Univerzális programozás

Így neveld a programozód!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Varga, Levente Zoltán	2019. május 10.	

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

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

Ábrák jegyzéke

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?

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.

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ha végtelen ciklus akarunk előidézni, érdemes azt for ciklus segítségével tennünk.

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

Ez a program a processzorunk egy magját fogja 100%-ban használni.

A végtelen ciklus 0%-ban használja a processzort :

```
#include <unistd.h>
int main()
{
for(;;)
{sleep(1);}
return 0;
}
```

A sleep függvény "elaltatja" a folyamatot a ()-ben szereplő másodpercre, így a programunk nem fogja használni a CPU-t.

Egy végtelen ciklus, amely a processzorunk összes magját 100%-ban használja:

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

Az OpenMP segítségével egy feladat párhuzamosan több szálon is futhat. Fontos, hogy a fordítás során használjuk a "-fopenmp" kapcsolót.

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra 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...

A példában látható T100-as programban található Lefagy függvény képes eldönteni, hogy egy program tartalmez-e végtelen ciklust. Ha van benne, akkor igazat ad vissza, egyébként hamis értéket. A T100 felhasználásval elkészített T1000 egy Lefagy2 függvénnyel egészül ki, amely a Lefagy eredményével dolgozik tovább. Ha a program lefagy, akkor a Lefagy2 igazat ad vissza, ha a program nem fagy le, akkor pedig végtelen ciklusba kerülünk. Adjuk meg a T1000 önmagát paraméterül. Ha a paraméterül adott T1000 lefagy, akkor a vizsgáló T1000 nem fagy le, hanem visszaad egy igaz értéket. Ellenkező esetben a vizsgált T1000 nem fagy le, de ami viszgálja az végtelen ciklusba kerül. Ezek alapján látható, hogy nem létezik olyan program, ami képes egy másikról eldönteni, hogy lefagy-e.

2.3. Változók értékének felcserélése

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

Megoldás videó: https://bhaxor.blog.hu/2018/08/28/10_begin_goto_20_avagy_elindulunk

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Segédváltozó bevezetésével könnyedén felcseréhetünk két váltizót.

```
#include <stdio.h>

int main()
{
int a= 4;
int b= 5;

int c= a;
a=b;
b=c;
```

Ezen nincs sok magyarázni való. Először c megkapja a-t értékül, majd az a-nak átdjuk b értékét, és végül b értékét felülírjuk c értékével, ami ugye a-nak a kezdeti értéke volt

2.4. Labdapattogás

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Először deklaráljuk a szükséges változókat. Az x és y a labdánk aktuális helyét jelzi, azért adunk nekik 0 kezdőértéket, hogy az ablakunk bal felső sarkából kezdjen pattogni. A lépkedés szabályozásában fog segítséget nyújtani az xnov és az ynov. Az mx és my változókban az ablakunk méretét fogjuk eltérolni.

```
    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;
```

Egy végtelen ciklusban a labdánk addig fog pattogni, amíg meg nem állítjuk.

```
    for ( ;; ) {

        getmaxyx ( ablak, my , mx );

        mvprintw ( y, x, "O" );

        refresh ();
        usleep ( 100000 );

        x = x + xnov;
        y = y + ynov;

        if ( x>=mx-1 ) {
            xnov = xnov * -1;
        }
        if ( x<=0 ) {
            xnov = xnov * -1;
        }
        if ( y<=0 ) {
            ynov = ynov * -1;
        }
        if ( y>=my-1 ) {
            ynov = ynov * -1;
        }
```

A getmaxyx függvény segítségével beolvassuk és eltároljuk az ablak méretét. Ezt azért írjuk a cikluson belülre, hogy ha pattogás közben változtatjuk az ablak méretét, akkor érzékelje azt a programunk. A mvprinntw-vel egyszerűen kiírjuk labdánkat, a usleep függvénnyel pedig a pattogás gyorsaságát módosíthatjuk

Ezek után a labda következő helyét adjuk meg.

Az if-ek arra szolgálnak, hogy eldöntse a program, hogy elérte-e az ablak szélét vagy sem. Ha elérte a megfelelő növekedési értéket megszorozza (-1)-gyel, így visszafordul a labdánk.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ahhoz, hogy megnézzük az int méretét szükségünk van 2 int-re. Az elsőnek a hosszát fogjuk viszgálni, a másodikban pedig eltároljuk.

```
int szo = 1;
int hossz = 0;
```

A bitshift operátor segítségével egyesével léptetjük az int szo bitjeit balra, amíg a szó végére nem érünk és minden lépésnél növeljük a hossz-t.

```
do
hossz++;
while (szo<<=1);
```

Ha lefuttatjuk a programot, és kiíratjuk a hossz-t látható, hogy egy szó hossza 32 bit.

2.6. Helló, Google!

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A PageRank egy algoritmus, ami rendezi az oldalkat a rájuk mutató linkek száma alapján, mivel amelyik oldalra több link mutat, az nagyobb eséllyel lesz hasznos. Ez az algoritmus Google keresőmotorjának szerves részét képezi.

A hálózatunk 4 honlapjai közti kapcsolatot egy 4x4-es mátrixban tároljuk el.

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

void kiir (double tomb[], int db)
{
  for(short i = 0; i < db; ++i)
    printf("%.2f \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);
}
```

```
int main()
{
  double L[4][4] = {
    {0.0, 0.0, 1.0 / 3.0, 0.0},
    {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
    {0.0, 1.0 / 2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0 / 3.0, 0.0}
  };
  double PR[4] = {0.0, 0.0, 0.0, 0.0};
  double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};
  int i,j;
  for(;;)
  {
    for(i = 0; i < 4; ++i)
    {
      PR[i] = 0.0;
      for(j = 0; j < 4; ++j)
  PR[i] += (L[i][j] * PRv[j]);
    }

    if(tavolsag (PR, PRv, 4) < 0.00000001)
      break;

    for(i = 0; i < 4; ++i)
      PRv[i] = PR[i];
  }
  kiir (PR, 4);
  return 0;
}
```

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

Azt a számot amelyik csak 1-gyel és önmagával osztható prímszámnka nevezzük. Ikerprímnek azokat a prímekez nevezzük, melyeknek különbsége 2 (pl.: 3 és 5, 5 és 7,...). Prímszámból végtelen sok van, de azt nem tudjuk biztosan, hogy mennyi ikerprím van.

A Brun tétel szerint az ikerprímek reciprokainak összegéből képzett sor konvergens. Ez azt jelenti, hogy egy értéket soha el nem érve növekszik. Ez az érték a Brun-konstans.

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...

Három ajtó közül az egyik mögött van a nyeremény. A játékos kiválaszt egy ajtót, de nem nyitja ki. A játékmester -aki tudja hol lapul a nyeremény- kinyit egy ajtót a maradék kettő közül, amely mögött nem a nyeremény van. Ezután a játékos eldöntheti, hogy marad-e a választott ajtónál vagy a másikat választja. Megéri-e változtatni?

Ezt foglalja magában a Monty Hall probléma.

Ha az ember belegondol, azt mondja, hogy ugyanannyi esélye van mindkét ajtóval. Ez azonban nem így van. Az első választásnál 1/3 esélyünk van nyerni, mivel három ajtó közül választunk. Annak az esélye, hogy a másik két ajtó mögött van a nyeremény 2/3. Miután a játékmester kinyitott egyet a két ajtó közül ez a 2/3 esély "átszáll" a maradék egy ajtóra. Ezt mutatja számtalan számítógépes szimuláció is.

Így a válasz a válasz az, hogy igen, megéri változtatni döntésünkön.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az unáris számrendszer csak egyesekből áll. Decimálsból úgy váltunk át egy számot, hogy annyi egyest -a programunkban vonalat- írunk, amennyi a szám értéke. Hogy egyszerűbb legyen kiolvasni az unáris értéket minden 5 vonal után teszünk egy szóközt.

```
#include <iostream>

int main()
{
  int szam;
  int vonal = 0;
  std::cout<<"Adj meg egy szamot!\n";
  std::cin >> szam;
  std::cout<<"Unárisban:\n";
  for (int i = 0; i < szam; ++i)
  {
    std::cout<<"|";
    ++vonal;
    if (vonal % 5 == 0)
  std::cout<<" ";
  }
  std::cout<<'\n';
  return 0;
}
```

A programunk először bekér egy decimális számot. For ciklussal 1-től a bekért számig megy, és minden kiírt vonal után növeli a számot eggyel. Minden lépésnél megnézzük hogy a szám osztható-e öttel, ha igen akkor a programm rak egy szóközt.

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...

Noam Chomsky, amerikai nyelvész volt, aki javasolta a generatív nylevek formalizálását. Nevéhez fűződik még a generítív nyelvtanok csoportosítása, melyek közül a környezetfüggőről (1.típus) lesz szó.

```
S, X, Y - változók
a, b, c - konstansok
S → abc, S → aXbc, Xb → bX, Xc → Ybcc, bY → Yb, aY -> aaX, aY → aa
S-ből indulunk.
                S (S → aXbc)
                aXbc (Xb → bX)
                abXc (Xc → Ybcc)
                abYbcc (bY → Yb)
                aYbbcc (aY -> aaX)
                aaXbbcc (Xb → bX)
                aabXbcc (Xb → bX)
                aabbXcc (Xc → Ybcc)
                aabbYbccc (bY → Yb)
                aabYbbccc (bY → Yb)
                aaYbbbccc (aY → aa)
                aaabbbccc
```

```
A, B, C - változók
a, b, c - konstansok
A → aAB, A → aC, CB → bCc, cB → Bc, C → bc
A-ból indulunk
                A (A → aAB)
                aAB ( A → aAB)
                aaABB ( A → aAB)
                aaaABBB ( A → aC)
                aaaaCBBB (CB → bCc)
                aaaabCcBB (cB → Bc)
                aaaabCBcB (cB → Bc)
                aaaabCBBc (CB → bCc)
                aaaabbCcBc (cB → Bc)
                aaaabbCBcc (CB → bCc)
                aaaabbbCccc (C → bc)
                aaaabbbbcccc
```

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:

Tanulságok, tapasztalatok, magyarázat...

A C89-es szabványban nem létezik egysoros komment, ezért ha írunk egy programot ami tartalmaz ilyen kommentet, az C99-ben lefordul, de C89-ben nem.

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

A programunk fordításánál használjuk az "-std=c89" majd az "-std=c99" kapcsolót, hogy megnézzük valóban működik-e C99-ben.

C89-ben a for ciklus kezdőértékét a cikluson kívül kell deklarálnunk, C99-ben cikluson belül is megengedhető.

```
int main()
{
for(int i=1; i<10; i++)
return 0;
}
```

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A lex program segítségével készítünk egy lexikális elemző szabályok megadásával. Szövegfájlból olvas be és C forráskódot készít.

```
%{
#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;
}
```

Fordítás és futtatás:

```
$ lex -o lexikalis.c lexikalis.l
$ gcc lexikalis.c -o lexikalis
$ ./lexikalis
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A leet nyelv lényege, hogy a betűket kinézetre hasonló számokkal vagy más karakterekkel helyettesítjük.

Mint az előző feladatban, itt is a lex-et fogjuk ígénybe venni.

```
%{
  #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", "4", "4"}},
  {'b', {"13", "8", "|3", "|}"}},
  {'c', {"[", "(", "<", "{"}},
  {'d', {"|>", "|)", "|]", "|}"}},
  {'e', {"3", "€", "&#x20a4;", "£"}},
  {'f', {"f", "|=", "ph", "|#"}},
```

```
  {'g', {"C-", "6", "[", "[+"}},
  {'h', {"|+|", "4", "|-|", "[-]"}},
  {'i', {"1", "9", "|", "!"}},
  {'j', {"_)", "_7", "_|", "_/"}},
  {'k', {"I{", "|<", "1<", "|{"}},
  {'l', {"][", "1", "|", "|_"}},
  {'m', {"^^", "44", "|V|", "(V)"}},
  {'n', {"][\\][", "|\\\|", "/\\/", "/V"}},
  {'o', {"0", "oh", "()", "[]"}},
  {'p', {"|7", "/o", "|D", "|o"}},
  {'q', {"kw", "9", "O_", "(,)"}},
  {'r', {".-", "I2", "12", "|2"}},
  {'s', {"s", "5", "$", "§"}},
  {'t', {"+", "7", "7", "'|'"}},
  {'u', {"{_}", "|_|", "(_)", "[_]"}},
  {'v', {"\\//", "\\/", "\\/", "\\/"}},
  {'w', {"2u", "VV", "\\/\\/", "(/\\)"}},
  {'x', {"><", "%", ")(", ")("}},
  {'y', {"y", "y", "y", "y"}},
  {'z', {"5", "2", "7_", ">_"}},

  {'0', {"D", " ", "D", "0"}},
  {'1', {"I", "I", "L", "L"}},
  {'2', {"Z", "Z", "e", "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"}}
  };

%}
%%
. {

    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;
}
```

Láthatjuk, hogy létrehozunk egy cipher struktrúrát, amely a megadott karakterekhez rendel egy négyelemű tömböt, amely a helyettesítési értéket fogja tartalmazni.

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);
```

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

Ha nem hagyja figyelmen kívül a jelet, akkor a jelkezelő függvénynek átadjuk.

Az második és harmadik kódcsipet ugyanazt az eredményt fogja visszaadni.

A negyedik csipet viszont már bugos. A "tomb[i] = i++" bal oldala az i-t a tömb indexeként használja fel a jobb oldal az i-t pedig növeli. A problémát az okozza, hogy nem ismerjük a végrehajtás sorrednjét, így ez hibás eredményhez vezethet.

Az ötödik kódban a probléma forrása az `&&` operátor jobb oldala. Mivel == helyett =-t használunk, így nem összehasonlítunk, hanem értéket adunk ami nem boolean típusú lesz.

A hatodik forrásban az f függvény két egészet kap, de a kiértékel sorrendjét nem ismerjük, ezért ez a kód is hibás eredményhez vezet.

A hetedik kód megfelelően működik. Először kiírja a függvény által a módosított értéket, majd a kezdőértéket.

Az utolsó kódcsipetben a printf két egészet fog kiírni. Az `f` függvény megkap egy memóriacímet, és az itt található értékkel fog dolgozni. A második számunk egy sima változó.

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...

Először is nézzük meg,hogy mely kifejezés mit is jelent.

`\forall`: univerzális kvantor (minden)

`\exist`: egzisztenciális kvantor (van olyan)

`\wedge`: implikáció ("ha..., akkor...")

`\supset`: konjunkció (és)

`S`: ez arnyelvben a rákővetkező függvény

`\text`: szöveg kiírása

Most hogy tisztában vagyunk a jelentésekkel, vessünk egy pillantást a feladatra.

1. Minden x-re létezik olyan y, amelynél ha x kisebb, akkor y prím. -> Végtelen sok prímszám van.

2. Minden x-re létezik olyan y, amelynél ha x kisebb, akkor y prím, és ha y prím, akkor annak második rákövetkezője is prím. -> Végtelen sok ikerprím van

3. Létezik olyan y, amelyhez minden x esetén az x prím ,és x kisebb, mint y. -> Véges sok prím van

4. Létezik olyan y, amelyhez minden x esetén az x nagyobb, és x nem prím. -> Véges sok prím van

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

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);
  ```

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
int a;
```

Egy egészet vezet be.

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

Ez egy egészre mutattó mutató

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

Egy egész referenciáját vezeti be a programba. A referencia értéke ugyanaz mint az egészé.

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

Egy ötelemű egészekből álló tömb.

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

Ötelemű egészekből álló tömb refernciája. Az összes elem referenciája, nem csak az első elemé.

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

Egészekre mutató mutatók tömbje.

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

Ez egy függvény ami, egészre mutató mutatót ad vissza.

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

Ez a függvény egy egészre mutató mutatót visszaadó függvényre mutató mutató

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

Egészet visszaadó és két egészet beolvasó függvényre mutató mutatót visszaadó és egészet kapó függvény.

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

Függvénymutató az előbbivel azonos függvényre.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Az alsó háromszög mátrixnak csak a főátlójában és az alatt vannak értékek, a főátló felett üres, és mivel négyzetes mátrixról beszélünk, ugyanannyi sora van mint oszlopa.

A programunk elején létrehozzuk az nr egészet, ami a sorok számát fogja jelenteni, és a **tm mutatót is. Az első hexadecimális szám amit a programunk kiír, az a mutatónk memóriacíme lesz.

A malloc segítségével tárhelyet foglalhatunk a memóriában és eredményül ennek a pointerét kapjuk vissza.

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

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

Itt lefoglalunk 5x8 bájtot, majd kiiratjuk a lefoglalt hely címét.

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

    }

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

Ezzel pedig minden sornak elemszámainak megfelelő alkalommal foglalunk 8 bájtot. A végén kiírjuk az első sor memóriacímét.

Ezután feltöltjük a mátrixunkat.

```
for (int i = 0; i < nr; ++i)
        for (int j = 0; j < i + 1; ++j)
            tm[i][j] = i * (i + 1) / 2 + j;
```

A külső ciklus a sorokon megy végig, a belső pedig az oszlopokon.

Itt láthatunk 4 variácót konkrét elemek megadására:

```
    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;
    *(tm[3] + 2) = 44.0;
    *(*(tm + 3) + 3) = 45.0;
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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

A kulcs mérete és a buffer mérete, ezeket még a main() előtt definiáljuk, a későbbiekben használni fogjuk őket.

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

A megzsokottól eltérő módon main-nek argumnetumokat adunk át, ezeket általában futtatáskor adjuk meg neki treminääből.

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

Létrehozunk 2 tömböt. Az elsőben a kulcsot, a másodikban a beolvasott karaktereket tárolja. A méret megadásához felhasználjuk a main() előtt értékeket.

```
  int kulcs_index = 0;
  int olvasott_bajtok = 0;
```

Ezeknek a tömb bejárásánál lesz szerepe.

```
  int kulcs_meret = strlen (argv[1]);
```

Az argv a paracsnsori argumentumot jelenti, tehát a kulcsméret a második argumentum hosszát kapja meg értékül.

```
  strncpy (kulcs, argv[1], MAX_KULCS);
```

Az argv[1]-et bemásoljuk a kulcs tömbbe karaterenként.

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

A read megadásához szükségünk lesz 3 argumentumra: honnan olvasunk be, hol tároljuk, beolvasott bájtok száma. A programunkban a read a standard inputról olvas be (0), a buffer tömbben tárol, és 256 karaktert olvas be.

```
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 for ciklussal bejárjuk a buffer tömböt és elemeit egyesével exor-ozzuk a kulcs megfelelő elemével. Ez addig megy amíg a kulcs_index eléri a kulcs_meret-et, utána pedig nullázódik. A végén pedig kiírjuk a buffer tartalmát standard outputra (1).

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Ez a titkosító Java-ban van megírva. Hasonló az előző feladathoz, csak más nyelven van. Nézzük hát hogyan is néz ki.

```
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();

        }

    }

}
```

Egy ExorTitkosító nevű publikus osztályban hozzuk létre a titkosítót (mivel a Java objektumorientált nyelv, így ez a természetes). A kulcs és a buffer is byte tömbökben fog tárolódni. A kulcs tratalmát a kulcsSzövegből fogjuk beolvasni. A buffer mérete csak úgy mint a C példában, itt is 256 bájt.

A while ciklus addig olvassa be az adatot, amíg van mit. A belső for ciklus pedig a titkosítást valósítja meg, majd kiírjuk a bufferbe a végeredményt.

A try-catch rész az esetleges hibák "elkapására" szolgál.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 8
```

Ahogy a titkosítónál, úgy itt is néhány állandó definiálásával kezdjük a programot.

```
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;
}
```

Ez a függvény a szóhossz áltlagára fog szolgálni. Végig megyünk a megadott tömbön és minden szóköz után növeljük az sz-t. A ciklus végén elosztjuk a szöveg méretét a szavak számával, így megkapjuk az átlagot.

```
int
tiszta_lehet (const char *titkos, int titkos_meret)
{
    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 program arra épít, hogy a tiszta szövegünk tartalmazza a legygyakoribb magyar szavakat ("hogy", "nem", "az", "ha"). Ha ez nem teljesel, nem tudjuk feltörni a szöveget.

```
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;

    }

}
```

Ha valamit kétszer exor-ozunk visszakapjuk azt, amiből indultunk. Az exor függvény ezt használja ki.

```
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);
}
```

Ez a függvény eldönti, hogy tiszta-e már a szövegünk.

```
int
main (void)
{

    char kulcs[KULCS_MERET];
    char titkos[MAX_TITKOS];
    char *p = titkos;
    int olvasott_bajtok;
```

Megérkeztünk a program main részéhez. A szükséges tömbök és változók deklarálásával indítunk. A tömbök méretét már a kód legelején definiált értékek határozzák meg.

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

for (int i = 0; i < MAX_TITKOS - (p - titkos); ++i)
        titkos[p - titkos + i] = '\0';
```

Beolvassuk a bájtokat, amíg a buffer be nem telik, vagy elérjük a bemenet végét. Ezután a for ciklussal kinullázzuk a bufferben megmaradt helyeket.

```
    for (int ii = '0'; ii <= '9'; ++ii)
        for (int ji = '0'; ji <= '9'; ++ji)
            for (int ki = '0'; ki <= '9'; ++ki)
                for (int li = '0'; li <= '9'; ++li)
                    for (int mi = '0'; mi <= '9'; ++mi)
                        for (int ni = '0'; ni <= '9'; ++ni)
                            for (int oi = '0'; oi <= '9'; ++oi)
                                for (int pi = '0'; pi <= '9'; ++pi)
                                {
                                    kulcs[0] = ii;
                                    kulcs[1] = ji;
                                    kulcs[2] = ki;
                                    kulcs[3] = li;
                                    kulcs[4] = mi;
```

```
                kulcs[5] = ni;
                kulcs[6] = oi;
                kulcs[7] = pi;

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

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

A program for ciklusok segítségével előállítja az összes lehetséges kulcsot. Meghívjuk az exor_tores függvényt, ha igazat ad vissza, kiírjuk a kulcsot és a tiszta szöveget. Végül pedig az exor függvéényt hívjuk meg egy második buffer létrehozásásnak elkerülése végett.

4.5. Neurális OR, AND és EXOR kapu

R

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

Tanulságok, tapasztalatok, magyarázat...

Ebben a feladatban újra visszatérünk a Monty Hall problémánál megismert R nyelvhez. Segítségével neurális hálózatot fogunk létrehozni. Nevét a neuronról kapta, mely agyunk egyik sejtje. Feladata az elektromos jelek összegyűjtése, feldolgozás és terjesztése. 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.

```
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)
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])
```

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

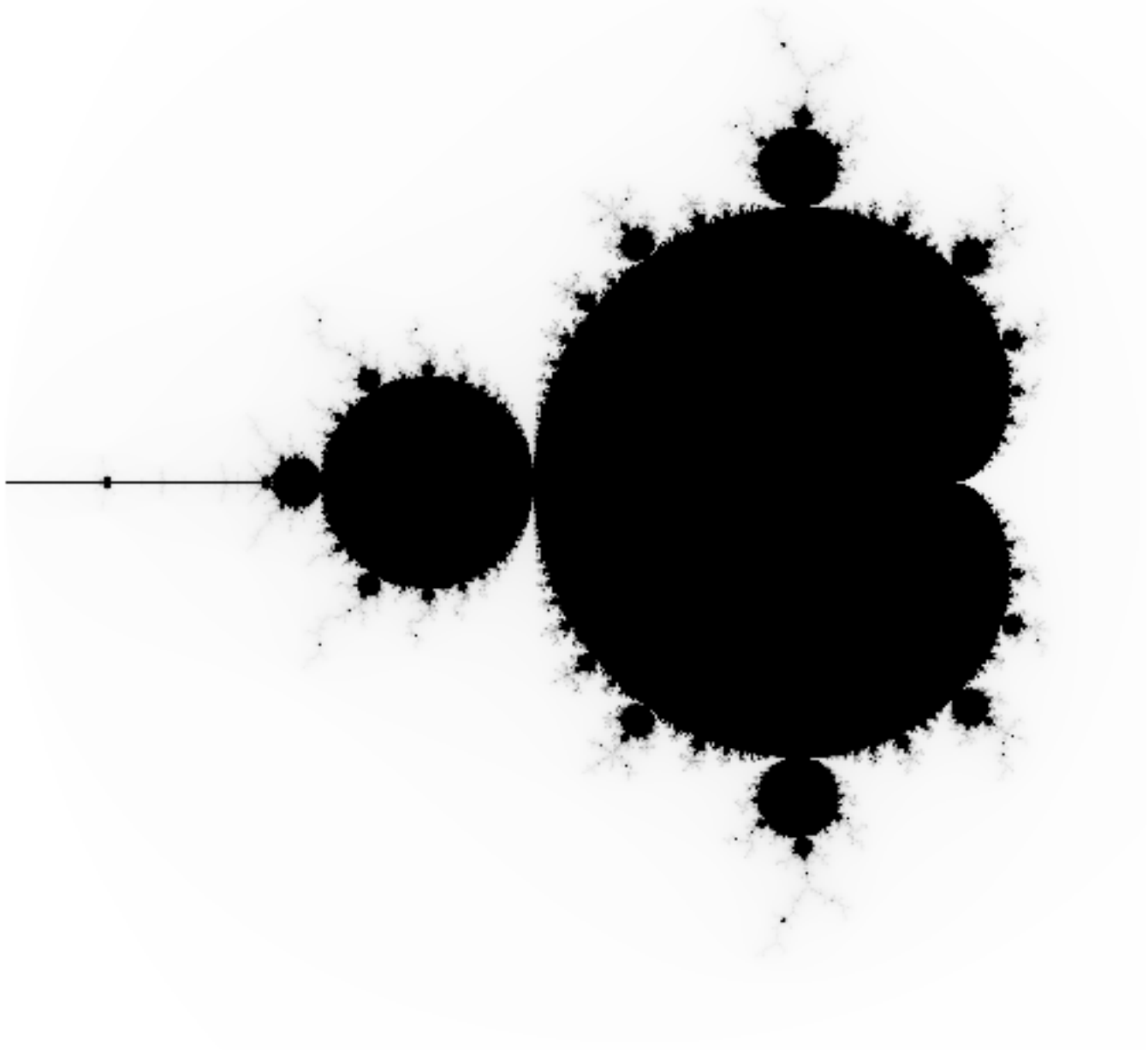
Megoldás forrása:https://github.com/nbatfai/nahshon/blob/master/ql.hpp#L64

Tanulságok, tapasztalatok, magyarázat...

Ehhez a feladathoz szükségünk lesz néhány fájlra. Először kelleni fog a mandelpng.cpp, aminek segítségével készíthetünk egy png fájlt ami a mandelbrot halmazt ábrázolja. A program nem tartalaz header-t így szükségünk lesz a -lpng kapcsolóra. Fordítása és futtatása itt látható:

```
g++ madndelpng.cpp -o mandel -lpng
./mandel mandel.png
```

Szükségünk van a mandel.png, ami a progrmanak bementül fog szolgálni.



4.1. ábra. mandel.png

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

int main (int argc, char **argv)
{
  png::image <png::rgb_pixel> png_image (argv[1]);
  int size = png_image.get_width() *png_image.get_height();
  Perceptron* p = new Perceptron (3, size, 256, 1);
```

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

A fent látható kód a main.cpp. Létrehoz egy üres png-t a bemenetként kapott fájl méreteivel. A kép méretét eltároljuk egy változóban, majd létrehozunk felhasználó által definiált típust

A két for ciklus segítségével egy újonnan lefoglalt tárba átmásoljuk a beolvasott kép piros pixeleit.

A program fordítása és futtatása:

```
g++ mlp.hpp main.cpp -o perc -lpng -std=c++11
./perc mandel.png
```

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

A Mandelbrot-halmaz Benoît Mandelbrot francia-amerikai matematikus nevéhez fűződik. Ő fedezte fel a fraktálokat. A fraktálok végtelenül komplex matematikai alakzatok, melyekben ismétlődés fedezhető fel.

Az előző fejezetben már látható volt hogyan is néz ki a halmaz. A következő programmal pedig ezt fogjuk kiszámolni.

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

A program működéséhez szükséges a png++ header amit telepíthetünk a `sudo apt-get install libpng++-dev` parancs segítségével.

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

Parancssori argumentummal adjuk meg, hogy milyen féjlba mentse az elkészült halmazt.

```
    if (argc != 2) {
        std::cout << "Hasznalat: ./mandelpng fajlnev";
        return -1;
```

Ha nem adunk meg fájlnevet a programnak, hibával fog visszatérni.

```
    double a = -2.0, b = .7,  c = -1.35, d = 1.35;
    int szelesseg = 600, magassag = 600, iteraciosHatar = 1000;
```

Először megadjuk a függvény értékkészletét és értelmezési tartományát. Az int-ek neve egyértelműek, megadjuk a szélességet, magasságot és az iterációs határt. Ezekre az adatokra szükségünk lesz a továbbiakban.

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

Ezzel a sorral létrehozunk eg yüres png-t a megadott méretekkel.

```
    for (int j=0; j<magassag; ++j) {
        for (int k=0; k<szelesseg; ++k) {
            reC = a+k*dx;
            imC = d-j*dy;
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            while (reZ*reZ + imZ*imZ < 4 && iteracio < iteraciosHatar) {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ*reZ - imZ*imZ + reC;
                ujimZ = 2*reZ*imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;

                ++iteracio;

            }

            kep.set_pixel(k, j, png::rgb_pixel(255-iteracio%256,
                                               255-iteracio%256, 255- ↩
                                                  iteracio%256));
        }
        std::cout << "." << std::flush;
    }
```

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

Megoldás videó:

Megoldás forrása:

Az előző példában a komplex számok imaginárius és valós részét két külön változóban adtuk meg. Az `std::complex` osztállyal nincs szükségünk két változóra. A megoldás hasonlít az elsőre, de lássuk hogyan is néz ki pontosan.

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

  int szelesseg = 1920;
  intmagassag = 1080;
  intiteraciosHatar = 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] );
    }
```

Mivel egy új osztállyal dolgozunk, így azt ne felejtsük include-olni azt. A program eleje tartalmaz alapértelemezett értékeket, ha rosszul futtatnánk. Viszont ha helyesen megadjuk a 9 argumentumot, akkor a tetszőleges értékekkel fog dolgozni a program.

```
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;
}
```

Itt látjuk a komplex típus használatát. Több double változót tárol(valós és képzetes rész).

```
          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;
      //kiírja, hogy hány százaléknél tart a képgenerálás
      std::cout << "\r" << szazalek << "%" << std::flush;
    }
```

Ez az egymásba ágyazott for ciklusok foltyatása.

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

```
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 );
```

A program hasonlóan kezdődik a másodikhoz, anny eltéréssel, hogy itt több argumentumot kell megadnunk. Ha nem megfelelően futtattuk itt is hibával tér vissza, mint az első példában. Miután a program mindent rendben talál, létrehozza az üres képfájlt.

```
    for ( int y = 0; y < magassag; ++y )
```

```
    {

        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;
    }
```

Az az első két for ciklus végigmegy a rácspontokon, a legbelső, harmadik pedig a függvényértéket számolja, amíg az iterációs határt el nem érjük vagy az if nem teljesül.

```
        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;
```

Az utolsó előtti sor pedig kiírja a fájlba a pixeleket.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

Ebben a feladatban a Mandelbrot halmaz megalkotásához igénybe fogjuk venni az NVIDIA CUDA technológiáját. Ezzel lehetőségünk nyílik a GPU erőforrásainak igénybevételéhez, amivel lényegesen felgyorsítjuk a számolási folyamatot. Szükségünk lesz hozzá egy Nvidia GPU-ra valamint az nvidia-cuda-toolkit telepítésére.

```
#define MERET 600
#define ITER_HAT 32000

}
```

Először is definiáljuk az iterációs határt és a méretet.

```
__device__ int
mandel (int k, int j)
{
    // Végigzongorázza a CUDA a szélesség x magasság rácsot:
    // most eppen a j. sor k. oszlopaban vagyunk

    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;

    // c = (reC, imC) a rács csomópontjainak
    // megfelelő komplex szám
    reC = a + k * dx;
    imC = d - j * dy;
    // z_0 = 0 = (reZ, imZ)
    reZ = 0.0;
    imZ = 0.0;
    iteracio = 0;
    // z_{n+1} = z_n * z_n + c iterációk
    // számítása, amíg |z_n| < 2 vagy még
    // nem értük el a 255 iterációt, ha
    // viszont elértük, akkor úgy vesszük,
    // hogy a kiinduláci c komplex számra
    // az iteráció konvergens, azaz a c a
    // Mandelbrot halmaz eleme
    while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
    {
        // z_{n+1} = z_n * z_n + c
        ujreZ = reZ * reZ - imZ * imZ + reC;
        ujimZ = 2 * reZ * imZ + imC;
        reZ = ujreZ;
```

```
        imZ = ujimZ;

        ++iteracio;

    }
    return iteracio;
}
```

A függvény előtti __device__ jelzi, hogy a GPU-val fogjuk számoltatni a kovetkező mandel függvényt. Ez fogja kiszámolni a Mandelbrot halmazt, az első feladatban már lathattunk példát a complex osztály nélküli megoldásra. Ez is ugyanaz.

```
__global__ void
mandelkernel (int *kepadat)
{

    int tj = threadIdx.x;
    int tk = threadIdx.y;

    int j = blockIdx.x * 10 + tj;
    int k = blockIdx.y * 10 + tk;

    kepadat[j + k * MERET] = mandel (j, k);

}
```

Itt a __global__ jelzi, hogy a GPU-val fogunk számolni. A threadIdx.x és threadIdx.y jelöl, hogy melyik szálon történik a számhoz tartozó érték számolása.

```
void
cudamandel (int kepadat[MERET][MERET])
{

int *device_kepadat;
cudaMallocManaged ((void **) &device_kepadat, MERET * MERET * sizeof (int)) ↩
   ;


// dim3 grid (MERET, MERET);
// mandelkernel <<< grid, 1 >>> (device_kepadat);

dim3 grid (MERET / 10, MERET / 10);
dim3 tgrid (10, 10);
mandelkernel <<< grid, tgrid >>> (device_kepadat);

cudaMemcpy (kepadat, device_kepadat,
        MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
cudaFree (device_kepadat);

}
```

A cudamandel függvény kap egy 600x600-as tömböt. Készítünk egy mutatót és akkora helyet foglalunk neki, mint az értékül kapott tömbé. Ha végzett a függvény átmásoljuk az értékeket a kepadat tömbbe, majd felszabadítjuk a lefoglalt tárhelyet.

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

// Mérünk időt (PP 64)
clock_t delta = clock ();
// Mérünk időt (PP 66)
struct tms tmsbuf1, tmsbuf2;
times (&tmsbuf1);

if (argc != 2)
    {
    std::cout << "Hasznalat: ./mandelpngc fajlnev";
    return -1;
    }

int kepadat[MERET][MERET];

cudamandel (kepadat);

png::image < png::rgb_pixel > kep (MERET, MERET);

for (int j = 0; j < MERET; ++j)
    {
    //sor = j;
    for (int k = 0; k < MERET; ++k)
    {
    kep.set_pixel (k, j,
            png::rgb_pixel (255 -
                    (255 * kepadat[j][k]) / ITER_HAT,
                    255 -
                    (255 * kepadat[j][k]) / ITER_HAT,
                    255 -
                    (255 * kepadat[j][k]) / ITER_HAT));
    }
    }
kep.write (argv[1]);

std::cout << argv[1] << " mentve" << std::endl;

times (&tmsbuf2);
std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
    + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

delta = clock () - delta;
std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;
```

```
}
```

A main függvény annyiban tér el a az eddig látottaktól, hogy futási időt is mér, így össze tudjuk hasonlítani a CUDA-s megoldás sebességét a simával.

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://github.com/vargalz/prog1konyv/tree/master/mandelnagyito

Megoldás videó:

Megoldás forrása:

Ebben a feladatban is szükségünk lesz egy Mandlbrot halmaz készítésére, viszont itt már lehetőségünk nyílik arra, hogy kinagyítsuk a kapott képet és megvizsgálhatjuk részletesen.

Először is telepítsük a szükséges könyvtárat: sudo apt-get install libqt4-dev. Ha ezzel megvagyunk szerezzük be a szükséges fájlokat.

A program működéséhez szükségünk lesz 5 fájlra. A frakablak.cpp, frakszal.cpp, és a hozzájuk tartozó header fájlokra, azaz a frakablak.h és a frakszal.h, valamint kelleni fog a main.cpp. Ezeknek egy mappábna kell lenniük, különben nem fog működni a program.

Ha ezekkel megvagyunk, akkor "qmake -project" paranccsal hozzunk létre egy .pro fájlt. Nézzük meg a nevét, majd a "qmake fajlnev.pro" paranccsal létrehozunk egy makefilet. Ezutám adjuk ki a make parancsot. Végül a megszokott ’./’-rel futtassuk és ezzel elkészült a nagyítónk.

5.6. Mandelbrot nagyító és utazó Java nyelven

Már láttuk hogyan zajlik C++-ban, most Java-ban kell életrekeltenünk.

```
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;

    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();
                    }
                }
            }
        });
        // Ablak tulajdonságai
        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() {
    // Az elmentendő kép elkészítése:
    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();
    }
}

 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;
            // z_0 = 0 = (reZ, imZ)
            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);
    }
}
```

Itt láthtaó a program. A "javac fájlnév.java" paranccsal futtassuk. Mostmár szemügyre vehetjük a mandelbrot halmazt, a nagyítások pedig külön ablakban jelennek meg.

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!

```
#ifndef POLARGEN_H
#define POLARGEN_H

#include <cstdlib>
#include <cmath>
#include <ctime>

class PolarGen
{
  public:
  PolarGen()
  {
    nincsTarolt = true;
    std::srand (std::time (NULL));
  }
  ~PolarGen ()
  {
  }
  double kovetkezo();

  private:
  bool nincsTarolt;
```

```
  double tarolt;
};

#endif

}
```

Itt láthatjuk a PolarGen osztály készítését. Igaz értéket adunk a nincsTarolt változónak, majd az srand függvénnyel fogjuk generálni a random számokat.

```
#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 kovetkezo függvény megnézi, hogy van-e már tárolt számunk, ha nincs akkor generálni fog kettőt és visszaadja az egyiket, ha már van eltárolva, akkor azt adja vissza.

Java-ban:

```
public class PolárGenerátor {
  boolean nincsTárolt = true;
  double tárolt;

  public PolárGenerátor() {
    nincsTárolt = true;
  }
```

```
  public double következő() {
    if (nincsTárolt) {
      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);
      tárolt = r*v2;
      nincsTárolt = !nincsTárolt;
      return r*v1;
    } else {
      nincsTárolt = !nincsTárolt;
      return tárolt;
    }
  }

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

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

} BINFA, *BINFA_PTR;
```

Létrehozunk egy struktúrát, amely áll egy értékből és két mutató a gyermekeknek.

```
BINFA_PTR
uj_elem ()
{
```

```
  BINFA_PTR p;

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

Ez a függvény memóriát foglal egy BINFA változónak, és hibával tér vissza ha a változó üres.

Mielőtt hozzálátunk a `main`-hez, deklarálunk néhány függvényt, amiket később definiálunk.

```
extern void kiir (BINFA_PTR elem);
extern void ratlag (BINFA_PTR elem);
extern void rszoras (BINFA_PTR elem);
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;
```

Létrehozzuk a gyökeret és az értékét '/'-re állítjuk, ez lesz a gyökér jelölése. Mivel még csak a gyökerünk van meg, ezért nincs bal vagy jobb gyermeke, így ezek értékét NULL-ra állítjuk.

```
while (read (0, (void *) &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;
      }
  }
    }
```

Alkossuk meg a binfát. A while ciklusban a standard inputról fogunk beolvasni, bitenként. Megnézzük, hogy a bemenet 0-e. Ha 0, és nincs nullásgyermek, akkor létrehozunk egyet és a fa mutatót a gyökérre állítjuk. Ha 0, és van nullásgyermek, akkor a fa mutatót a bal oldali gyermekre állítjuk. Amennyiben a bemenet nem 0, mégnézzük, hogy van-e jobb oldali gyerek. Ha nincs, akkor létrehozzuk és a fa mutatót a gyökérre állítjuk. Ha van, akkor a fa mutató a jobb oldali gyermekre állítjuk.

A program fordítása, futtatása, és eredménye:

```
$ g++ z.c -o z
$ ./z <bin.txt

---------1(2)
------------0(3)
------1(1)
---/(0)
---------1(2)
------0(1)
---------0(2)
melyseg=3
altag=2.333333
szoras=0.577350
```

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

Az inorder bejárás azt jelenti, hogy a viszgáladnó részfának először a bal oldali gyermekét dolgozzuk fel, majd a gyökerét, és végül a jobb oldali gyermeket.

A preorder bejárás során -ha nem üres a fa- először a gyökeret dolgozzuk fel, utána a bal oldali gyermeket, majd a jobb oldalit.

```
void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
      max_melyseg = melyseg;
      for (int i = 0; i < melyseg; ++i)
      printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
          ,
        melyseg);
      kiir (elem->bal_nulla);
      kiir (elem->jobb_egy);
      --melyseg;
    }
}
```

A postorder bejárás -ha nem üres a fa- a részfa bal oldali gyökerével kezdi a feldolgozst, a jobb oldali követi és legvégül dolgozzuk fel a gyökeret.

```
void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
      max_melyseg = melyseg;
      kiir (elem->bal_nulla);
      kiir (elem->jobb_egy);
      for (int i = 0; i < melyseg; ++i)
      printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
          ,
        melyseg);
      --melyseg;
    }
}
```

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:

A megoldás teljes forrása maga a z3a7.cpp

Az osztály definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ez lesz a beágyazott Csomopont osztály. Külön nem szánunk neki szerepet, ezzel is jelezzük, hogy csak a fa részeként számolunk vele. Tagfüggvényként túlterheljük a << operátort, ekkor így nyomhatjuk a fába az inputot: binFa << b; ahol a b egy '0' vagy '1' betű. Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett paraméterként" kapott) példány, azaz annak a fának, amibe éppen be akarjuk nyomni a b betűt, a tagjai (pl.: "fa", "gyoker") használhatóak a függvényben. A kiFile << binFa azt jelenti tagfüggvényként, hogy a kiFile valamilyen std::ostream stream osztály forrásába kellene beleírni ezt a tagfüggvényt, amely ismeri a mi LZW binfánkat (kiFile.operator<<(binFa)). Globális függvényként pedig: operator<<(kiFile, binFa). A paraméter nélküli konstruktor az alapértelmezett '/' "gyökérbetűvel" hozza létre a csomópontot, ilyet hívunk a fából, aki tagként tartalmazza a gyökeret. Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu" tagba, a két gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is megteszi. A bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen írjuk.

```
$ more z3a7.cpp
#include <iostream>
#include <cmath>
#include <fstream>

class LZWBinFa
{
public:

    LZWBinFa ():fa (&gyoker)
    {
    }
    ~LZWBinFa ()
    {
        szabadit (gyoker.egyesGyermek ());
        szabadit (gyoker.nullasGyermek ());
    }

    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 ();
            }
        }
    }

    void kiir (void)
    {
        melyseg = 0;
        kiir (&gyoker, std::cout);
    }

    int getMelyseg (void);
    double getAtlag (void);
    double getSzoras (void);

    friend std::ostream & operator<< (std::ostream & os, LZWBinFa & bf)
    {
        bf.kiir (os);
        return os;
    }
    void kiir (std::ostream & os)
    {
        melyseg = 0;
        kiir (&gyoker, os);
    }

private:
    class Csomopont
    {
    public:
        Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0)
        {
        };
        ~Csomopont ()
        {
        };

        Csomopont *nullasGyermek () const
        {
            return balNulla;
        }
```

```
    Csomopont *egyesGyermek () const
    {
        return jobbEgy;
    }

    void ujNullasGyermek (Csomopont * gy)
    {
        balNulla = gy;
    }

    void ujEgyesGyermek (Csomopont * gy)
    {
        jobbEgy = gy;
    }

    char getBetu () const
    {
        return betu;
    }

private:

    char betu;
    Csomopont *balNulla;
    Csomopont *jobbEgy;
    Csomopont (const Csomopont &);
    Csomopont & operator= (const Csomopont &);
};

Csomopont *fa;
int melyseg, atlagosszeg, atlagdb;
double szorasosszeg;
LZWBinFa (const LZWBinFa &);
LZWBinFa & operator= (const LZWBinFa &);

void kiir (Csomopont * elem, std::ostream & os)
{

    if (elem != NULL)
    {
        ++melyseg;
        kiir (elem->egyesGyermek (), os);
        for (int i = 0; i < melyseg; ++i)
            os << "---";
        os << elem->getBetu () << "(" << melyseg - 1 << ")" << std:: ↩
           endl;
        kiir (elem->nullasGyermek (), os);
        --melyseg;
    }
```

```
    }
    void szabadit (Csomopont * elem)
    {
        if (elem != NULL)
        {
            szabadit (elem->egyesGyermek ());
            szabadit (elem->nullasGyermek ());
            delete elem;
        }
    }

protected:

    Csomopont gyoker;
    int maxMelyseg;
    double atlag, szoras;

    void rmelyseg (Csomopont * elem);
    void ratlag (Csomopont * elem);
    void rszoras (Csomopont * elem);

};
```

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:

Az előző példában a gyökér objektumként volt jelen, most pointert csinálunk belőle. A famutató ráállítjuk a gyökérre és helyet foglalunk neki.

A gyökér többe nem objektum, hanem mutató lett, így gyemekek szabadítása során már nem '.'-ra, hanem '->'-ra lesz szülségünk(tehátgyoker.nullasGyermek helyett gyoker->nullasGyermek fog kelleni). Mivel a gyökérenk korábban helyet foglaltunk, most azt is fel kell szabadítanunk delete()-tel

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

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása: https://github.com/vargalz/prog1konyv/tree/master/hangya

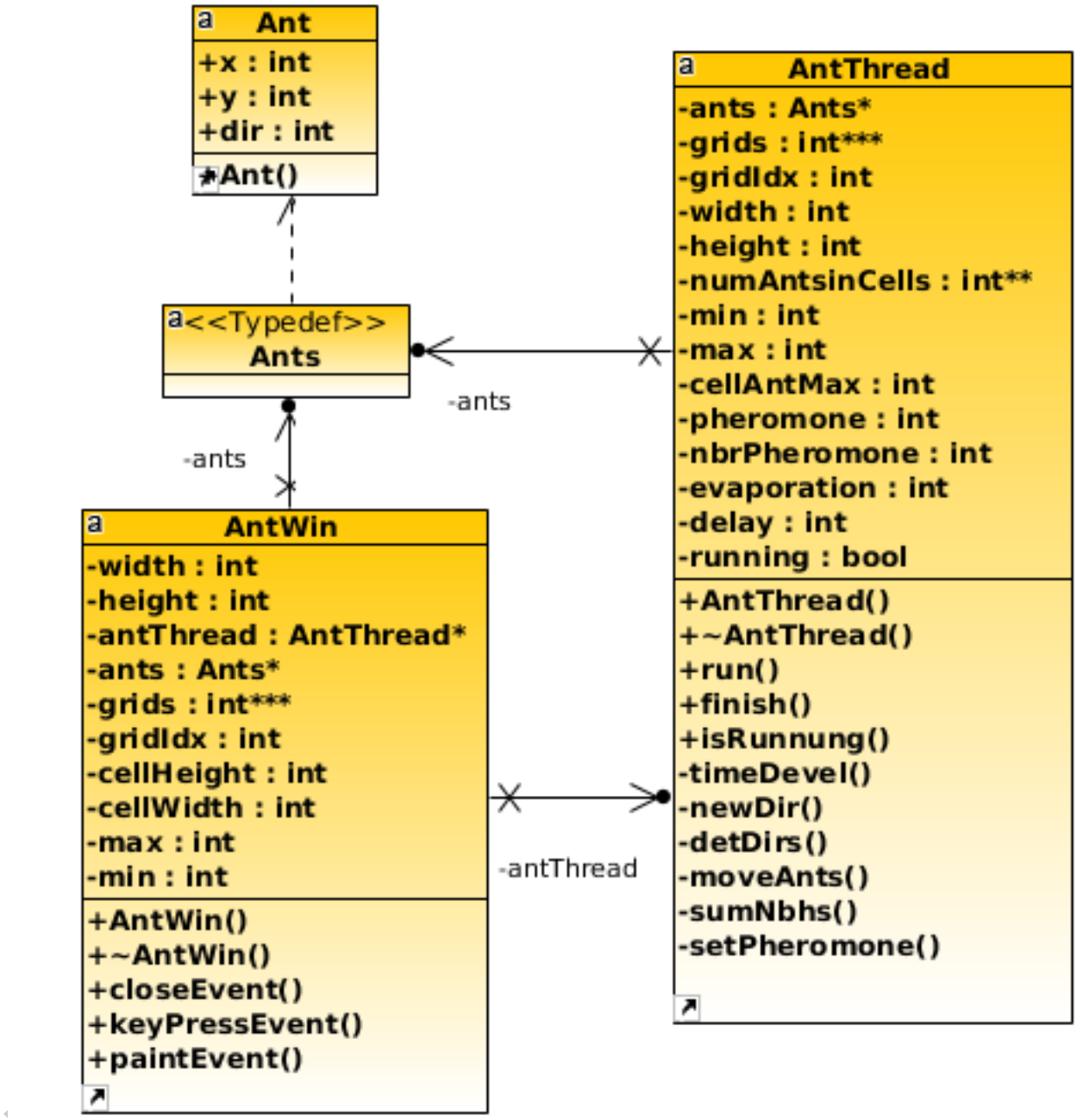
Tanulságok, tapasztalatok, magyarázat...

A hangyák feromonok kibocsájtásával kommunikálnak társaikkal. A szimuláció lényege, hogy a képernyőt cellákra osztjuk. Ezek a cellák rendelkezni fognak feromon erősséggel. Az egyes cellákbna lévő hangyák mindig a legerősebb feromonnal rendelkező szomszéd felé fognak menni.

A futtatáshoz szükségünk lesz a `main.cpp`, `antthread.cpp`, `antwin.cpp` fájlokra,az ezekhez tartozó header fájlokra és végül a `myrmecologist.pro`-ra. Ha egy mappában vannak a fentiek, akkor:

```
qmake myrmecologist.pro
make
./myrmecologist -w 250 -m 150 -n 400 -t 10 -p 5 -f 80 -d 0 -a 255 -i 3 -s 3 ↩
     -c 22
```

A program rendelkezik alapértékekkel, így a kapcsolók opcionálisak. De láthatjuk, hogy elég sok kapcsoló van, ezért vizsgáljuk meg, hogy melyik mit csinál. A `w` kapcsoló a cellák szélességét adja meg, az `m` pedig azok magasságát. Az `n` kapcsoló a hangyák számát határozza meg. A `t` kapcsoló a hangyák lépéseinek gyakoriságát szabályozza. A `p` kapcsoló a feromon párolgásának gyorsaságát állítjuk. Ha egy hangya belép egy cellába, akkor ott növeli fogja a feromon szintet, az `f` kapcsolóval pedig ezt tudjuk meghatározni. A `d` kapcsolóval az alapérteémezett feromonértéket adjuk meg a celláknak. Az `a` a maximális, az `i` a minimális feromont szablya meg a celláknak. Az `s` kapcsoló megmondja, hogy a hangyák mennyi feromont hagyjanak a cellákban. A `c` kapcsoló pedig szabályozza, hogy mennyi hangya lehet egyszerre egy cellében.



7.1. ábra. UML diagram

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A siklókilövő lényege, hogy a kezdeti sejteket meghatározott helyre rakjuk, utána pedig folyamatosam "lövi ki" a siklókat felfelé. A fehér cellák üresek, a feketék az elő sejtek. Van 3 szabály, amit be kell tartani: 1.: Egy sejt akkor marad életben,ha 2-3 szomszédja van. 2.: Egy sejt meghal ha 3-nál több szomszédja van (túlnépesedés), vagy ha 2-nél kevesebb szomszédja van (elszigetelődés). 3.: Egy sejt akkor születik, ha egy üres cellának 3 élősejt szomszédja van.

A fenti három szabály kód formájában láthatjuk, ezalapján változik az alakzatunk.

```
for(int i=0; i<rácsElőtte.length; ++i) { // sorok
            for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok

                int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);

                if(rácsElőtte[i][j] == ÉLŐ) {

                    if(élők==2 || élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
                }  else {

                    if(élők==3)
                        rácsUtána[i][j] = ÉLŐ;
                    else
                        rácsUtána[i][j] = HALOTT;
}
```

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

A siklókilövő grafikus felületen való megjelenítése a cél. Szükeség lesz 4 fájlra: `sejtablak.cpp`, `sejtablak.h`, `sejtszal.cpp` és `sejtszal.h`.

A szabályok nem változtak. Meghal a sejt, ha 2-nél kevesebb vagy 3-nál több szomszédja van, születik, ha a cella rendelkezik 3 élő szomszéddal, 2-3 szomszéddal pedig tovább élnek a sejtek zavartalanul.

```
void SejtAblak::siklóKilövő(bool **racs, int x, int y) {

    racs[y+ 6][x+ 0] = ELO;
    racs[y+ 6][x+ 1] = ELO;
    racs[y+ 7][x+ 0] = ELO;
    racs[y+ 7][x+ 1] = ELO;
```

```
    racs[y+ 3][x+ 13] = ELO;

    racs[y+ 4][x+ 12] = ELO;
    racs[y+ 4][x+ 14] = ELO;

    racs[y+ 5][x+ 11] = ELO;
    racs[y+ 5][x+ 15] = ELO;
    racs[y+ 5][x+ 16] = ELO;
    racs[y+ 5][x+ 25] = ELO;

    racs[y+ 6][x+ 11] = ELO;
    racs[y+ 6][x+ 15] = ELO;
    racs[y+ 6][x+ 16] = ELO;
    racs[y+ 6][x+ 22] = ELO;
    racs[y+ 6][x+ 23] = ELO;
    racs[y+ 6][x+ 24] = ELO;
    racs[y+ 6][x+ 25] = ELO;

    racs[y+ 7][x+ 11] = ELO;
    racs[y+ 7][x+ 15] = ELO;
    racs[y+ 7][x+ 16] = ELO;
    racs[y+ 7][x+ 21] = ELO;
    racs[y+ 7][x+ 22] = ELO;
    racs[y+ 7][x+ 23] = ELO;
    racs[y+ 7][x+ 24] = ELO;

    racs[y+ 8][x+ 12] = ELO;
    racs[y+ 8][x+ 14] = ELO;
    racs[y+ 8][x+ 21] = ELO;
    racs[y+ 8][x+ 24] = ELO;
    racs[y+ 8][x+ 34] = ELO;
    racs[y+ 8][x+ 35] = ELO;

    racs[y+ 9][x+ 13] = ELO;
    racs[y+ 9][x+ 21] = ELO;
    racs[y+ 9][x+ 22] = ELO;
    racs[y+ 9][x+ 23] = ELO;
    racs[y+ 9][x+ 24] = ELO;
    racs[y+ 9][x+ 34] = ELO;
    racs[y+ 9][x+ 35] = ELO;

    racs[y+ 10][x+ 22] = ELO;
    racs[y+ 10][x+ 23] = ELO;
    racs[y+ 10][x+ 24] = ELO;
    racs[y+ 10][x+ 25] = ELO;

    racs[y+ 11][x+ 25] = ELO;

}
}
```

Ez az eljárás létrehozza a siklókillövőnket

7.4. BrainB Benchmark

Megoldás videó:

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

Tanulságok, tapasztalatok, magyarázat...

A benchmarknak célja, hogy felmérje az ember kognitív képességeit. Az játéknak a lényege az, hogy a kurzort rajta kell tartanunk a Samu Entropy-n, amíg egyre több New Entropy fog megjelenni, ha elveszítjuk Samut, a New Entropy szaporodás lelassul, így könnyebben megtalálhatod. A teszt 10 perces. Minél bonyolultabb képet kapsz a végén, annál jobb vagy.

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...

Az MNIST egy adatbázis, ami kézzel írott számokat és karaktereket tartalmaz. Gyakran használják képfelismerő programok tanítására.

A feladathoz szükségünk lesz a TensorFlow köyvtárra, amit gépi-tanuláshoz használnak. Telepítése:

```
sudo apt update
sudo apt install python3-dev python3-pip
sudo pip3 install -U virtualenv
sudo apt-get install python3-tk
virtualenv --system-site-packages -p python3 ./venv
source ./venv/bin/activate
pip install --upgrade tensorflow
python -c "import tensorflow as tf; tf.enable_eager_execution(); print(tf. ↩
   reduce_sum(tf.random_normal([1000, 1000])))"
pip install matplotlib
```

Ha a telepítés megvan, lássuk a programot.

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse

# Import data
from tensorflow.examples.tutorials.mnist import input_data
```

```
import tensorflow as tf
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

import matplotlib.pyplot
```

```
def main(_):
  mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])
  W = tf.Variable(tf.zeros([784, 10]))
  b = tf.Variable(tf.zeros([10]))
  y = tf.matmul(x, W) + b

  # Define loss and optimizer
  y_ = tf.placeholder(tf.float32, [None, 10])

  # The raw formulation of cross-entropy,
  #
  #   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
  #                                 reduction_indices=[1]))
  #
  # can be numerically unstable.
  #
  # So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
  # outputs of 'y', and then average across the batch.
  cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits( ↩
     labels = y_, logits = y))
  train_step = tf.train.GradientDescentOptimizer(0.5).minimize( ↩
     cross_entropy)

  sess = tf.InteractiveSession()
  # Train
  tf.initialize_all_variables().run(session=sess)
  print("-- A halozat tanitasa")
  for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    if i % 100 == 0:
      print(i/10, "%")
  print("------------------------------------------------------------")

  # Test trained model
  print("-- A halozat tesztelese")
  correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  print("-- Pontossag: ", sess.run(accuracy, feed_dict={x: mnist.test. ↩
     images,
                                      y_: mnist.test.labels}))
```

```
  print("------------------------------------------------------------")

  print("-- A MNIST 42. tesztkepenek felismerese, mutatom a szamot, a  ↩
     tovabblepeshez csukd be az ablakat")

  img = mnist.test.images[42]
  image = img

  matplotlib.pyplot.imshow(image.reshape(28, 28), cmap=matplotlib.pyplot.cm ↩
     .binary)
  matplotlib.pyplot.savefig("4.png")
  matplotlib.pyplot.show()

  classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

  print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
  print("------------------------------------------------------------")

  #print("-- A sajat kezi 8-asom felismerese, mutatom a szamot, a  ↩
     tovabblepeshez csukd be az ablakat")
  print("-- A MNIST 11. tesztkepenek felismerese, mutatom a szamot, a  ↩
     tovabblepeshez csukd be az ablakat")
#  img = readimg()
#  image = img.eval()
#  image = image.reshape(28*28)
  img = mnist.test.images[11]
  image = img
  matplotlib.pyplot.imshow(image.reshape(28,28), cmap=matplotlib.pyplot.cm. ↩
     binary)
  matplotlib.pyplot.savefig("8.png")
  matplotlib.pyplot.show()

  classification = sess.run(tf.argmax(y, 1), feed_dict={x: [image]})

  print("-- Ezt a halozat ennek ismeri fel: ", classification[0])
  print("------------------------------------------------------------")

if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/ ↩
     mnist/input_data',
                      help='Directory for storing input data')
  FLAGS = parser.parse_args()
  tf.app.run()
```

A program az adatbázis 28x28-as képeit fogja elemezni és tanítani magát.

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...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

Lisp-ben a kifejezések prefix formáját használjuk, azaz az operátor a két operandus előtt szerepel. Például ha az 1+1-et szeretnénk megadni akkor (+ 1 1)-et kellene írnunk.

Itt látható a faktoriális iteratív, azaz ismétlő változata.

```
(define (faktorialis n)
        (define (iter product counter)
                (if (> counter n)
                        product
                        (iter (* counter product) (+ counter 1))
                ))
        (iter 1 1))

(display (faktorialis 3))
```

Ez a 3!-t fogja megadni.

A rekurzív algoritmusok lényege, hogy önmagukat hívják meg. A faktoriális rekurzív verziója:

```
(define (faktorialis n)
        (if (= n 1)
                1
                (* n (faktorialis (- n 1) ))
        ))

(display (faktorialis 3))
```

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...

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

Alapfogalmak

A programozási nyelveknek három szintjük van. Gépi nyelv, assembly szintû nyelv és magas szintû nyelv. A magas szinten írt program forrásprogramnak hívjuk, az ezekre vonatkozó szabályokat pedig szintaktikai szabálynak nevezzük. A nyelvtani és formai szabályok a szintaktikai szabályok. Ahhoz, hogy egy proceszor értelmezni tudja a kódunkat, interpreterre vagy fordítóprogramra van szüksége. Mindkettõ végez szintaktikai és lexolális elemzést. A szintaktikai elemzés megnézi, hogy teljesülnek-e a a szintaktikai szabályok, a lexikális elmezás pedig felbontja a forrást lexikális egységekre. Hogy helyesen írjunk meg egy programot tisztában kell lennünk a nyelv szabványaival. Napjaink problámja a nyelvek hordozhatósága, azaz egy magas szintû nyelv másik magas szintû nyelvbe való implementációja

Adattípusok

Az adattípus konkrét programozási eszközök komponenseként jelenik meg. Meghatározó dolog az adattípusok világában a reprezentáció, azaz egyfajta belső ábrázolási mód. Saját reprezentációt megadni azonban csak nagyon kevés programozási nyelvben lehet. Az adattípusok két nagy csoportja az egyszerű és az összetett adattípusok.

Változók

Az imperatív programozási nyelvek fő eszköze, négy komponense van: név, attribútumok, cím, érték. A változó mindig a nevével jelenik meg, a másik három komponenst a névhez rendeljük hozzá.Az eljárásorientált nyelvek leggyakoribb utasítása az értékadó utasítás (pl.: C)

Kifejezés

Szintaktikai eszköz, két komponense az érték és a típus. Összetevői: operandusok, operátorok, és a kerek zárójelek. A kifejezés kiértékelésének nevezzük azt a folyamatot, amikor a kifejezés értéke és típusa meghatározódik.

Utasítások

Olyan egységek, amelyekkel megadjuk az algoritmusok lépéseit és amely segítségével a fordítóprogram tárgyprogramot generálhat. Két nagy csoport: deklarációs és végrehajtható utasítások.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

Ez a könyv az alapoktól indít. Megismerkedünk a változók fogalmával és típusaival (pl.: int, double, char), méretével. Ezekből az egyszerű típusokból épülnek fel a tömbök és struktúrák. Ezenkívül megismerkedünk a - for és while- ciklusokkal. Bemutatja az aritmetikai és logogikai operátorokat, a deklarációt. Inkrementálás éés dekrementálás.

Ezek után a vezérlési szerkezeteket ismerteti a könyv. If-else és fodrdítva, break, goto, címkék és switch.

Egyszerű függvények és az azokhoz tartozó fogalmak, külső változók, a header állomány, Változó inicializálás és statikus változó. Blokkstruktúrák és rekurzió. Ezekről mind szót ejt a negyedik fejezet.

Megismerkedünk a mutatókkal, ami egy C nyelvet tanító könyhöz elmaradhatatlan. Mutatótömbök, mutatókra mutató mutatók és többdimenziós tömbök bemutatása. Függvények és struktúrák bemutatása. Struktúratömbök,önhivatkozó struktúra, typedef parancsok.

Az adat be- és kivitelről is tanulhatunk a könyvből. Először természetesen a standard inputtak és outputtal ismerkedünk meg. Majd kitér a printf() és scanf() függvénykre és használatára. Mivel szó van a beolvasásról, így szót ejt a fejezet a hibakezelésről is a standard error(stderr) és exit formájában.

10.3. Programozás

[BMECPP]

A C++ tulajdonképpen a C nyelv továbbfejlesztése, ami az objektumorientált porgramozást tette lehetõvé. A nyelv atyjának Bjarne Stroustrup-t tekintjük, mivel õ fejlesztette ki a ’80-as években. Ami C kód az C++ is egyben, de ez fordítva már nem igaz. Azt is mondhatnánk, hogy C nyelv részhalmaza a C++-nak.

C++-ban a main függvényt megadhatjuk simán ’int main()’-ként, de ’int main (int argc, char* argv[])’-ként is. Az argc parancssori argumentumok száma, az argv pedig a maga a parancssori argumentum. Ha a main függvényünk végén elhagyjuk a return-t, akkor a fordító automatikusan ’return 0’-nak értelmezi majd.

Nézzük meg a boolean típusú változókat. Ilyen típus sima C-ben még nem létezett, ezért int vagy enum típust használunk helyette. Ezek két értéket vehetnek fel: true, false.

C++-ban lehetõségünk nyílik a változók deklarálására egy ciklus megadása közben, ám ez csak a ciklus keretein belül fog élni.

C-ben nem tehettük meg, hogy két függvénynek ugyanazt a nevet adjuk. A C++-szal más a helyzet, mivel a függvény itt már a függvény neve és a hozzátartozó argumentumlista azonosítja be a függvényt. C++-ban adhtunk a függvényeinknek alapértelmezett értékeket.

Amíg C-ben a paraméterátadás érték szerint történt, C++-ban ez referenciatípussal történik. Nem szabad összekevrnünk a mutatót és a refernciát. A mutató egy memóriacímet kap, aminek segítségével hozzáférünk a változóhoz és változtathtajuk azt.

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.