több, egybe ágyazott utasítással tudja elérni. A while utasítás addig ismétli a benne található utasításokat amíg a while ciklus fejében lévő feltétel teljesül. A

for utasítás hasonlóan működik viszont annak meghatározott, előre definiált ismétlési száma van. Abreak utasítás megállítja azt az utasítást amelybe be van helyezve. A goto utasítás a vezérlés minden további dolog nélkül átadható a program bármely részének.

10.3. Programozás

[BMECPP]

A C++ nem objektumorientált tulajdonságai

A c++ nyel a c programnyelv továbbfejlesztett változata. A C nyelv többször gondot okozó részeit cseréli le, más kényelmesebben használható szolgáltatásokra. C++ban a void függvény egy üres paramüterlistával rendelkező függvény, ellentétben a c nyelvben ugyanez a gaykorlat azt jelenti hogy a függvény bármennyi paraméterrel meghívható. C++ban a min függvényben nem kötelezőa return használata. Bevezették a bool típust ami csak true vagy false értéket vehet fel. Bármilyen helyen állhat változódeklaráció ahol utasítás is állhat. A címszerinti paraméterátasás abban változott hogy a paraméter deklarációjában csak egy és jelet kell írnunk a praméter elé.

III. rész

Második felvonás

Bátf41 Haxor Stream
A feladatokkal kapcsolatos élő adásokat sugároz a https://www.twitch.tv/nbatfai csatorna, melynek permanens archívuma a https://www.youtube.com/c/nbatfai csatornán található.

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/-pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, https://groups.google.com/forum/#!forum/nemespor, az UDPROG tanulószoba, https://www.facebook.com/groups/udprog, a DEAC-Hackers előszoba, https://www.facebook.com/-groups/DEACHackers (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.