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	Tóth, Balázs	2019. április 6.	

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

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?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó: https://github.com/ghjbku/DE/blob/master/video.flv

Megoldás forrása:

Elsőként a 100%-os végtelen ciklust készítettem el, hiszen ezt volt a legegyszerűbb megírni. Amint láthatjuk elég egyszerűen meg lehet oldani, hogy a cpu 100%-ban dolgozzon a program futása alatt. Itt én a WHILE ciklus-t választottam, de FOR-ral is hasonlóképpen lehet megvalósítani a végtelenítést. Az egész program lényege egyetlen értéken alapszik, amit az *asd* változó hordoz. Mivel ez a változó semmiképp sem kap 1-et értékül, a program soha sem fog kilépni a ciklusból.

```
//100%-ban megdolgoztat egy magot
//lefordítás: gcc forrásnév -o késznév
#include <stdio.h>
int main()
{
int asd =0;
while (asd=1){}
return 0;
}
```

A következő program a 0%-os végtelen ciklus volt. Ha ismerjük az API-t, vagy tapraesettek vagyunk a google-n való keresést illetően, akkor itt is egyszerű dolgunk volt. Amint azt észrevehettük, a programkód nagyon hasonlít az előző kódra, csupán annyi változás történt, hogy a ciklus belsejében megjelent egy függvény, a *sleep()*. Ez a függvény annyi milisecond-ig állítja meg a programot, amely számot a két zárójel közé írtunk. Jelen esetben ez *1*, de mivel egy végtelen ciklusban vagyunk, ezért végtelen sokszor vár majd 1 milisec-ot a program, így tehát nem használ erőforrást.

```
//0%-os cpu használat
//lefordítás: gcc forrásnév -o késznév
#include <stdio.h>
```

```
int main()
{
int asd =0;
while (asd=1)
{
sleep(1);
}
return 0;
}
```

Utolsóként pedig jön a "legnehezebb", minden magot 100%-on futtatni. Az igazat megvallva, ez sem valami nagy ördöngősség, itt is csak egy pár dolog változott a legelső programhoz képest. A legfontosabb dolog ez a sor *#include "omp.h"*, ez a header fájl előfeltétele annak, hogy a *#pragma omp parallel* kódot értelmezni tudja a fordítóprogram. A *#pragma...* sor veszi rá a programunkat, hogy párhuzamos módon, az összes magon futtassa a programot a számítógép.

```
//minden mag 100%-on fut
//lefordítás: gcc -fopenmp forrásnév -o késznév
#include <stdio.h>
#include <unistd.h>
#include "omp.h"
int main () {
int asd=0;

#pragma omp parallel
  while(asd=1)
  {
  }
   return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Amint láthattuk, elég egyszerű dolgunk volt ezen programok megírását illetően, viszont ez nem azt jelenti, hogy félvállról vehetjük a programozást, hiszen kevés olyan program létezik, aminek valamilyen hétköznapi haszna van, és mégis ilyen egyszerű lenne megírni. Ezen programkódok csak az egyszerűbb megértést segítik elő, gyakorlati hasznuk sajnos nincs.

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...

Amint láthatjuk, lehetetlen olyan programot írni, amely egy másik programról eldöntené, hogy az le fog-e fagyni, vagy sem.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

Ezen feladat megoldása igencsak egyszerűnek bizonyult.Na persze nem annyira egyszerű, mint egy végtelen ciklus megírása, de közel azonos szinten mozognak. A lentebb lévő forráskód elég egyszerűen értelmezhető, ezért hát nem megyek bele részletesen, csak a nagyon fontos dolgokat mondom el. A *C* nyelvben a változók értékét egy paranccsal tudjuk hozzáfűzni egy printf függvényhez, attól függően, hogy milyen típusú adatot hordoz a változó. Esetünkben mindkét változó *szám/Digit* típust hordoz, ezért a kód, amivel meghívjuk a behelyettesítő paramétert, ez lesz: * %d *, majd ha végeztünk a kiírni kívánt szöveggel, egy vesszővel jelezzük a fordítóprogramnak, hogy most a behelyettesítendő változók következnek. A kódban megjelenik egy másik kód is, ami ismeretlen lehet az olvasó számára, ez a * \n * , amely annyit tesz, hogy új sorba kezdi az *n* után beírt szöveget, és a szóközt is értelmezi!

```
#include <stdio.h>
int main()
{
int a=5,b=3;
printf("A value = %d\n",a);
printf("B value = %d\n",b);
b=b-a;
a=a+b;
```

```
b=a-b;
printf("A value = %d\n",a);
printf("B value = %d\n",b);
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Ne tévesszen meg bennünket a feladat komplexitása, ha nem gondolunk bele, hogy pontosan hogyan is kellene segédváltozó nélkül elérni céljainkat, elég sokáig el tudunk időzni ezen az egyszerű feladaton. Tehát próbáljunk meg minden feladatot úgy kezdeni, hogy elgondolkozunk azon, hogyan tudnánk megvalósítani a feladatban megírtakat.

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:

A feladat nehézségi szintjét tekintve már egy magassabb szinten van, vagyis inkább, gondolkodást igényel. Ebben a forráskódban már megjelenik egy pár új parancs, melyeket eddig még nem láttunk. Kezdve az új Header fájllal, a <math.h> fájllal, amely a matematikai függvényekért felel és minden értéket double típussal kezel(double típust kér, és azt ad vissza), ilyen függvény például az *abs*, amely az abszolút értéket jelöli, de ebben a header fájlban található a *pow* és az *sqrt* is, az előbbi a hatványozást, míg utóbbi a négyzetgyököt kezeli. Aztán ott van az a furcsa sor két sorral alább, az a bizonyos *#define*... ezeket a sorokat úgynevezett "Nevesített konstansok" definiálásánál használjuk. Ezek a konstansok értéket nem változtatnak a program futása során, és bármilyen értéket adhatunk nekik.

```
//Labdapattogás if nélkül (mentorálva Gila Attila által)
#include<stdio.h>
#include<math.h>

#define szel 80
#define mag 24

int putX(x,y)
{
int ix,iy;

for(ix=0;ix<x;ix++)
printf("\n");

for(iy=0;iy<y;iy++)
printf(" ");

printf("O\n");
```

```
return 0;
}

int main()
{
long int x=0,y=0;

while(1)
{
system("clear");
putX(abs(mag-(x++%(mag*2))),abs(szel-(y++%(szel*2))));
usleep(15000);
}

return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

A labda "pattogása" egyszerű módon van megoldva, miszerint minden egyes "tick" után, amit a program a végtelen ciklusban tölt, a *system("clear")* parancs miatt a terminál jelenlegi tartalma törlődik, de mivel az túl gyorsan történik, mi csak úgy érezzük, hogy a labda szépen mozog az ablakban. a "tick" periódust az *usleep()* függvény zárójelében megadott szám határozza meg, a mértékegység microsecond. Viszont ha fontos a pontosság, akkor számolnunk kell a számítógép kalkulációs képességeivel, plusz az is időbe telik, hogy a program eljut az *usleep* függvényhez, ezután az egész program "alvó" állapotba kerül, kilép a processor ütemezési sorából, és a delay attól is függhet, hogy a processor maga mikor válassza újra a programot, miután a *usleep* függvény lefutott. Tehát ne lepődjünk meg, ha néhány ezer microsec-ot téved a program.

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: https://github.com/ghjbku/DE/blob/cpp/bitshift.cpp

A feladat megoldása C++ nyelven történt, viszont C-ben is hasonló módon kell megoldani a problémát.

```
//a bitshift C nyelvben
#include <stdio.h>

int main(){

  unsigned int the_Bit = 1;
  int length = 0;

  do
```

```
    length++;
  while((the_Bit <<= 1));

  printf("A szóhossz mérete: %u\n", length);

  return 0;
}
```

A C megoldás BogoMIPS-el: itt található

Tanulságok, tapasztalatok, magyarázat...

A bit méretét a **length** változó tárolja, amit úgy töltünk fel, hogy amíg a bit el nem éri a kezdési értéket, addig a ciklusban mingíg növeljük a változó értékét 1-el. Majd ezen értéket a végén kiírjuk. Az unsigned típus 2^n különböző értéket vehet fel 0 és n között.

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 C++-ban: https://github.com/ghjbku/DE/blob/cpp/bearazas.cpp és C-ben: https://github.com/ghjbku/DE/blob/master/c%20cuccok/bearaz.c

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

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

double tavolsag(double pagerank[],double pagerank_temp[],int db)
{
double tav = 0.0;
int i;
for(i=0;i<db;i++)
tav +=abs(pagerank[i] - pagerank_temp[i]);
return tav;
}

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};

long int i,j,h;
i=0; j=0; h=5;

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

if ( tavolsag(PR,PRv, 4) < 0.00001)
break;
}
kiir (PR,4);
return 0;

}
```

Tanulságok, tapasztalatok, magyarázat...

A pagerank algoritmust a google fejlesztette ki azzal a céllal, hogy a weboldalak minőségét rangsorolja.

A feladat komplexitása miatt a soronkénti értelemzést választottam.Kezdjük is el.

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

Ez a függvény a minsősítés végeredményét fogja kiírni. Egy egyszerű for ciklusból áll, amely a függvényparaméterként megadott **db**-szor fog lefutni és kiírja az ugyancsak függvényparaméterből származó **tomb[]** tömb elemeit.

```
double tavolsag(double pagerank[],double pagerank_temp[],int db)
{
double tav = 0.0;
int i;
```

```
for(i=0;i<db;i++)
tav +=abs(pagerank[i] - pagerank_temp[i]);
return tav;
}
```

Ez a következő függvény már bonyolultabb.A függvényünk két double típusú tömböt és egy számot kér paraméterül. A távolság kiszámítására itt is egy for-ciklus lesz segítségünkre, azon belűl pedig egy abszolútérték függvény, amelyben a **pagerank** tömbből kivonjuk a **pagerank_temp** tömböt.

```
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};

long int i,j,h;
i=0; j=0; h=5;
```

Amint láthatjuk, a main függvényen belül elkezdjük definiálni a változókat, melyeket az előbbi két függvényre majd ráeresztünk.

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

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

Továbbra is a main függvényben vagyunk. Már megtörtént a változó deklarálás, tehát elkezdődhet a rangsorolás. Belépünk egy for-ciklusba, majd azon belűl rétegezve létrehozunk még 3 másik for-ciklust. Az első réteg a **PR** tömböt azonosítja a **PRv** tömbbel. A második réteg egy **temp** változóban összeszorozza az **L[][]** kétdimenziós tömböt és az újonan kapott **PR** tömböt. Majd minden ciklus végén hozzáadja az új értékeket az előző értékhez. Ezután a **PRv** tömb értékéül adjuk a temp változót. ezután visszatérünk az

első for-ciklusba, ahol pedig egy if elágazással megnézzük, hogy a **tavolsag()** függvény visszatérési értéke kissebb-e, mint 0.00001 . Ha igen, akkor kilép a ciklusból. A program a végén kiírja a **PR** tömb tartalmát a **kiir()** függvény segítségével.

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

```
library(matlab)

stp <- function(x){

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

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

A Brun tétel az ikerprímekkel foglalkozik, és kimondja, hogy ezen prímek reciprokösszege egy véges értékhez konvergál, ún. Brun-konstans felé. jelölése: $\mathbf{B_2}$

Mivel az R nyelvben még gyakorlatlanok vagyunk, így megint soronként fogok magyarázatot adni a forráskódra.

```
library(matlab)

stp <- function(x){

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

A library(matlab) paranncsal meghívjuk a matlab külső fájlt, amelyben egyéb függvények mellett megtalálható a **primes()** függvény. Ez a függvény a paraméterként megadott számig kiszámolja a prímszámokat. R-ben a függvény létrehozása így történik:

függvénynév <- function(paraméter lista){függvény törzs return(visszatérési érték)}

az első sorban feltöltjük a primes változót a **primes(x)** függvény értékeivel. majd a második sorban a diff változóba belerakjuk nd> és a **primes[1:length(primes)-1]** vektorok különbségét.

a **primes[2:length(primes)]** vektor a primes 2. elemétől a változó hosszáig tartalmazza a számokat. Ezzel szemben a **primes[1:length(primes)-1]** rész a primes 1. elemétől az utolsó előtti elemig tartalmazza a számokat. Ezeket kivonva megkapjuk a prímszámok különbségét. Ha ez a külnbség 2, akkor beszélünk ikerprímekről. Azt, hogy a különbség 2-e, az idx változó nézi meg, majd az indexüket eltárolja.

a t1primes változó tartalmazza azokat a prímeket, amelyeknek a helyét már meghatároztuk az idx változóban. Tehát az ikerprímek első fele. A t2primes viszont nem szimplán a primes[idx]-et adja vissza, hiszen az az ikrek első fele lenne, de tudjuk, hogy a különbség a kettő prím között 2, tehát a másik felét úgy kapjuk meg, ha hozzáadunk az [idx] helyen álló számhoz 2-őt.

Az rt1plust2 változóban összeadjuk a t1 és a t2 reciprokait. majd végezetül visszaadjuk a return()-ben a reciprokértékek összegét.

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

```
        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)
```

Tanulságok, tapasztalatok, magyarázat...

a kísérlet változóban található a nyeremény helye, a játékos változóban található a játékos által választott ajtó. Az első for ciklusban megnézzük, hogy a játékos eltalálta-e a helyes ajtó számát, és a műsorvezető ezen feltételtől függően választ ajtót magának. Ha eltalálta, akkor a műsorvezető véŁetlenszerűen választ a két üres ajtó közül. Viszont ha nem találta el a játékos, akkor a vezető csak 1 ajtót választhat, hiszen nem nyithatja ki a nyereményt, se a játékos által választott ajtót.

3. fejezet

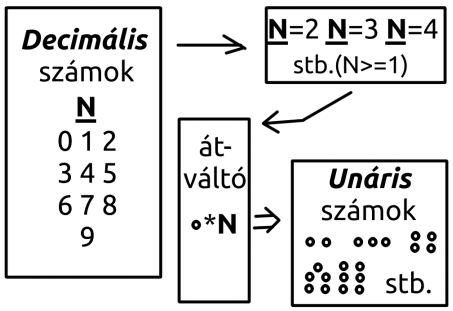
Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:



3.1. ábra. Az átváltó Turing gép

Tanulságok, tapasztalatok, magyarázat...

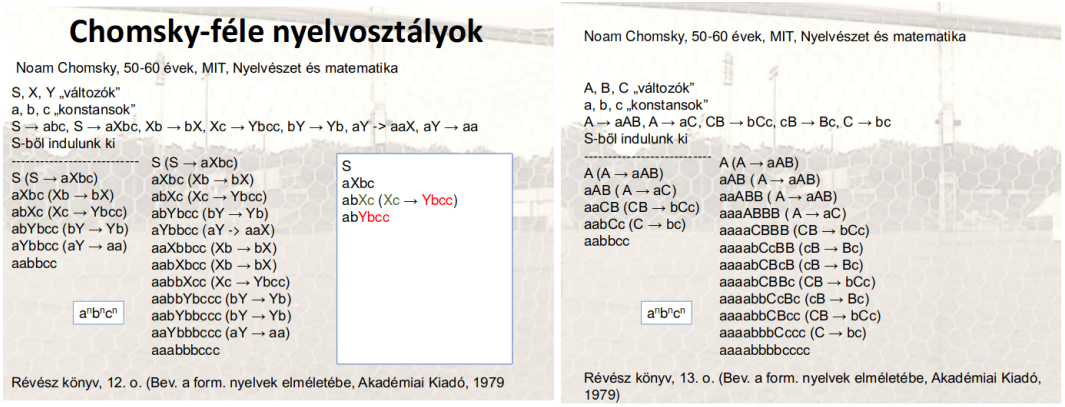
Az átváltó bekér egy Decimális számot, legyen ez egy tetszőleges szám, és nevezzük el N-nek. Ezután egy képlettel átváltja azt Unáris számrendszerbe. Ez a képlet a következő: **unar=egysegelem*N** majd "kirajzolja" az eredményt.

3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen

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

Megoldás videó:

Megoldás forrása: A lentebb látható képek itt megtalálhatóak a 30. oldalon



3.2. ábra. A környezetfüggő grammatikák

Tanulságok, tapasztalatok, magyarázat...

A konstansok és változók alatt helyezkednek el a helyettesítési szabályok. Ezeket a szabályokat alkalmazva addig változtatjuk a megadott szót, amíg a szó maga már csak konstansokból áll. Mivel nincs olyan lehetőség, hogy a szó csak változókból áll, ezért a nyelvezet nem lehet környezetfüggetlen.

Környezetfüggő(hossznemcsökkentő)

P1XP2 → P1QP2, P1, P2 eleme (VN U VT)*, X VN beli, Q (VN U VT)+ beli, kivéve S → üres, de akkor S nem lehet jobb oldali egyetlen szabályban sem, tehát Az $a^nb^nc^n$ nyelv nem környezetfüggetlen

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

```
#include <stdio.h>

int main()
{
  int inline asdfunc(int a)
  {
  int b = a*a;
  a=b*a;
  return a;
  }

asdfunc(5);
return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

https://github.com/ghjbku/DE/blob/master/c%20cuccok/Bildschirmfoto%20von%202019-03-12%2011-35-07.png Amint láthatjuk, a funkció minden gond nélkül lefordul C99-ben, míg C89-ben hibát észlel. A hiba az *inline* parancs miatt van. ez a cmd a C99-el jött be, amely általában *extern inline* -al párban jelenik meg egy programkódban. Segítségével egy program nagyobb sebességre képes, viszont ezért cserébe nagyobb a helyigénye.

3.4. Saját lexikális elemző

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU (15:01-től).

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

```
        //lex fájl
        //fordítás c-re : lex -o output.c lexfajl.l (szükséges hozzá a flex ↩
           )
        //jelen esetben: lex -o realnumbers.c realnumbers.l
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
%%
{digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
 yylex ();
 printf("The number of real numbers is %d\n", realnumbers);
 return 0;
}
```

a C-re fordított program ha készen van a fordítás, akkor mondhatjuk a gcc-nek, hogy csináljon nekünk futtatható programot a c forrásból. Ezt a következő sor beírásával tehetjük meg: * gcc realnumber.c -o realnumber -lfl *

Tanulságok, tapasztalatok, magyarázat...

Ebben a feladatban kicsit eltértünk a megszokott dolgoktól, ugyanis nem mi írtuk meg a C forráskódot, hanem a lexer. Ez nagyban megkönnyíti a dolgunkat, hiszen ha megnézzük a C forrást, amit a lex elkészített helyettünk, láthatjuk, hogy nem éppen egy rövid kis kódsorozatról van szó A lexer segítségével nekünk már csak annyi a dolgunk,hogy megmondjuk neki, milyen típust keressen az inputban"digit[0-9]", és hogyan ismerje azt fel " {digit}*(\.{digit}+)? "

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

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

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

```
%{
//a fordítás megegyezik az előző feladatéval: lex -o output.c lexfájl.l
//majd gcc output.c -o output
  #include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <ctype.h>

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

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

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

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

// https://simple.wikipedia.org/wiki/Leet
  };

%}
%%
. {

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

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

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

          if(r<91)
          printf("%s", l337d1c7[i].leet[0]);
          else if(r<95)
          printf("%s", l337d1c7[i].leet[1]);
        else if(r<98)
          printf("%s", l337d1c7[i].leet[2]);
        else
          printf("%s", l337d1c7[i].leet[3]);

        found = 1;
        break;
      }

    }

    if(!found)
       printf("%c", *yytext);

  }
%%
int
main()
{
```

```
  srand(time(NULL)+getpid());
  yylex();
  return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

Ez a program a terminálról beolvassa a karaktereket, és randomizálva 4 különböző karaktert rak az eredeti helyére. Ha számára ismeretlen karaktert írunk be, akkor visszaadja ugyan azt. A program magja egy l337d1c7 tömb, amely tárolja a helyettesítési értékeket minden karakterhez. Ha nem talál egyetlen karaktert sem az inputban, akkor nem ír ki semmit.

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** jelzés nem volt ignorálva, akkor ignorálja.

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

egy for ciklus, amely 0-tól 5-ig tart, tehát 5x fut le, és minden lefutás után inkrementálja az i értékét 1-el, majd az inkrementálási értéket adja vissza.

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

ez is egy for ciklus, viszont ebben az esetben az i inkrementálása után az eredeti, növelés előtti értéket adja vissza.

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

Ebben a for ciklusban nem csak, hogy növeljük az i értékét minden kör után, de ezen értéket behelyezzük egy tömbbe is.

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

a ciklus 0tól indul, és addig megy, amíg i kissebb mint n, továbbá a **d** pointer növelt értéke megegyezik az **s** pointer növelt értékével.

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

kiír két függvényt, melyek számokat adnak vissza, viszont a visszatérési érték precedenciát sugall.

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

visszaad kettő számot, melyekből az egyik egy függvény visszatérési értéke

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

ugyanúgy két számot ad vissza, de az f függvényben az **a** változó memóriacíme helyezkedik el.

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...

az első formula természetes nyelvi értelmezése: **minden x számra létezik egy olyan y szám, amely nagyobb, mint x és y prím**

a második formula természetes nyelvi értelmezése: **minden x számra létezik egy olyan y szám, amely nagyobb, mint x és y prím.Továbbá y rákövetkezőjének a rákövetkezője is prím.**

a harmadik formula természetes nyelvi értelmezése: **minden y számra létezik egy olyan x szám, hogy x prím és x kisebb, mint y**

a negyedik formula természetes nyelvi értelmezése: **minden y számra létezik egy olyan x szám, hogy y kisebb, mint x, és x nem prím.**

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
  int a;
  ```
  egy szám típusú **a** változót
- ```
  int *b = &a;
  ```
  egy szám typusú **b pointert**, aminek az értéke **a memóriában foglalt helye**
- ```
  int &r = a;
  ```
  integer típusú **r értéke a** lesz
- ```
  int c[5];
  ```
  létrehoz egy 5 számnak helyet adó **c** tömböt
- ```
  int (&tr)[5] = c;
  ```
  létrehoz egy **tr tömb referenciát**, melynek az értéke c
- ```
  int *d[5];
  ```
  egy egészekre mutató **d tömb pointer**

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

  h funkcióra mutató pointer

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

  egy pointer, ami az **l függvényre mutató pointerre mutat**

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

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

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

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

Megoldás videó:

Megoldás forrása:

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

```
#include <stdio.h>

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

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

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

}

int
```

```
main ()
{

    int (*f) (int, int);

    f = sum;

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

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

    g = sumormul;

    f = *g (42);

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

    return 0;
}
```

```
#include <stdio.h>

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

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

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

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

int
main ()
{

    F f = sum;
```

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

    G g = sumormul;

    f = *g (42);

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

    return 0;
}
```

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

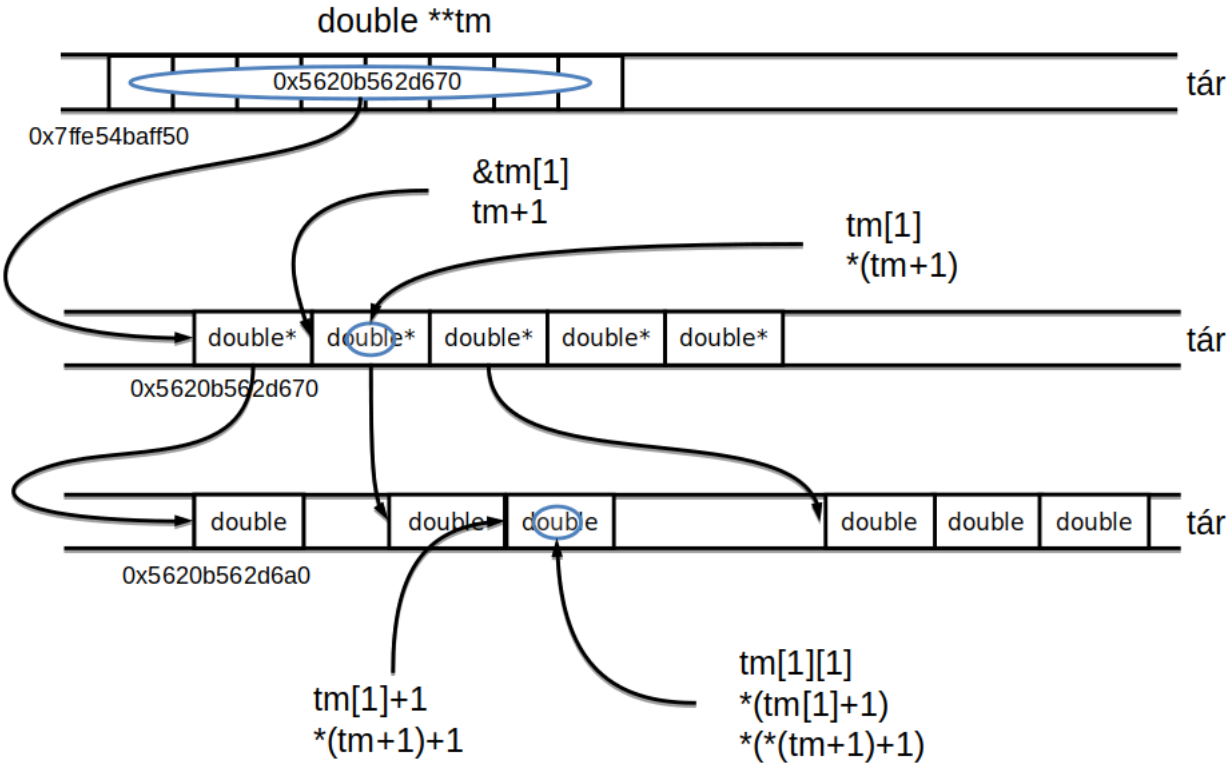
Írj egy olyan `malloc` és `free` párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

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

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

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

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

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

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

    }

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

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

    for (int i = 0; i < nr; ++i)
    {
        for (int j = 0; j < i + 1; ++j)
            printf ("%f, ", tm[i][j]);
        printf ("\n");
    }
    //ezek a sorok átírják az előbb feltöltött tm változó értékeit a ↩
       következőkre:
    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
    *(tm[3] + 2) = 44.0;
    *(*(tm + 3) + 3) = 45.0;

    //majd újra kiírja a program az egész tömböt, immár a megváltoztatott ↩
       értékekkel
    for (int i = 0; i < nr; ++i)
    {
        for (int j = 0; j < i + 1; ++j)
            printf ("%f, ", tm[i][j]);
        printf ("\n");
    }

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

    free (tm);

    return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat...

Az

```
int nr = 5;
```

sorral létrehozunk egy integer változót, amelynek az értéke 5, ez a változó lesz a háromszög sorainak száma. A

```
double **tm;
```

sor pedig létrehoz egy double típusú változót, ez lesz később a programunk magja.

Ezek a sorok:

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

pedig megnézik, hogy a programunk le tud e foglalni **nr*8** bájtot, ha nem, akkor kilép a programból a **-1** visszatérési értékkel.

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

```
        {
            return -1;
        }
    }
```

Itt azt láthatjuk, hogy a program a tm változót tömbként kezelve bejárja azt, és mindíg **(i+1)*8**bájtot allokál/foglal le az aktuális tömb pozíciójához. Ha ez nem sikerül, akkor megint csak kilép **-1**es visszatérési értékkel.

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

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

Ezek a ciklusok a kiíratásért felelnek. **az első egybeágyazott for ciklus pár** a tm változót tölti fe1 számokkal 0-tól 15-ig, majd **a második cikluspár** kiíratja azokat

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

    free (tm);
```

Ezek a sorok a program végén felszabadítják a lefoglalt memóriahelyeket, először a változó tömbelemeitől kezdve, majd végül a most már üres változót is letörli.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása :

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

```
char kulcs[MAX_KULCS];
char buffer[BUFFER_MERET];
int kulcs_index=0;
int olvasott_bajtok=0;
int kulcs_meret=strlen (argv[1]);
strncpy (kulcs,argv[1], MAX_KULCS);

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

Tanulságok, tapasztalatok, magyarázat...

Amint láthatjuk, a main() függvényben megjelent két ismeretlen paraméter. ezek a program futtatásánál játszanak szerepet: **az argc** jelöli az argumentumok számát, beleértve a **./programnév** sort is. ezzel szemben a ****argv** egy vektor, amely az argumentumokat tárolja. Itt például, ha a terminálba ezt a sort írjuk: **./fájlnév 1234 -o output.txt**, akkor az argc értéke 4 lesz, míg a **argv vektor így néz ki: <./fájlnév; 1234; -o; output.txt>

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

Ez a két sor is ismeretlen lehet számunkra, de nem kell tőlük megijedni, elég egyszerű a kezelésük. az első sor az argumentum_vektor 1. elemét(ami az előző példában az 1234 volt) lekéri, és megszámolja annak hosszát(**ez az strlen() függvény dolga**), majd a kulcs_meret nevű változónak ezt a hosszt értékül adja. a második sor pedig egy string másoló függvény, ennek szintaktikája a következő: **strncpy(char cél_változó,char másolni_kívánt_érték,a_másolni_kívánt_érték_hossza)** (ha a másolt érték kisebb, mint az utolsó paraméterben megadott szám, akkor a maradékot **NULL** bájtokkal fogja kipótolni a program)

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

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

```
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();

        }

    }

}
```

Tanulságok, tapasztalatok, magyarázat...

A Java verzió is hasonlóképpen működik, mint a C verzió, csak a nyelvi sajátosságoknak köszönhetően találhatóak különbségek a két kód között. Mivel a Java is magasszintű programozási nyelv, ezért a forráskód értelmezése könnyebb, mint egy assembly nyelvé.

A main függvényben található egy try-catch blokk, ez egyfajta hibakeresés. A try részbe kerülnek a kódok, amik nagy eséllyel hibát dobhatnak, és a catch részben ezeket a hibákat elkapja a program, és a programozó által megadott üzenetet dobja ki. Itt például Ha valami nem stimmel a megadott argumentumokkal, a program kiad egy exception-t. Ez a C programban nincs jelen, de ha szeretnénk, akár oda is beleírthatjuk.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

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

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

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

    int kulcs_index = 0;

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

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

    }

}

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

    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

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

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

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

    // osszes kulcs eloallitasa
    for (int ii = '0'; ii <= '9'; ++ii)
        for (int ji = '0'; ji <= '9'; ++ji)
            for (int ki = '0'; ki <= '9'; ++ki)
                for (int li = '0'; li <= '9'; ++li)
                    for (int mi = '0'; mi <= '9'; ++mi)
                        for (int ni = '0'; ni <= '9'; ++ni)
                            for (int oi = '0'; oi <= '9'; ++oi)
                                for (int pi = '0'; pi <= '9'; ++pi)
                                {
```

```
                                    kulcs[0] = ii;
                                    kulcs[1] = ji;
                                    kulcs[2] = ki;
                                    kulcs[3] = li;
                                    kulcs[4] = mi;
                                    kulcs[5] = ni;
                                    kulcs[6] = oi;
                                    kulcs[7] = pi;

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

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

    return 0;
}
```

és a több magos változat:

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

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

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

    return (double) titkos_meret / sz;
}

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

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

    int kulcs_index = 0;

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

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

    }

}

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

    char *buffer;

    if ((buffer = (char *)malloc(sizeof(char)*titkos_meret)) == NULL)
    {
        printf("Memoria (buffer) faliora\n");
        exit(-1);
    }

    exor (kulcs, kulcs_meret, titkos, titkos_meret, buffer);

    if (tiszta_lehet (buffer, titkos_meret))
    {
        printf("Kulcs: [%c%c%c%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
               kulcs[0],kulcs[1],kulcs[2],kulcs[3],kulcs[4],kulcs[5],kulcs ↩
                  [6],kulcs[7], buffer);
```

```
    }

    free(buffer);

}

int
main (void)
{

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

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

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

    // osszes kulcs eloallitasa
#pragma omp parallel for private(kulcs)
    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;

                                    exor_tores (kulcs, KULCS_MERET, titkos, ↩
                                        p - titkos);
```

```
                                        }

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat...

az atlagos_szohossz függvény bekéri a titkos szöveget és méretét, majd egy 'sz' változóban megszámolja, hogy hány szóköz található a szövegben. Ezután a szöveg teljes méretét elosztja az 'sz' változó értékével, így megkapva az átlagos szóhosszt.

az strcasestr függvény azt keresi, hogy a titkos-ban megtalálható-e a **2. paraméterben megadott szöveg** eg.:"hogy" és "nem"

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

```
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
# https://youtu.be/Koyw6IH5ScQ

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])
```

Tanulságok, tapasztalatok, magyarázat...

Ez a program a neurális hálóra alapszik. **A neurális hálózat biológiai neuronok összekapcsolt csoportja. Modern használatban a szó alatt a mesterséges neurális hálót értjük, amelyek mesterséges neuronokból állnak.** forrás: https://hu.wikipedia.org/wiki/Neur%C3%A1lis_h%C3%A1l%C3%B3zat

Itt a library(neuralnet) sor hasonlóképpen működik, mint a #include parancs a C nyelvekben. A számításokért ez a könyvtár felel. Ezek a neurális hálók egyfajta ai-ként tekinthetőek, azaz megtanítjuk a számítógépnek, hogy az egyes kapukat felismerje.

4.6. Hiba-visszaterjesztéses perceptron

C++

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

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

A forráskódot a hossza miatt nem tenném bele a könyvbe, de kódsnippet-eket fogok használni.

```
//main.cpp fájl tartalma
#include <iostream>
#include "mlp.cpp"
#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;
}
```

Tanulságok, tapasztalatok, magyarázat...

A program lényegében annyit csinál, hogy végigfut a bemeneten, és megszámolja a piros pixeleket. Ezt a két egymásba épített forciklusban láthatjuk. az első ciklus **for(int i {0};i<png_image.get_width();++i)** 0-tól a kép szélességéig fut, míg a második ciklus **for(int j{0};j<png_image.get_height();++j)** a kép magasságáig fut, a ciklus belsejében található maga a számolási művelet. **image[i*png_image.get_Width()+j]= png_image[i][j].red;** A kép szélességét szorozza i-vel és hozzáad j-t, és ez lesz az **image** változónk indexe.

ezen változót egyenlővé tesszük a png_image[i][j] kép piros(red) tagjával. ezután egy value változóban eltároljuk az image változó értékét, amely perceptronra mutat. Végül kiírjuk a **value** értékét. a lefordításhoz ezt kell beírni:**g++ mpl.hpp main.cpp -o perceptron -lpng -std=c++11**

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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

A Mandelbrot halmaz a komplex síkon

5.1. ábra. A Mandelbrot halmaz a komplex síkon

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

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

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

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

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

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

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

Megoldás forrása:

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

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040  ↩
   -0.019473810573093663922605855987058021128l8  ↩
   -0.019473810572541341845642648422654019668 7  ↩
   0.798505756933826860155534177465597167611 1  ↩
   0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913 1  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327097512154281  ↩
   0.213538705180497528912653137922461610287 4
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.


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

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

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

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

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

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

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

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

```
    {
      // k megy az oszlopokon

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

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

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

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

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

              ++iteracio;
            }

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

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

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

}
```

A kep.set_pixel függvény felel a kép elkészítéséért, míg a benne lévő rgb_pixel a színének módosításáért.

5.3. Biomorfok

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

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

A biomorfokra (a Julia halmazokat rajzoló bug-os programjával) rátaláló Clifford Pickover azt hitte természeti törvényre bukkant: https://www.emis.de/journals/TJNSA/includes/files/articles/Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf (lásd a 2307. oldal aljától).

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

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

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

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

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

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

            ++iteracio;
          }
```

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

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

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

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

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

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

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

    int szelesseg = 1920;
    int magassag = 1080;
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása: https://github.com/ghjbku/DE/tree/master/sajat/biomoprhs

Ezen feladat megoldása hihetetlenül sokáig tartott, és ez alatt nem a program megírását értem, hanem annak lefordítását. a legnagyobb fejfájást a makefile legenerálása okozta, viszont hosszas keresgélés után ráleltem a megoldásra, ami egyetlen sor:**sudo apt-get install qt5-default** ezután jöhet a **qmake -project**

parancs, amivel létrehozzuk a *.pro fájlunkat, ebből lesz a make fájl. A makefile legenerálása ezen parancs beírásával történik: **qmake profájlneve.pro**, majd **make** és már futtatható a program. a képet a jobb egérgomb használatával lehet nagyítani.

5.6. Mandelbrot nagyító és utazó Java nyelven

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

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

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 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[] args) {

        PolárGenerátor g = new PolárGenerátor();

        for(int i=0; i<10; ++i)
            System.out.println(g.következő());

    }

}
```

Megoldás forrása C++-ban:

```
#include "std_lib_facilities.h"


class PolarGenerator{

bool nincsTarolt=true;
double tarolt;

 public :
double kovetkezo()
 {


  if(nincsTarolt)
  {
  double u1,u2,v1,v2,w;
  u1= ((double) rand() / (double)(RAND_MAX));
  u2= ((double) rand() / (double)(RAND_MAX));
  v1=(2*u1)-1;
  v2=(2*u2)-1;

  w=(v1*v1)+(v2*v2);
   while(w>1)
```

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

};

int main()
{
std::srand(std::time(0));
PolarGenerator g;
for(int i=0; i<10; ++i)
  std::cout<<g.kovetkezo()<<std::endl;
return 0;
}
```

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! A java forrás szemantikailag megegyezik a JDK forrásokkal.

A Java kódhoz képes van egy kis változtatás a kódban. Már az elején található egy különbség, a **public** kulcsszó, amely Java-ban minden függvény elé bekerül, amit public-ként szeretnénk kezelni, amíg C++-ban egy egyszerű **public:** kulcsszó után bármennyi függvényt deklarálhatunk, az minden publikus lesz.

Egy másik különbség, hogy amíg Java-ban a matematikai műveleteket a **Math.művelet** előtaggal hívjuk meg, addig C++-ban az összes ilyen művelet függvényként van beépítve egy cmath header fájlba, így csak a függvény neveit kell meghívnunk. Viszont hátrányként tekinthető, hogy az **#inculde <cmath>** sor nélkül egy művelet se használható.(persze az alapműveletek kivételek: +-*/%)

6.2. LZW

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

Megoldás videó:

Megoldás forrása:

```
// z.c
//
// LZW fa építő
// Programozó Páternoszter
//
```

```
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail ↩
   .com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthetõ illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszõleges) késõbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1, http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2, csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3, http://progpater.blog.hu/2011/03/05/ ↩
   labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
//

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

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

```
} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
  BINFA_PTR p;

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

extern void 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;

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

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

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

  extern int max_melyseg, atlagosszeg, melyseg, atlagdb;
  extern double szorasosszeg, atlag;
```

Katt a továbbra a teljes forrásért:

```
  printf ("melyseg=%d\n", max_melyseg-1);

  /* Átlagos águhossz kiszámítása */
  atlagosszeg = 0;
  melyseg = 0;
  atlagdb = 0;
  ratlag (gyoker);
  // atlag = atlagosszeg / atlagdb;
  // (int) / (int) "elromlik", ezért casoljuk
  // K&R tudatlansági védelem miatt a sok () :)
  atlag = ((double)atlagosszeg) / atlagdb;

  /* Águhosszak szórásának kiszámítása */
  atlagosszeg = 0;
  melyseg = 0;
  atlagdb = 0;
  szorasosszeg = 0.0;

  rszoras (gyoker);

  double szoras = 0.0;

  if (atlagdb - 1 > 0)
    szoras = sqrt( szorasosszeg / (atlagdb - 1));
  else
    szoras = sqrt (szorasosszeg);
```

```
  printf ("altag=%f\nszoras=%f\n", atlag, szoras);

  szabadit (gyoker);
}


 // a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
 // http://sourceforge.net/projects/javacska/
 // az atlag() hivasakor is inicializalni kell oket, a
 // a rekurziv bejaras hasznalja
int atlagosszeg = 0, melyseg = 0, atlagdb = 0;

void
ratlag (BINFA_PTR fa)
{

  if (fa != NULL)
    {
      ++melyseg;
      ratlag (fa->jobb_egy);
      ratlag (fa->bal_nulla);
      --melyseg;

      if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
  {

    ++atlagdb;
    atlagosszeg += melyseg;

  }

    }

}

 // a Javacska ONE projekt Hetedik Szem/TudatSzamitas.java mintajara
 // http://sourceforge.net/projects/javacska/
 // az atlag() hivasakor is inicializalni kell oket, a
 // a rekurziv bejaras hasznalja
double szorasosszeg = 0.0, atlag = 0.0;

void
rszoras (BINFA_PTR fa)
{

  if (fa != NULL)
    {
      ++melyseg;
      rszoras (fa->jobb_egy);
      rszoras (fa->bal_nulla);
```

```
      --melyseg;

      if (fa->jobb_egy == NULL && fa->bal_nulla == NULL)
  {

    ++atlagdb;
    szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));

  }

    }

}

//static int melyseg = 0;
int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
  max_melyseg = melyseg;
      kiir (elem->jobb_egy);
      // ez a postorder bejáráshoz képest
      // 1-el nagyobb mélység, ezért -1
      for (int i = 0; i < melyseg; ++i)
  printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
          ,
        melyseg-1);
      kiir (elem->bal_nulla);
      --melyseg;
    }
}

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

6.3. Fabejárás

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

Megoldás videó:

Preorderbejárás: azaz a gyökér elem majd a bal oldali részfa preorder bejárása, végül ajobboldali részfa preorder bejárása.

Inorderbejárás: azaz először a bal részfa inorder bejárása, majd a gyökérelem, végül ajobboldali részfa inorder bejárása.

Postorderbejárás: azaz először a bal részfa posztorder bejárása, majd a jobboldali részfaposztorder bejárása, végül a gyökérelem feldolgozása.

forrás

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 nagy forráskód miatt csak linkként jelenítem meg

```
protected:      // ha esetleg egyszer majd kiterjesztjük az osztályt, mert
    // akarunk benne valami újdonságot csinálni, vagy meglévő tevékenységet ↩
        máshogy... stb.
    // akkor ezek látszanak majd a gyerek osztályban is

    /* A fában tagként benne van egy csomópont, ez erősen ki van tüntetve,  ↩
       Ő a gyökér: */
    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: z3a7_gyoker.cpp

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

```
LZWBinFa ( LZWBinFa && regi ) {
          std::cout << "LZWBinFa move ctor" << std::endl;

          gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
          gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

          regi.gyoker.ujEgyesGyermek ( nullptr );
          regi.gyoker.ujNullasGyermek ( nullptr );

     }
```

7. fejezet

Helló, Olvasónapló!

7.1. Juhász István féle könyv

Egy számítógép programozására három nyelvi szintet különböztetünk meg:

-Gépi kód

-assembly szint

-Magas szint

A magas szintű nyelveken megírt algoritmusokat forráskódoknak nevezzük. A forráskódok nyelvtani szabályi a szintaktikai szabályok, míg a jelentésbeli, tartalmi szabályzat a szemantika. Ezeket a kódokat interpreterrel, vagy fordítóprogrammal gépi kóddá kell konvertálni, hogy a processor értelmezni tudja. Egy fordítóprogram tetszőleges nyelvről tetszőleges nyelvre fordít.Amíg ez az egész kódból egy tárgyprogramot készít, addig az interpreter értelezi és rögtön lefuttatja a kódot, programfájl nélkül.

7.2. FerSML Prog1_1

A híres legelső program 3 típusban: 1. könyvtári hívással:

```
#include <stdio.h>

int main(){

printf("Helló, olvasónapló!\n");
return 0;
}
```

2. rendszer hívással:

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

write (1, "Helló megint!\n",16);
```

```
return 0;
}
```

3. assembly kóddal:

```
.data

hello:
  .ascii "hello, olvasonaplo!\n"
.text
  .global _start

_start:
  movl $4, %eax
  movl $1, %ebx
  movl $hello, %ecx
  movl $22, %edx

  int $0x80

  movl $1, %eax
  movl $0, %ebx

  int $0x80
```

itt a _start parancs hasonló feladatot lát el, mint a C típusú nyelveknél a main() függvény az int $0x80 kéri a processor megszakítást. Maga a 0x80 a megszakítás vektor, amely a rendszerhívást végzi a kernelből.

A könyvben említésre kerülnek a turing gépek is.

7.3. FerSML Prog1_2

A google által tervezett és kivitelezett PageRank algoritmusról szól a fejezet. Ez az algoritmus arra hivatott, hogy a weblapok minőségét rangsorolja. The PageRank Citation Ranking: Bringing Order to the Web.http://dbpubs.stanford.edu:8090/pub/1999-66/

8. fejezet

Helló, Conway!

8.1. 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...

8.2. Java életjáték

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.4. BrainB Benchmark

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Gutenberg!

9.1. Programozási alapfogalmak

[?]

9.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

9.3. Programozás

[BMECPP]

10. fejezet

Helló, Schwarzenegger!

10.1. Szoftmax Py MNIST

aa Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.2. Szoftmax R MNIST

R

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.3. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.4. Deep dream

Keras

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

10.5. Robotpszichológia

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11. fejezet

Helló, Chaitin!

11.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó:

Megoldás forrása:

11.2. Weizenbaum Eliza programja

Éleszd fel Weizenbaum Eliza programját!

Megoldás videó:

Megoldás forrása:

11.3. 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...

11.4. 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...

11.5. Lambda

Hasonlítsd össze a következő programokat!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.6. 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.

12. fejezet

Helló, Arroway!

12.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

12.2. 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

12.3. Általános

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

12.4. C

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

12.5. C++

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

12.6. 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.