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	2019. március 28.	

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

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

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 forrása:

Kód:

```
#include <stdio.h>

int main(){
while(1){
  }
}
```

Magyarázat: 1.Egy szál 100%-osan futtatása: A main függvényben lévő while ciklus mindaddig lefog futni, amíg a while utáni zárójelben lévő feltétel igaz lesz. Jelen esetben a feltételnek '1' van beírva, ezt a gép mindig igaznak tekinti(a 0-át pedig hamisnak). Ezért ez a ciklus mindaddig futni fog amíg manuálisan ki nem iktatjuk.Ez a ciklus 1 szálat fog folyamatosan 100%-on futtatni. Kód:

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

int main(){
while (1){
sleep(1);
  }
}
```

Magyarázat: 2.Egy szál 0%-on való futtatása: Az előzőhöz képest itt is egy végtelen while ciklusunk van a main függvényben, azonban itt megadunk egy sleep nevű utasítást is. A sleep működéséhez include-olnunk kell az unistd.h nevezetű header file-t is. A sleep függvény utáni zárójelben megadhatjuk,hogy az adott szálat hány másodpercig altassa a program.Ezt használhatjuk még programok késleltetésére is. Kód:

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

int main () {

#pragma omp parallel

  while (1) {
}
}
```

Magyarázat: 3.Minden szál futtatása Ahhoz, hogy minden szálat lefoglaljon a programunk, párhuzamossá kell tennünk a program futását. Ehhez include-oljuk az openmp(Open Multi-Processing) alkalmazásprogramozási felületet(API-t). Ezt az omp.h header file-al tehetjük meg.Ezáltal a programunk párhozamosan fog futni több szálon.A main függvényünk kibővült a #pragma omp parallel sorral.Ez a sor fogja a while ciklusban lévő utasításokat az összes szálra irányítani.Így a végtelen ciklusunk az összes szálon fog egyszerre futni. Ahhoz,hogy ez a program lefordúljon használnunk kell a -fopenmp kapcsolót is.

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... Ha T1000-ben meghívjuk saját magát, és van benne végtelen ciklus akkor a Lefagy függvény igazat ad vissza ha nincs akkor pedig hamisat.A Lefagy2 pedig igaz értéknél igazat ad vissza, hamisnál pedig belép egy végtelen ciklusba,tehát ha nem is vlt benne végtelen ciklus most,már lesz így ellentmondás jön létre.Tehát ilyen program tényleg nem létezhet.

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:

Kód:

```
    #include <stdio.h>



int main()
{
    int a=7;
    int b=1;

    printf("Valtozok csere elott\n");

    printf("%d\n",a);
printf("%d\n",b);


    a=a+b;
    b=a-b;
    a=a-b;

     printf("Valtozok csere utan\n");
 printf("%d\n",a);
printf("%d\n",b);




    return 0;
}
```

Magyarázat:Változók cseréje különbséggel: A main függvényünk elején deklarálunk két int típusú változót(a,b). Jelen esetünkben a=7 és b=1.A printf függvény segítségével tudunk a konzolra kiíratni.A printf utáni zárójelben idézőjelek közé írhatjuk be amit ki szeretnénk íratni. A %d kapcsoló azt adja meg, hogy az utána írt 'a' változót egész számként fogja kiírni. A \n kapcsoló pedig a sortörés. Maga a változó értékének cseréje egyszerű matematika. Az 'a' értéke kezdetben 7, 'b' értéke pedig 1. Első lépésként 'a' értékét

egyenlővé tesszük ’a’ és ’b’ összegével,így a=7+1=8.Második lépésként ’b’ értékét egyenlővé tesszük ’a’ és ’b’ különbségével, így b=8-1=7. Majd utolsó lépséként ’a’ értékét egyenlővé tesszük ’a’ és ’b’ különbségével, azaz a=8-7=1. Amint látjuk ’a’ értéke 1 lett ’b’ értéke pedig 7, tehát megcserélődtek. Kód:

```
include <stdio.h>



int main()
{
    int a=7;
    int b=1;

   printf("Valtozok csere elott\n");

    printf("%d\n",a);
printf("%d\n",b);

    a=a*b;
    b=a/b;
    a=b/a;

    printf("Valtozok csere utan\n");
 printf("%d\n",a);
printf("%d\n",b);
    return 0;
}
```

Magyarázat: Ez a példa annyiban különbözik az előzőtől,hogy itt az összeadás és különbség helyett szorzást és osztást használunk.Az ’a’ értéke kezdetben megint csak 7, ’b’ értéke pedig 1. Első lépésként ’a’ értékét egyenlővé tesszük ’a’ és ’b’ szorzatával, a=7*1=7. Második lépésként ’b’ értékét egyenlővé tesszük ’a’ és ’b’ hányadosával,b=7/1=7. Majd utolsó lépésként ’a’ értékét egyenlővé tesszük ’b’ és ’a’ hányadosával azaz a=7/7=1.Ezek a változó cserék a régebbi időkben voltak hasznosak, ugyanis így nem kellett felesleges erőforrásokat lefoglalni egy segédváltozónak.

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:

Kód:

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

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

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;

    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );
 mvprintw ( y, x, "0" );

        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;
        }

    }
 return 0;
}
```

Magyarázat: A programunk futásához include-olnunk kell a curses.h nevezetű header file-t, melyben a különböző képernyőkezeléshez szükséges függvények találhatóak. A main függvényünk elején a WINDOW paranccsal adjuk meg az ablak ábrázolását, és a későbbiekben ablak néven érjük el.Az initscr függvénnyel "tisztítjuk" le a konzolt.Ezután deklaráljuk az x és y koordinátákat melyek kezdőértékei 0 lesz.Utána az xnov és ynov deklarálása következik melyeknek értékei 1, ezekkel fogjuk növelni a koordinátákat, azaz léptetni a labdánkat.A pattogásunk alapja egy végtelen for ciklus, ezért ez addig fog futni amíg manuáli-

san ki nem lőjük. A getmaxyx függvény meghatározza a konzolunk maximális koordinátáit.Az mvprintw függvénnyel fogunk x és y koordinátákra jelen esetben egy ’0’-t írni.Az usleep mögötti zárójelbe beírt szám fogja meghatározni a pattogás gyorsaságát. Minél nagyobb számot írunk be annál lassabb lesz, ugyanis ez a parancs altatja a ciklusunkat a megadott ideig. Ezután az x és y értékét növeljük xnov-el és ynov-el(jelen esetben 1 el). Az ezután következő 4 db if utasítás segít nekünk abban,hogy labdánk a konzolon belül maradjon.Ha elérte a bal, vagy jobb oldalt, akkor az xnov változó érétkét beszorozzuk -1 el, tehát előjelet fog váltani, ezáltal a következő körben csökkeni fog x értéke, tehát visszapattan a labdba. Ha pedig az ablak tetejét, vagy alját érte el akkor ugyanezt csináljuk meg csak az ynov változó értékével.

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:

Kód:

```
          #include <stdio.h>


int main()
{
    int a=1;
    int count=0;
do
count++;

    while (a <<= 1) ;



printf("%d", count);
printf("bites a szohossz");
    return 0;
}}
```

Magyarázat: A fenti programunk számítógépünk int típusú változójának méretét fogja megadni.A main függvényben deklarálunk egy ’a’ változót és értékét 1 re állítjuk.Ennek segítségével fogjuk mérni a szóhosszt. Valamint deklarálunk egy count nevezetű változót is, melyet majd növelünk és a programunk végénk kiíratjuk, ez lesz az int hossza.A main fő része egy do while ciklus. Ez, a sima while-al ellentétben hátultesztelős utasítás, tehét először fogja növelni a count értékt, és csak azután fogja letesztelni a while utáni feltételt. Ez azért fontos mert így tudjuk elérni,hogy az első shift is bele számítson az eredményünkbe.A szóhosszt az úgy nevezett bitshift operátor segítségével állapítjuk meg.Jelen példánkban a balra shiftelőt használjuk, de van jobbra shiftelő is. Ez az operátor annyit fog tenni,hogy a memóriában eltárolt biteket eggyel balra fogja eltolni és jobbra kiegészíti egy nullával.Tehát például az ’a’ értéke egy, ez kettes számrendszerben van eltárolva a memóriában ennek bináris kódja 0001. A bitshift ezt fogja eggyel eltolni tehát

0010 lesz, ami már a 2-es szám. Majd 0100 lesz, ami a 4-es.És mindenegyes shiftnél növeljük a count értékét eggyel.Ezt mindaddig fogja így csinálni amíg az adott memóriacímben csak 0-ások lesznek, tehát az értéke 0 lesz.

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:

Kód:

```
#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)
{
  double osszeg = 0.0;
  int i;

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

  return sqrt(osszeg);
}

int
main (void)
{

  double L[4][4] = {
    {0.0, 0.0, 1.0 / 3.0, 0.0},
    {1.0, 1.0 / 2.0, 1.0 / 3.0, 1.0},
    {0.0, 1.0 / 2.0, 0.0, 0.0},
    {0.0, 0.0, 1.0 / 3.0, 0.0}
  };

  double PR[4] = { 0.0, 0.0, 0.0, 0.0 };
```

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

  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;
}
```

Magyarázat:A PageRank a Google keresőmotorjának legfontosabb eleme.Minél többen hivatkoznak egy oldalra, az annál jobb. A fenti C program négy lap PageRank értékét számolja ki.A kódunkhoz szükséges a math.h header file include-olása,ebben matematikai függvények találhatóak, pl.sqrt.A kód elején található egy void típusú függvény.A void annyit takar, hogy a függvény nem fog visszatéríteni semmilyen értéket.A függvény neve után zárójelek között adjuk meg a függvény paramétereit, jelen esetben a függvény egy double típusú tömböt, illetve egy int típusú változót vár. Ez a függvény egy for ciklus segítségével fogja kiíratni a konzolra a kapott tömb elemeit.A következő függvény már double típusú,tehát egy double típusú változót fog visszaadni.Paraméterként pedig két tömböt illetve egy egész számot vár.Ez a függvény fogja számolni a távolságot.A két tömb elemeit egyesével kivonja majd lényegében négyzetre emeli. Majd a függvény végén gyököt vonunk az sqrt segítségével, ezáltal biztos,hogy pozitív eredményt kapunk.A main függvényben lévő 'L' nevezetű mátrix mondja meg,hogy melyik oldalra hányan mutatnak linkekkel.A main függvény fő része egy végtelen for ciklus.A számítást egy másik for ciklussal végezzük el, ez egy sima mátrix szorzás. Ezután az így kapott eredményt átadjuk a tavolsag függvénynek, és ha az így visszakapott érték kisebb mint 0.00000001 akkor megszakítjuk a ciklust és átadjuk az értékeket kiíratásra.

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

A Brun tétel kimondja,hogy az ikerprímek(olyan prímpárok melyeknek különbsége 2) reciprokösszege egy véges értékhez konvergál.A következő kód ennek a tételnek R-beli megvalósítása.

```
library(matlab)

stp <- function(x){

    primes = primes(x)
    diff = primes[2:length(primes)]-primes[1:length(primes)-1]
    idx = which(diff==2)
    t1primes = primes[idx]
    t2primes = primes[idx]+2
    rt1plust2 = 1/t1primes+1/t2primes
    return(sum(rt1plust2))
}

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

Ahhoz,hogy kódunk működjön telepítenünk kell a matlab csomagot, ezt megtehetjük az intsall.packages("matlab") parancssal.Ezután ezt a csomagot a library(matlab) parancssal be is töltjük. Ezek után stp néven létrehozunk egy függvényt amely paraméterül kap majd egy számot.primes néven létrehozunk egy vektort amelyben eltároljuk a paraméterként kapott számig a prímszámokat. Ezután a diff vektorban kiszámoljuk a prím párok különbségét, ezt úgy tesszük meg,hogy először eltároljuk a primes vektor második tagjától a vektor végéig a prímeket, majd ezekből kivonjuk a primes vektor első tagjától az utolsó előtti tagjáig a prímeket.A következő sorban az idx vektorban eltároljuk azoknak a vektoroknak az indexét ahol a különbség kettő(tehátikerpímek).A t1primes vektorban eltároljuk az így kapott prímpárok első tagját, a t2primes-ban pedig a párok második tagját.Az rt1plust2 vektorba kiszámoljuk a két vektor reciprokösszegét, majd return-el ezeknek az összegüket íratjuk ki. Az utolsó három sorral pedig ki is rajzoltatjuk az így kapott eredményt.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

A Monty Hall paradoxon egy az USA-ban futó televíziós vetélkedőn alapul.A játékost megkéri a műsorvezető, hogy válasszon 3 csukott ajtó közül. A három ajtó közül az egyik mögött egy nagyon értékes nyeremény van, a másik kettő mögött pedig valami nagyon értéktelen. A játékos választása után a műsorvezető a másik két ajtó közül kinyit egyet ami mögött biztosan nincsen az értékes nyeremény. Ezután a műsorvezető megkérdezi a játékost,hogy szeretné-e a másik ajtót választani,vagy marad az eredeti döntésénél.És itt jön a paradoxon. Ugyanis az első választásnál a játékosnak 1/3 esélye volt arra,hogy az értékés nyereményt választja.Most a józan ész azt sugallja,hogy mindegy melyik ajtót választja, úgyis 50-50% esélye van a nyereményre,de igazából 2/3 esélye lenne a nyereményre ha átvált a másik ajtóra.Ezt talán könnyebb elfogadni ha elképzeljük ugyanezt csak 100 ajtóval. Tehát a játékos választ egyet,1/100 az esélye arra,hogy

eltalálja a nyereményt és 99/100 az esély arra,hogy valamelyik másik ajtó mögött van. Ezután a játékvezető kinyit 98 ajtót ami mögött nincs semmi és felteszi ugyanazt a kérdést:Váltunk-e ajtót?Talán így könnyebb belátnunk,hogy sokkal nagyobb esélyünk van nyerni ha váltunk a másik ajtóra, ugyanis az az eredeti 99/100 esély oda koncentrálódik.

A következő kód ennek a problémának R szimulációja:

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

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)]

}

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

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)


sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

Kódunk elején a kiserletek_szama vektorban mentjük el,hogy hányszor hajtuk végre a szimulációt.Ezután a kiserlet vektorba tároljuk el,hogy melyik ajtó mögött lesz a nyeremény, 1-3 ig fog egy számot random választani annyiszor ahányszor szimuláljuk.A jatekos vektorban ugyanezt csináljuk, ez fogja megadni a játékos választását.A musorvezeto vektor azt az ajtót fogja tárolni amit a műsorvezető fog kinyitni. Ezt

úgy határozzuk meg, hogy indítunk egy for ciklust ami végigmegy az összes létrejött eseten. Ezután egy if segítségével először megnézzük azt az esetet amikor a játékos eltalálta a nyereményt rejtő ajtót, azaz a kiserlet és a jatekos vektor egyenlő. Ha ez az eset áll fent akkor a maradék két ajtó közül random választ egyet a program. Ha pedig nem egyezik meg a választása a játékkosnak a nyereményt rejtő ajtóval akkor, azt a két ajtó kivesszük a választásból és a maradék egy ajtót fogja kinyitni a műsorvezető.Ezután a nem-valtoztatesnyer vektorban eltároljuk azokat az eseteket amikor akkor nyerne a játékos ha nem változtat. A valtoztat vektorban pedig azokat az eseteket amikor akkor nyerne amikor változtat. Ezután kiíratjuk ezeket.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

```
        #include <stdio.h>

int main(){
int sz=0;
int a=0;
printf("Adj meg egy számot");
scanf("%d", &sz);
printf("Unarisba atvaltva:");
for(int i=0;i<sz;i++)
{
printf("%d",1);
       }

return 0;
}
```

A fent látható program,egy decimális számot fog bekérni a konzolról és azt a számot unáris számrendszerbe fogja kiírni.Az unáris számrendszerbe úgy váltunk át,hogy annyiszor fogunk kiírni egy db egyest amennyi a szám értéke. Tehát pl ha beírjuk ,hogy 5 akkor 11111-et fog kiírni.Ezt egy egyszerű for ciklussal tesszük meg. A Turing gép úgy végezné ezt az átváltást,hogy a szám végéről indulva, ha 0-át lát, akkor kilencre vált, és mindig ki vonna egy számot, tehát 8,7,6,5,4,3,2,1,0 és a kivont egyeseket eltárolja, majd a folyamat végén kiírja őket.

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

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

```
S, X, Y legyenek változók
a, b, c legyenek konstansok

S->abc, S->aXbc, Xb->bX, Xc->Ybcc, bY->Yb, aY->aaX, aY->aa

S (S->aXbc)
aXbc (Xb->bX)
abXc (Xc->Ybcc)
abYbcc (bY->Yb)
aYbbcc (aY->aa)
aabbcc

de lehet így is:

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 legyenek változók
a, b, c legyenek konstansok

A->aAB, A->aC, CB->bCc, cB->Bc, C->bc

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

de lehet így is:

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
```

A generatív grammatika különböző transzformációs szabályokat tartalmaz. Ezekhez szükségünk van egy alap szimbólumra, és ezen a szimbólumon tetszés szerint bármelyik szabályt bármennyiszer elvégezhetünk.Az a lényege, hogy egy fajta szimbólumot egy féleképpen tudunk létrehozni, ha több féleképpen is elő tud állni akkor ez a nyelv már nem generatív.

Először megadunk konstansokat és változókat is . Ezután leírjuk a megfelelő nyelvtani szabályokat. Ezt nyilak segítségével tehetjük meg.Bal oldalon megadjuk hogy miből, jobb oldalon meg hogy mi lesz belőle.Ezt rövidebben is láthatjuk alatta. A második generatív nyelv létrehozásánál lényegében elég volt a változók nevét átírni a változók neveit, és máris egy másik generatív nyelvet kapunk amely ugyanazt a nyelvet generálja.

3.3. Hivatkozási nyelv

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

```
          #include <stdio.h>

int main(){
for(int i=1;i<10;i++)
printf("lefut");
}}
```

1989-ben adták ki a C c89 es szabványát és 1999-ben a c99-et.Ezekben a szabványokban különböző szemantikai szabályok érvényesek. A fenti kódban egy ilyen különbségre láthatunk példát. Első ránézésre valószínű fel sem tűnik benne semmi érdekes.Egy sima for ciklus benne egy utasítással. Azonban a for ciklust a c89-es szabvány szerint így nem adhatjuk meg,ugyanis ebben még tilos volt a zárójelek között deklarálni a ciklusváltozót.Ahhoz,hogy ezt lássuk, a megadott szabvány szerint kell lefordítanunk a programot.Ezt a -std=gnu89 kapcsolóval tehetjük meg.Ez a c89-es szabvány szerint fogja lefordítani a programunkat.Így meg is kapjuk azt a hibakódot miszerint c99 szerint kellenne fordítanunk a programot mert amit próbálunk csinálni az eszerint a szabvány szerint nem megengedett.

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!

Kód:

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

```
 int  szamok_szama =0;
%}
%%
[0-9]+    {++szamok_szama;}



%%

int
main()
{
 yylex();
 printf("%d szam\n",szamok_szama);
return 0;

}
```

Magyarázat:A fent látható kód, a bemenetén megjelenő számokat fogja összeszámolni egy lexer segítségével.Egy lexer a bemenetén megjelenő szövegeket vizsgálja, és megadhatunk neki különböző mintákat,karaktereket amiket keres.A programot három fő részre lehet bontani, ezeket a fő részeket a % jelek határolják. Az első fő részben adhatjuk meg a különböző definíciókat,jelen esetben egy egész típusú változót hoztunk létre. A második részben adhatjuk meg,hogy mit keressen a lexer, és ha talált olyan karaktert,mit tegyen vele.Jelen esetünkben az egész számokat fogja keresni, és ha talál egyet, akkor a változónk értékét növeli eggyel. A harmadik részben pedig megadjuk, hogy a program végén írassa ki a változó értéket. Ezt a programot először egy .l kiterjesztésbe kell elmentenünk. Fordítása két lépésben történik. Először flex valami.l, ez a parancs létrehoz egy valami.yy.c kiterjeszésű fájlt,majd ezt a fájlt kell c fordítóval lefordítanunk a -lfl kapcsoló segítségével.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

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

%}
%%
"3" {printf("E");}
"4" {printf("A");}
"9" {printf("G");}
"1" {printf("I");}
"5" {printf("S");}
"2" {printf("Z");}
"(" {printf("C");}
"_|" {printf("J");}
"|_" {printf("L");}
"|)" {printf("D");}
```

```
"|\/|" {printf("M");}

"E" {printf("3");}
"A" {printf("4");}
"G" {printf("9");}
"I" {printf("1");}
"S" {printf("5");}
"Z" {printf("2");}
"C" {printf("(");}
"J" {printf("_|");}
"L" {printf("|_");}
"D" {printf("|)");}


%%

int
main()
{
 yylex();

return 0;

}
```

Magyarázat:A leet egy főleg az interneten elterjedt rejtnyelv. Különböző számokkal és jelekkel helyettesítenek betűket, az alapján h mihez hasonlítanak.A fenti kód egy ilyen szöveget alakít olvashatóvá, vagy sima szöveget alakít át.Ennek a programnak is a lexer az alapja csak úgy mint az előző feladatnál. A második fő részben vannak megadva a betűk és számok. Például az első sor: ha talál a program egy 3-ast azt át fogja írni egy E betűre.

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);
```

Ha a SIGINT jelkezelése nem volt figyelmen kívűl hagyva akkor most se legyen. Ha figyelmen kívűl volt hagyva akkor a jelkezelő függvény kezelje.

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

Ez egy for ciklus.A ciklusváltozónk az 'i' melynek kezdőértéke nulla.A ciklus addig fog futni amíg az i kissebb lesz mint 5, és minden ciklus végén az 'i'-t növeljük eggyel.

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

Ez a for ciklus első ránézésre olyan mint az előző, azonban van egy különbés. Az előző ciklusnál ++i volt most pedig i++. Ennek a for ciklusnál nincs nagy jelentősége,mindkettő növelni fogja eggyel i értékét. Azonban vannak olyan helyzetek amikor nagy jelenősége lehet.Pl i=1 v=i++ és v=++i. Az elsőnél v értéke 1 lesz miközben i értéke kettő, mert hamarabb fogja v felvenni i értékét minthogy növelné eggyel i értékét. A másodiknál pedig előbb növeli, aztán teszi egyenlővé, szóval kettő lesz v értéke.

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

Egy tömböt tölt fel i értékeivel. Jelen esetben egy öt elemű tömböt hoz létre.

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

Ebben a for ciklusban az i kisebb mint n mellett van még egy feltétel ,és a ciklus csak akkor fog lefutni ha mindkét feltétel igaz lesz .A második feltétel nem lesz jó,mert nem logika értéket fog visszaadni(== kellene).

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

A printf segítségével kiíratunk két egész számot melyeket egy egy függvény fog meghatározni.

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

Kiíratunk kt egész típusú számot, az egyik egy függvény fogja meghatározni.

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

Ugyanaz mint az előző csak itt az f függvény egy memóriacímet fog kapni

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

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

Ennek a feladat megoldásához a következő logikai összekötőjeleket kell ismernünk:\neg=negáció(tagadás),\vee=v
A kvantorok \exist=létezik, és a minden =\forall. Az első: Bármelyik x számhoz, tartozik egy olyan y prím szám, ami x-nél kisebb. Második:

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;
```

- Egy int típusú, azaz egész számot vezet be,melynek neve a lesz

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

- A ’b’ tárolni fogja ’a’ memóriacímét, azaz ’b’ ’a’-ra mutató mutató lesz.

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

- Itt az r nem vátlozó hanem egy memóriacím, mégpedig ’a’ memóriacíme.

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

- Egy egész típusú 5 elemű tömb deklarálása. Az indexelés 0-tól kezdődik.

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

- c 5 elemű tömg, minden eleméhez referenciát rendel.

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

- d 5 elemű tömb összes tagja egy mutató lesz.

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

- int típusú pointert visszaadó függvény.

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

- Egész típusra mutató pointert visszaadó függévny

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

- Két egészt kapó

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Kód:

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

int main()
{
  int nr=5;
  double **tm;
  //printf("%p\n",&tm);

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

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

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

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

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

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

for(int i=0; i<nr; i++)
{
  for(int j=0; j<i+1; j++)
    printf("%f,", tm[i][j]);
  printf("\n");
}
tm[3][0]=42.0;
(*(tm+3))[1]=43.0;
*(tm[3]+2)=44.0;
*(*(tm+3)+3)=45.0;

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

for(int i=0; i<nr; i++)
  free(tm[i]); //felszabadítjuk a lefoglalt sorokat

free(tm); //mindent felszabadítunk

return 0;
}
```

A fent látható kód egy alsó háromszögmátrixot fog kiíratni.Az nr változóban adhatjuk meg,hogy hány soros legyen a mátrix, jelen esetünkben 5 lesz. Ezután a **tm el egy pointernek foglalunk le helyet. Majd a malloc függvény segítségével annyiszor foglalunk le double-nyi méretet ahány sorunk van. A malloc alapesetben egy pointert ad vissza, de mi típuskényszerítéssel double ** ot kapunk vissza. Ha a malloc valami hiba során nem tudja lefoglalni a megfelelő mennyiségű memóriaterületet akkor egy NULL pointert fog visszaadni,ezáltal az if igaz lesz és kilép a programunk, ha nem történik ilyen akkor tovább lép.Ezután indítunk egy for ciklust amely 0-tól, jelen esetben négyig fog futni.Ez a for ciklus fogja lefoglalni minden sornak a megfelelő méretet. Az első sornak 1-et a másodiknak 2-őt és így tovább.Itt megint csak a malloc függvényt használjuk.Ezután még két for ciklus segítségével feltöltjük a mátrixot. Majd a végén kiíratjuk a mátrixot.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Kód:

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

int main (int argc, char **argv)// argc - stringek száma, ami az argv-re ↩
   mutat, az **argv pedig tárolni fogja a parancssori argumentumokat
{

  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 fenti kód a kizáró vagy(EXOR) segítségével fogja titkosítani a megadott tiszta szövengüket.A program fordítása után, a futtatási parancs után megadjuk a kulcsunkat(ez alapján fogja titkosítani a szöveget) majd "" segítségével átadjuk a filet amit tikosítani szeretnénk és ">" segítségével kiíratjuk egy másik file ba a titkosított szöveget.A kód elején a #define segítségével MAX_KULCS és BUFFER_MÉRET néven megadunk két konstanst.A main elején létrehozzuk a szükséges változókat. A kulcs_meretet egyenlővé tesszük az strlen függvény segítségével az argv[1] méretével. Az argv tömb tárolja a parancssori argumentumokat, az első tagja igazából a 2. tagja lesz mert 0-tól indul az indexelés, tehát ez a kulcsunk mérete lesz.A strncpy függvény a kulcs változóba másolja át a megadott kulcsunkat.Ezután egy while ciklus következik, ez addig fog futni amíg a fájlunkból adatot tud olvasni,közben eltároljuk az olvasott bájtok számát. Ezekután indítunk egy for ciklust ami az olvasott bájtok számáig fog futni.Ebben a ciklusban zajlik maga titkosítás.A bufferben lévő bájtokat össze exorálja a kulcs bájtjaival és így egy másik karaktert kapunk.A végén pedig kiíratjuk a titkosított szövegünket.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Kód:

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

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

int tiszta_lehet(const char titkos[], int titkos_meret) //tiszta szöveg lehet, függvény nézi, kap egy állandó karakter tömböt, meg egy méretet
{
  //tiszta szöveg valószínű, hogy tartalmazza a gyakori magyar szavakat
  return strcasestr(titkos,"hogy") && strcasestr(titkos,"nem") && strcasestr(titkos,"az") && strcasestr(titkos,"ha"); //megkeresi hogy van e benne hogy, nem, az, ha
}

void exor(const char kulcs[], int kulcs_meret, char titkos[], int titkos_meret)
//exor eljárás, kap egy kulcs tömböt, egy kulcs méretet, a titkos tömböt és a titkos méretét
{
  int kulcs_index=0; //deklaráció

  for(int i=0; i<titkos_meret; ++i) //a ciklus a titkos méretig fut
  {
  titkos[i]=titkos[i]^kulcs[kulcs_index]; //exorral nézi a kulcsokkal a dolgokat, ya know this bad boi
  kulcs_index=(kulcs_index+1)%kulcs_meret;
```

```
  }
}


int exor_tores(const char kulcs[], int kulcs_meret, char titkos[], int  ↩
   titkos_meret)
//exortores kulcs tömb, kulcs méret, titkos tömb, méret
{
  exor(kulcs, kulcs_meret, titkos, titkos_meret); //exortörés go for it

  return tiszta_lehet(titkos, titkos_meret); //tiszta lehet megnézi

}

int main(void)
{
//deklarációk
  char kulcs[KULCS_MERET];
  char titkos[MAX_TITKOS];
  char *p=titkos;
  int olvasott_bajtok;

//titkos fajt pufferelt berantasa //HÜLP
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%c]\nTiszta szoveg: [%s]\n",ii,ji,ki,li ↩
         ,mi,ni,oi,pi,titkos);
    //ujra EXOR-ozunk, igy nem kell egy második buffer
    exor(kulcs,KULCS_MERET,titkos,p-titkos);
  }
return 0;
}
```

A kód elején define segítségével vannak deklarálva konstansok, ezekre a későbbiekben könnyebb lesz hivatkozni.Ezután a main függvény előtt vannak deklarálva különböző eljárások és függvények, amiket majd a programunk során használni fogunk.A tiszta_lehet függvény az első. Ez paraméterként kap egy karakter tömböt és egy méretet,majd ebben a tömbben a 'hogy', 'nem','az' és 'ha' szavakat keresi és ezt adja vissza is .Alatta az exor nevezetű eljárás található,azért eljárás és nem függvény mert nincs visszatérítési értéke.Ez az eljárás paraméterként kap egy kulcs tömböt,kulcs méretet illetve a titkos tömböt és titkos méretet.Ezután egy for ciklus segítségével a titkos szöveget bájtonként össze exorálja a kulccsal.Ez alatt található az exor_tores függvény amyel meghívja az exor eljárást. Ezután a tiszta_lehet függvény és ennek fogja visszatéríteni az értékét.Ezután következik a main függvény.A deklarálások után egy while ciklussal beolvassuk a bufferbe a titkos szöveget, ezután egy for ciklussal a bufferben lévő maradék helyet nullákkal töltjük fel. Ezután nyolc egybeágyazottfor ciklussal melyek mindegyike 0-tól 9-ig fut, előállítjuk az összes lehetséges kulcsot. Ez a program maximum 8 számjegyű kulcsot tud feltörni. Ezután meghívjuk az exor_tores fuggvényt és ha igazat ad vissza kiíratjuk a kulcsot és a tiszta szöveget.

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...

4.6. Hiba-visszaterjesztéses perceptron

C++

Kód:

```
        #include <iostream>
#include "mlp.hpp"
#include "png++/png.hpp"
//g++ mlp.hpp main.cpp -o perc -lpng -std=c++11

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;
}
```

Megoldás forrása:

A program futásához szükségünk lesz a libpng csomagra.Ezzel a programmal a mandel.png RGB kódjának a red részét adjuk át,és a program egy három rétegűt hálót alkot belőle.A futáshoz szükségünk van az mlp.hpp header fájlra is,ugyanis ez tartalmazza a perceptron classt.A main-ben először lefoglaljunk a szükséges helyeket,majd egy for ciklussal feltöltjük a képet és átadjuk számolásra.Majd a végén kiíratjuk.

5. fejezet

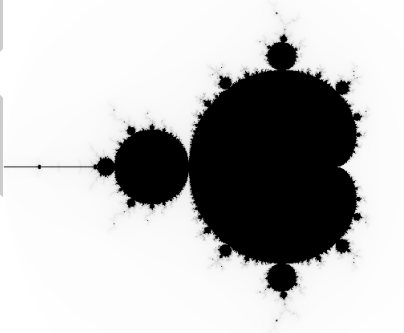
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

```
            // Copyright - no copyright

#include <png++/png.hpp>

#define N 500
#define M 500
#define MAXX 0.7
#define MINX -2.0
#define MAXY 1.35
#define MINY -1.35

void GeneratePNG(int tomb[N][M]) {
  png::image<png::rgb_pixel> image(N, M);

  for (int x = 0; x < N; ++x) {
    for (int y = 0; y < M; ++y) {
      image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y]);
    }
  }
  image.write("kimenet.png");
}

struct Komplex {
  double re, im;
};


int main() {
  int tomb[N][M];

  int i, j, k;
  double dx = (MAXX - MINX) / N;
  double dy = (MAXY - MINY) / M;
```

```
  struct Komplex C, Z, Zuj;

  int iteracio;

  for (i = 0; i < M; ++i) {
    for (j = 0; j < N; ++j) {
      C.re = MINX + j * dx;
      C.im = MAXY - i * dy;

      Z.re = 0;
      Z.im = 0;
      iteracio = 0;

      while (Z.re * Z.re + Z.im * Z.im < 4 && iteracio++ < 255) {
        Zuj.re = Z.re * Z.re - Z.im * Z.im + C.re;
        Zuj.im = 2 * Z.re * Z.im + C.im;
        Z.re = Zuj.re;
        Z.im = Zuj.im;
      }
      tomb[i][j] = 256 - iteracio;
    }
  }
  GeneratePNG(tomb);

  return 0;
}
```

A fenti kód a következő png képet fogja generálni:



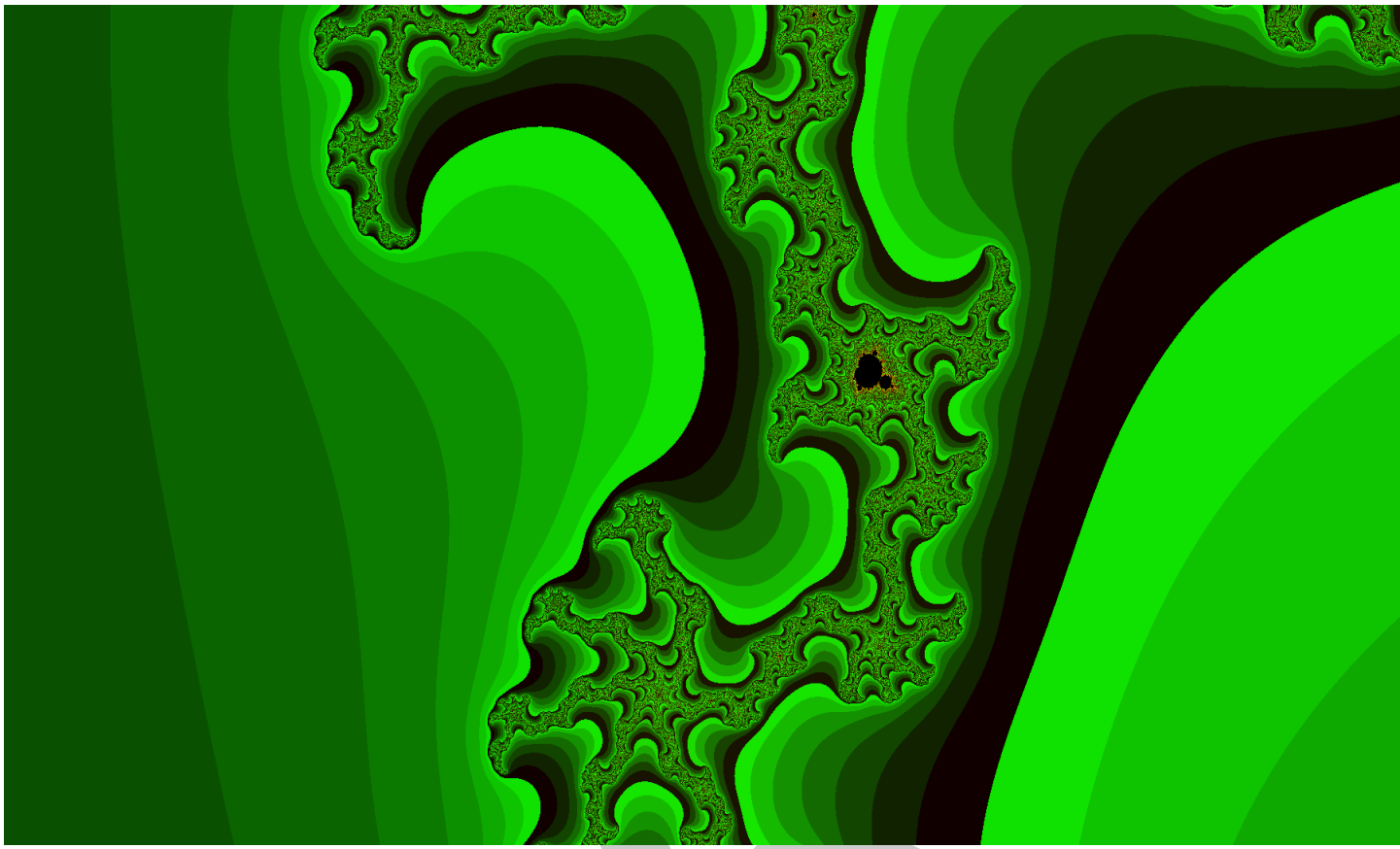
Ez a mandelbrot halmaz síkbeli ábrázolása.A mandelbrot halmaz komplex számokból áll.Ezekre a számokra igaz az,hogy X(n+1)=Xn^2+c. És ez nem tart a végtelenbe, azaz korlátos. A forráskód fordításához szükség van egy -lpng kapcsolóra,ugyanis a kód tartalmazza a png++ header file-t, ez szükség a png generáláshoz. A kódunk elején deklarálunk pár konstans számot, majd a png generáló eljárást is.Ez paraméterül kap egy NxM-es két dimenziós tömböt.Egy 500x500 -as képet fog létrehozni, két for ciklus segítségével és a tömbben lévő értékek szerint vagy fehér lesz az adott pixel vagy fekete. Az eljárás végén write-al hozzuk létre a png-t.Ezután létrehozzuk a Komplex struktúrát amiben egy valós és egy komplex szám lesz.A main elején deklaráljuk a szükséges változókat, és létrehozunk 3 Komplex osztályú változót is.Ezután a megfelelő képletet használva feltöltjük a mátrixot a megfelelő értékekkel és átadjuk a GeneratePNG eljárásnak a

mátrixot.

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

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

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

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

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

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


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

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

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

```
    {


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


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

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

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

              ++iteracio;
            }

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

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

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

}
```



Ez a kód nem sokban különbözik az előzőtől. Bekerült a complex header file. Ezáltal nem kell struktúrát használnunk,hogy kezeljük a komplex számokat, hanem ez a header file fogja kivitelezni azt. A kód elején deklaráljuk a szükséges változókat. Futtatáskor meg kell adnunk a kép magasságát és szélességét, azt hogy milyen néven hozza létre a képet, és az n a b c d értékeit. Ezt követően a program beállítja a megfelelő értékekre a változókat. Ha viszont rosszul adtuk meg a dolgokat(pl kevés számot adtunk meg) akkor kiíratja a használatot.Ezután a szükséges méretet lefoglaljuk a majd létrejövő képünknek.Maga a számolás a while ciklusban történik. A program futása közben folyamatosan tájékoztat arról,hogy hány százaléknál jár a generálás és arról is ha sikeresen lementette a képet.

5.3. Biomorfok

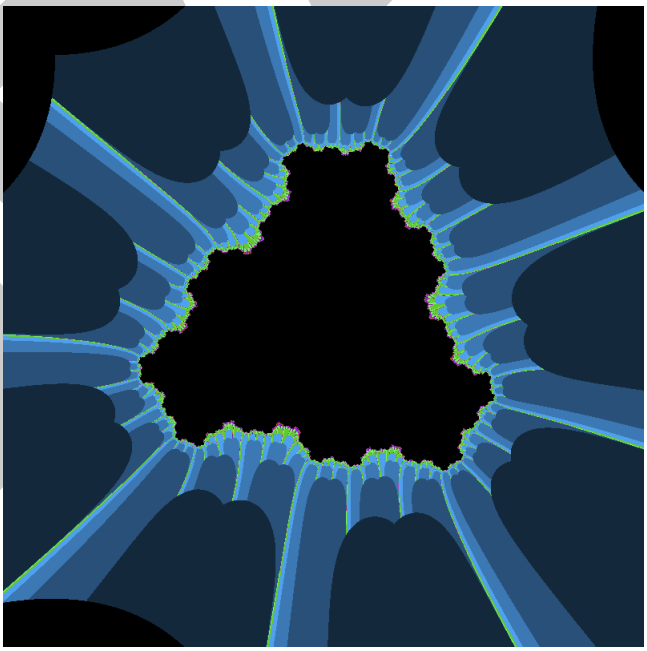
Megoldás videó: https://youtu.be/IJMbgRzY76E

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

```
#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;

}
```



Ez a program lényegében Julia halmazokat fog létrehozni. Julia halmazokból végtelen mennyiségű van. Ennek a programnak az alapja az előző mandelbrotos program.A mandelbrot halmaz tartalmazza az összes Julia halmazt. A mandelbrotos programmal ellentétben itt a c nem változó lesz,hanem állandó, és a konzolról kérjük be. A kód elején megint csak létrehozzuk a megfelelő változókat. Majd bekérjük azokat és ha megfelelőek akkor a megfelelő változóhoz rendeljük azokat.Lefoglaljuk a helyet a képünknek. For ciklu-

sukkal végig megyünk az összes pixelen és a megfelelő egyenlet segítségével kiszámoljuk az értékeket.Ez a program is jelzi,hogy hány százaléknál jár és hogy lementette-e már a képet.

5.4. A Mandelbrot halmaz CUDA megvalósítása

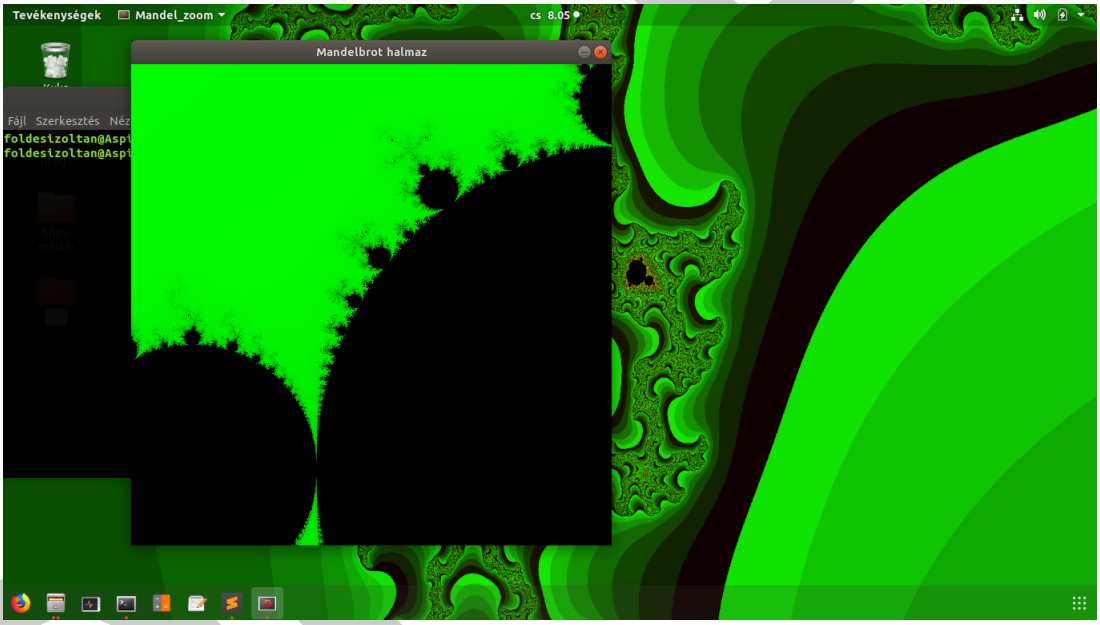
Megoldás videó:

Megoldás forrása:

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

```
              #include <QApplication>
#include "frakablak.h"

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

    FrakAblak w1;
    w1.show();

    /*
    FrakAblak w1,
    w2(-.08292191725019529, -.082921917244591272,
       -.9662079988595939, -.9662079988551173, 600, 3000),
    w3(-.082921917248806 25, -.0829219172470933,
       -.9662079988581493, -.9662079988563615, 600, 4000),
    w4(.14388310361318304, .14388310362702217,
       .6523089200729396, .6523089200854384, 600, 38655);
    w1.show();
    w2.show();
    w3.show();
    w4.show();
*/
    return a.exec();
}
```



A fent látható kód működéséhez szükséges telepíteni a libqt4-dev-et.Valamint szükséges összesen 5 fájl. A fájlokat az előbb említett csomaggal fogjuk összefűzni. Először qmake -project parancssal létrefogunk hozni egy .pro kiterjesztésű fájlt, ennek a fájlnak az utolsó sorába be kell írnunk a QT += widgets mondatot. Ezek után qmake *.pro paranccsal létrejön egy makefile.Majd a make parancssal létrehozzuk a futtatható fájlt. Ezután futtatva a kódot megjelenik a Mandelbrot halmaz, amiben kijelölve egy területet bele tudunk nagyítani szinte a végtelenségig. Ez látható az első és a második képen.

Megoldás forrása:

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:

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

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

6.3. Fabejárás

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

Megoldás videó:

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó:

Megoldás forrása:

6.5. Mutató a gyökér

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

Megoldás videó:

Megoldás forrása:

6.6. Mozgató szemantika

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

Megoldás videó:

Megoldás forrása:

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

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

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

7.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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:

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.

10. fejezet

Helló, Arroway!

10.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. Java osztályok a Pi-ben

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

10.3. Általános

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

10.4. C

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

10.5. C++

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

10.6. Lisp

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