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	Kiss, Máté	2019. március 20.	

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

A végtelen ciklus egy olyan ciklus, amelyben a feltétel állandóan adott(igaz), ezért a ciklus nem lép ki, hanem újra és újra lefut. például for(;;)

Vannak ciklusok amik a szálakat 100%-ban vagy 0%-ban dolgoztatják.

0%-ban dolgoztat:

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

int main(){
while(1){
    sleep(100);
 }
}
```

Magyarázat: Az unistd header tartalmazza a sleep() függvényt. Ezért kell include-olni az stdio.h header (standart input/output) mellett. Az int main() a fő függvényünk. A while() pedig a ciklus. A () belülre kell írnunk a feltételt. Amíg ez igaz , a ciklus újra és újra lefut. A példánkban a ciklusban az 1 szám szerepel. Ez az érték mindig igazat ad vissza, tehát a ciklus állandóan újraindul amíg ki nem lőjjük. A sleep(100) függvény pedig azért kell, mivel ez altatja a processzor folyamat szálát. A függvényben megadott érték jelenti azt, hogy hány másodpercig altatja a processzort.

100%-ban dolgoztat egy szálat:

```
#include <stdio.h>
#include <unistd>
int main(){
while(1){
 }
}
```

Magyarázat: Az előző példától nem sokban tér el. Az include-k és a ciklus magyarázata megegyezik, az előző példáéval. Itt annyi a különbség, hogy nincs benne a sleep() függvény, azaz a szál nincs altatva. Így a végtelen ciklus 100%-ban dolgoztat 1 szálat.

100%-ban dolgoztat minden szálat:

```
#include <stdio.h>
#include <unistd>
#include <omp.h>
int main(){
#pragma omp parallel
while(1){
}
}
```

Magyarázat: A programunk, az előzőhöz egy openmp-vel bővült. Ezzel az include-val belépünk, a párhuzamos programozás küszöbére. #pragma omp parallel sor adja azt az utasítást a gépnek, hogy a feladat az összes szálon fusson. (A fordításnál -fopenmp kapcsolóval kell bővítenünk a parancsot.)

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 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: Ha úgy vesszük, hogy a T100 és T1000 létező program és T1000 ben meghívjuk saját magát. A t100 alapján ha a programunkba van végtelen ciklus, akkor igaz értéket ad a Lefagy program a Lefagy2 programnak ,így tehát az is igaz értéket fog adni, viszont ha a Lefagy false értéket ad vissza akkor a Lefagy2 belém egy végtelen ciklusban, tehát a program le fog fagyni. Tehát olyan program mint a T100 nem működik mivel, ha egy olyan program érkezik bele amiben van végtelen ciklus, akkor a program beáll mert a ciklus nem áll meg.

2.3. Változók értékének felcserélése

A feladat két változó értékének felcserélése. Például a=1 , b=2 , ebből lesz a megoldás, hogy a=2, b=1. Napjainkba a számítógép fejlettsége és gyorsasága miatt, már egyszerűen megcsinálhatjuk egy segédváltozóval vagy exort-tal, de régen nagyon sokat számított az erőforrások jó felhasználása, elosztása. Ezért ezek a megoldásoknál sokkal könnyebb volt a számítógépeknek számolni, ha különbséggel vagy szorzással cseréltük fel a változókat. Az utóbbi kettőt nézzük most meg:

Változócsere különbséggel:

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

int main()
{
    int a=1;
    int b=2;

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

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

Magyarázat: A fejlécet már ismerjük az előző feladatból. A printf() függvény a kiíratáshoz kell majd nekünk. az első argumentum a kíratás formátuma, a többi pedig a változók kiíratása. A „%d” azt jelenti, hogy egy egész típusú változót fogunk kiíratni, még a „\n” a sortörést jelenti. Maga a feladat egyszerű matematika. Legegyszerűbben a példával lehet megérteni.

a=1, b=2

a=1-2=-1 „a” értéke -1 lesz.

b=-1+2=1 „b” értéke 1 lesz, ami az „a” értéke volt.

a=1-(-1)=2 az „a” értéke 2, ami a „b” értéke volt

Kész is a cserénk.

Változócsere szorzattal:

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

int main()
{
    int a=1;
    int b=2;

    printf("%s\n%d %d\n","szorzassal:",a,b);
```

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

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

Magyarázat: A megoldás itt annyiban különbözik, hogy nem „+” és „–„ -t használunk hanem „*” és „/” -t. Példa:

a=1, b=2

a=1*2=2 „a” értéke 2 lesz.

b=-2/2=1 „b” értéke 1 lesz, ami az „a” értéke volt.

a=2/1=2 az „a” értéke 2, ami a „b” értéke volt.

Kész is a cserénk.

2.4. Labdapattogás

```
#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, "O" );

        refresh ();
        usleep ( 100000 );

        x = x + xnov;
```

```
        y = y + ynov;

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

    }

    return 0;
}
```

Magyarázat: Az új dolog ami a fejlécnél feltűnik az a curses.h header. Ez képernyő kezelési függvényeket tartalmaz, és a program megjelenítéséhez szükségünk van rá.

A következő részlet:

```
    WINDOW *ablak;
    ablak = initscr ();
```

Így formázzuk meg a kimenetet. Az initscr () függvény curses módba lépteti a terminált.

A deklarált x és y -on lesz a kezdő értékünk. Az xnov és ynov pedig a lépésközöt mutatja. (lépésenként a koordináta rendszeren xnov, ynov-al való elmozdulást). Az mx és my lesznek a határértékek, hogy a program csak az ablakon belül mozogjon.

A végtelen ciklus következtében, a labda addif pattog, amíg ki nem lőjük a programot. A ciklusban az első függvény a getmaxyx () . Ez határozza meg,hogy mekkora az ablakunk mérete. refresh() függvénnyel frissítjük az ablakot. Közöttük a mvprintw() függvény az x és y tengelyen megrazolja a „ „ között lévő szöveget, számot vagy karaktert, esetünkben az O-t. Az usleep függvény azt szabályozza mennyi ideig altassa a ciklust még újra indul, azaz milyen gyorsan pattogjon a labda.

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

Megnöveljük az értékeket, minden ciklus lefutásnál (mozog a "labda").

A kővetkező négy if-el pedig azt vizsgáljuk, hogy a labda az ablak szélén van e, ha igen akkor -1 -el szorozzuk, ezáltal a labda irányt változtat. A fordításnál -lncourses kapcsolót kell használnunk.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Szóhossz:

```
#include <stdio.h>

int main()
{
   int a=1;
   int bit=0;
   do
   bit++;
   while(a<<=1);
   printf("%d %s\n",bit,"bites a szohossz a gepen");
}
```

Ez a program a gépünk szó hosszát fogja kiírni, azaz az int méretét. A feladatot a BogoMIPS ben használt while ciklus feltétellel írjuk meg (A BogoMIPS a processzorunk sebbeségét lemérő program amit Linus Torvalds írt meg).

A main függvényben az első sor az int a=1. Itt deklaráljuk a változót, amivel vizsgáljuk meg a gépünk szóhosszát(Az int méretét). A "bit" változó fogja a lépéseket számlálni. A programot dowhile ciklussal(hátultesztelős) futtatjuk, mivel a sima while nem számítaná bele az első lépést, tehát ha a gépünk 32 bites, a program 31 bitet írna. A ciklus addig fut amíg az "a" nem lesz egyenlő nullával. És akkor mi is az a bitshift operátor. Ugye vesszük az 1 et, a=1. ennek a Bináris kódja a 0001, a bitshift operátor egy 0 -val eltolja, azaz 0010 kapjuk, ez a 2 szám, a count növekedik tehát az értéke 1 lesz. A ciklus újra lefut és eltolja még egyszer a számot egy 0-val, így 0100 kapunk ami a négy. Ez addig fut, még a gépünk szó hosszán (az int méretén) kívül nem tolja az 1-est. Ekkor az a értkében csak 0 fog szerepelni, azaz az "a" értéke 0 lesz, a while ciklus befejeződik, és kiíratjuk hányat lépett a ciklus, és ez a szám adja meg, hogy hány bites a szóhossz.

2.6. Helló, Google!

A PageRank egy keresőmotor amit a Google használ. A programot két fiatal írta meg 1998-ban. Nevét az egyik kitalálója után kapta.

A következőben, egy 4 lapból álló PageRank-at fogunk megnézni. A lapok PageRank-ét az alapján nézzük, hogy hány oldal osztotta meg a saját honlapján az oldal hiperlinkjét.

```
#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;
}
```

Kezdjük az új headerrel, ez a math.h. Ez tartalmazza a matematikai számításokhoz szükséges függvényeket. A main() fügvénnyben először is létrehozunk egy mátrixot, ami a lapok összeköttetését adja meg. Ha az érték 0 akkor a lap nincs összekötve az adott lappal és persze önmagával sincs. Ahol 1/2 vagy 1/3 az érték az azt jelzi, hogy az oldal hány oldallal van összekötve, például az 1/2: Az oldal 2 oldallal van összekötve és abbol az egyik kapcsolatot jelzi (az 1).

A PR tömb fogja a PageRank értéket tárolni. A PRv tömb pedig a mátrixal való számításokhoz kell. A következő lépés egy végtelen ciklus.Ez majd a számítások végén a break parancsal lép ki, ha a megadott feltétel teljesül. A forciklusban van maga a PageRank számítása ami majd a tavolság függvényt is meghívja, ami egy részszámolást tartalmaz. A végtelen cikluson belül lévő ciklusok azért 4 ig mennek mert 4 oldalt nézünk. A ciklusbol való kilépés a "break" parancsal történik majd ha a tavolsag függvényben kapott eredmény kisebb mint 0.00000001. A végén a kiir függvény megkapja a PR értékeket és az oldalak számát és kiíratja azokat.

2.7. 100 éves a Brun tétel

A tételt Viggo Brun bizonyította 1919-ben. Ezért is nevezték el róla. A tétel kimondja hogy az ikerprímek reciprokösszege a Brun konstanthoz konvergál, ami egy véges érték.

Brun tétel R szimulációban:

```
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 számoláshoz elősször is kell egy matlab könyvtár. A program fő része az stp függvény. egy a függvény megkapja x-et. X egy szam lesz ami megmondja meddig kell a prímeket számolni. Ehez a primes függvényt használjuk. primes(x) kiírja x-ig a prímeket. A diff vektorban eltároljuk a primes vektorban tárolt egymás melletti prímek különbségét. A számítást úgy végezzük, hogy a 2 prímtől indulva kivonjuk a prímből az előtte lévő prímet. Az idx el vizsgaljuk meg, hogy mely prímek különbsége 2 és ezek hol vannak (a helyüket a which függvény adja meg). a t1primes vektorban elhelyezzük ezeket a prímeket. A t2primes vektorba pedig ami ezeknél a prímeknél kettővel nagyobb (azaz ikerprímek). rt1plust2 vektorban végezzük a recikropképzést és a pár reciprokát összeadjuk. A returnban pedig a sum függvénnyel vissza adjuk ezek summázott összegét. Végezetül a plot függvénnyel lerajzoljuk grafikusan.

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

2.8. A Monty Hall probléma

Ez egy valószínűségi paradoxon. A kérdés egy vetélkedő játékból indul. Van 3 ajtó és az egyik mögött egy értékes nyeremény van, a másik kettő mögött semmi. A versenyzőnek a 3 ajtó közül választania kell egyet. Miután választott, a műsorvezető kinyit egy ajtót, ami mögött nincs a nyeremény. Felteszi a kérdést, hogy akarunk e változtatni a választásunkon. Itt jön a felvetés, hogy megéri e változtatni, vagy nem.

Megoldás:Első ránézésre mi is, és szinte mindenki azt mondaná, hogy nem számít, hogy vált e mert 50-50% az esélye, hogy melyik ajtó mögött van a nyeremény. Mivel már nem 3 hanem 2 ajtó közül lehet választani, így már figyelembe se veszik azt a harmadik ajtót. De a megoldás az, hogy igen, nagyobb az esélyünk akkor ha az előző döntésünket megváltoztatjuk és a másik ajtót választjuk.

Magyarázat: Kezdetben 3 ajtóból 1 ajtót kell választanunk, azaz 1/3 az eséyle , hogy eltaláljuk a jó megoldást és 2/3 hogy nem. Ezek után a műsorvezető kinyit egy ajtót ami mögött nincs a nyeremény. Ez a kezdeti valószínűségen nem változtat, úgyanúgy 1/3 eséllyel választottuk azt az ajtót ami mögött a nyeremény van. Viszont azok az ajtók közül ami mögött nincs semmi, azokból már csak az egyik van csukva. Biztosra tudjuk, hogy a nyeremény a maradék két ajtó közül valamelyik mögött van. Tehát 2/3 az esélye annak, hogy a másik ajtó mögött van a nyeremény.

Szimuláció:

```
kiserletek_szama=10000
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)
```

Most a kisérletet 10000x fogjuk szimulálni. a kiserlet vektorban 1 és 3 "ajtó" közül választunk 10000x. A replace=T-vel tesszük lehetővé, hogy egy eredmény többször is kijöhessen. A játékos valasztásait a jatekos vektornál ugyan így meghatározzuk.A sample() fügvénnyel végezzük aa kiválasztást. A musorvezeto vektort a length függvényel a kisérletek számával tesszük egyenlővé. Következik a for ciklus ami i=1 től a kisérletek számáig fut (100). A ciklusban egy feltétel vizsgálat következik. az if-fel megvizsgáljuk, hogy a játékos álltal választott ajtó megegyezik e a kisérletben szereplő ajtóval. Ha a feltétel igaz egy mibol vektorba beletesszük azokat az ajtokat amiket a játékos nem választott, az else ágon pedig ha a feltétel nem igaz ,akkor azt az ajtót eltároljuk amit nem a választott és a nyereményt rejtő ajtót. A musorvezeto vektorban pedig azt az ajtót amit ki fog nyitni. A nemvaltoztat es nyer vektorban azok az esetek vannak amikor a játékos azt az ajtót választotta elsőre ami mögött az ajtó van és nem változtat a döntésén. A valtoztat vektorban pedig azt mikor megváltozatja a döntését és így nyer ezt egy forciklussal vizsgaljuk. A legvégén kiíratjuk az eredményeket, hogy melyik esetben hányszor nyert.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Magát a fép fogalmát 1936 -ban Alan Turing alkotta meg. A gép decimális számrendszerből unáris számrendszerbe írja át a számot. Az unáris számrendszer másnéven egyes számrendszer. A lényege, hogy 1 eseket írunk csak, ha az 1 számot akarjuk unárisba átváltani, az értéke egy, ha a 2-őt akkor az értéke 11, a tíz pedig 1111111111, Az a program c++ ban a következő:

```
#include <iostream>

using namespace std;

int main()
{
  int a;
  int tiz=0, szaz=0;
  cout<<"Decimalis szam:\n";
  cin>>a;
  cout<<"A szam unarisban:\n";
  for (int i=0; i<a; i++){
    cout<<"1";
    ++tiz;
    if (tiz==10) {cout<<" "; tiz=0;}
    if (szaz==100){cout<<"\n";szaz=0;}
  }
  return 0;
}
```

A kód egyszerű. Bekérünk egy decimális számot "a"-ba, és egy forciklus segitségével addig irunk mindig egy 1-est amíg i(ami kezdetben 0 és mindig egyel növeljük) kisebb mint a. Én hogy a kimenet szebb legyen 2 változót használtam, 10 db 1 es után egy szóközt teszünk, míg 100 db után egy sortörést.

Magyarázat az Állapotmenet grafikájának:

A gép beolvassa a memoríaszalag számjegyeit, (Az ábrán a szám a 10) ha elér az "=" ig, az előtte lévő számmal kezd el dolgozni, még az 0 nem lesz. Az első elem egy 0, de mivel a következő nem nulla, hanem

1 ezért ebből kivon 1 et, azaz hátulról a második elemet 0 ra állítja. a kezdő elem ami 0 volt, az pedig 9 lesz, és ebből mindig kivon egyet még 0 nem lesz, (8,7,6,5,4,3,2,1,0), minden kivonásnál kiírat sorban 1 est, így annyi 1 lesz, mint a decimális szám értéke. (Ha 100 lenne a szám akkor az 100 után 099 lenne aztán 098,097....089,088 és így megy 000 ig, és a kimeneten 100db 1 es lesz.)

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

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.3. Hivatkozási nyelv

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

Ahogy a beszélt nyelv, úgy a programozási nyelv is fejlődik. Ennek a bemutatására az alábbi programot fogjuk használni:

```
#include <stdio.h>

int main(){
  for(int i=0;i<1;i++){
  printf("Lefut");
 }
}
```

Itt ami lényeges, nem a kódban lesz, hanem a fordításnál. Megvizsgáljuk, hogy a C89 es nyelvtan és a C99-es szerint hogyan fordítja le a programot a fordító. Ha a C89 es nyelvtannal fordítom: "gcc -std=gnu89 fajlnev.c -o fajlnev". A program hibát fog írni a for ciklusnál. Most ha a fordításnál átírjuk "gcc -std=gnu99 fajlnev.c -o fajlnev"-re (Azaz a fordító a 99 nyelvtan lesz) ,akkor láthatjuk, hogy lemegy a fordítás és a program működik. A kódon belül, a for ciklusban deklaráltuk az int i-t.

magyarázat: Az okot a kódon belül, a for ciklusban kell keresni,ugyanis az "i" -t a forcikluson belül deklaráltuk. A C89 nyelvtanban ez még nem volt megengedett, így a fordító hibát írt, de a C99-ben már igen, ezért nem jelez hibát.

3.4. Saját lexikális elemző

A program a bemeneten megjelenő valós számokat összeszámolja.

A lexikális elemző kódja:

```
 %{
 #include <string.h>
 int szamok=0;
%}
%%
[0-9]+    {++szamok;}
%%

int
main()
{
 yylex();
 printf("%d szam",szamok);
return 0;
}
```

A szamok változóval számoljuk hányszor fordul elő szám a bemenetben. A programot a % - jelekkel osztjuk fel részekre. a

```
[0-9]+    {++szamok;}
```

Ez a sor adja azt, hogy 0-9 vagy nagyobb számot talál akkor növelje a "szamok" valtozót. A printf el pedig csak kiíratjuk hogy hány szám volt a bemenetben(ez az elemzés). A yylex() a lexikális elemző

a fordítás a következő:

```
flex program.l
```

ez készít egy "lex.cc.y" fájlt. ezt az alábbi módon futtatjuk.

```
cc lex.yy.c -o program_neve -lfl
```

A futtatáshoz pedig hozzá kell csatolni a vizsgált szöveget.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

```
%{
 #include <string.h>
 int szamok=0;
%}
%%
"0" {printf("o");}
"1" {printf("i");}
"3" {printf("e");}
"4" {printf("a");}
"5" {printf("s");}
"7" {printf("t");}
```

```
"o" {printf("0");}
"i" {printf("1");}
"e" {printf("3");}
"a" {printf("4");}
"s" {printf("5");}
"t" {printf("7");}

%%

int
main()
{
 yylex();
 printf("%d szam",szamok);
return 0;
}
```

Ez a program lefordítja a l33t nyelven írt titkos szöveget vagy a rendes szöveget írja át a l33t nyelvre.

A program müködése az előzővel majdnem megegyezik, csak annyiban tér el, hogy valós számok helyett, itt most a megadott számokat keresi a bemenetben és azok a számok helyett a l33t nyelvben való megfelelő betűket írja a helyére. Ha pedig a l33t nyelvre akarjuk fordítani, akkor a betűket vizsgálja és a megfelelő számot írja be.

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);
```

A példában szereplő kód részlet ellentetje, azaz ha a SIGNT jel kezelése nem lett figyelmen kívül hagyva akkor, a jelkezelő függvény kezelje.

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

Ez egy forciklus, benne az i értéket 0 állítjuk, és amíg az i értéke kisebb mint 5 addig a ciklus újra és újra lefut. A 3 argumentumban növeljük az i értékét.

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

A ciklus majdnem ugyan az mint az előző. az eltérés az i érték növelésében van, nem tünik nagy eltérésnek, de fontos. Az előzőbe ++i míg itt i++. A különbség az, hogy a ++i nél először növeli az i értékét, aztán az i értékét átadja, még az i++ először átadja az i értékét és aztán növeli az i értékét. Ez a forciklusban úgy van ++i -nél hogy megnöveli egyel az i-t és utánna hajtja végre újra a lefutást. Az i++ nál pedig elöször végre hajtja aztán növeli az i értékét.

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

Ez a forciklus egy tombot feltölt az i értékével. a tomb ezek után úgy fog kinézni, hogy tomb[5]={0,1,2,3,4}, mivel i++, ezért előbb átadja az értéket és utánna növeli az i értékét. Bug: A programba máshogy viselkedik , mivel az eslő érték mindig egy memóriaszemét lesz. A megoldás, hogy a forcikluson belül adjuk hozzá a tombhoz az értéket for(){ ezen belül }.

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

Itt a forciklusunk második argumentumába az i kisebb mint n feltétel mellett van egy másik feltétel. A forciklus csak akkor fut le ha mind a 2 feltétel teljesül. A második feltétel az, hogy az s és a d mutató egyenlő (minden ciklusnal növeljük az értékeket). A feltételt az és operátorral kötjük össze. Bug: A hiba, hogy a második feltétel nem logikai feltétel. Ezt a feltétel is egy if el a forcikluson belül kéne vizsgálnunk.

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

A printf fügvénnyel kiíratunk valamit. Ebben az esetben két egész tipusú változót. A printf-en belül az f függvénnyel határozzuk meg a számot. Bug: Rossz a sorrend, ezért hibát kapunk.

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

A printf fügvénnyel kiíratunk két egész számot, az első számot az f függvény adja (az f függvény az "a"-t kapja meg), míg a másik az a változó értéke.

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

A printf fügvénnyel kiíratunk két egész számot. Az előzőnél annyival másabb, hogy a függvény az a memória címét kapja meg.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

```
$(\forall x \exists y ((x<y)\wedge(y \text{ prím})))$
Minden x esetén létezik olyan y ahol x<y és y prím. Ez azt jelenti, hogy a ↩
   prímek száma végtelen.

$(\forall x \exists y ((x<y)\wedge(y \text{ prím})\wedge(SSy \text{ prím})) ↩
   )$
Minden x esetén létezik olyan y ahol x<y és y prím és az SSy is prím, ↩
   lefordítva azt jelenti, hogy az ikerprímek száma végtelen.

$(\exists y \forall x (x \text{ prím}) \supset (x<y)) $
Létezik olyan y, minden x számra, hogy ha x prím akkor x<y , lefordítva a ↩
   prímek száma véges.

$(\exists y \forall x (y<x) \supset \neg (x \text{ prím}))$
Létezik, olyan y ami minden x számra y<x akkor hax nem prím, lefordítva ↩
   ugyan azt jelenti mint az előző , csak tagadással megfogalmazva.
```

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

A megoldáshoz tudnunk kell mi mit jelent, vannak a logikai összekötőjelek, mint az és=\wedge, \neg=nem ,\vee=vagy, \supset=implikáció A kiíratást a \text el végezzük. Vannak kvantorok a "létezik"=\exists és a "minden"=\forall. Az "S" értéknövelés.

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

A program:

```
#include <iostream>

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

Mit vezetnek be a programba a következő nevek?

- `int a;`

  Egy egész tipusú változót deklarál.
- `int *b = &a;`

  Egy int tipusú mutatót deklarál, ami képes egy változó memóriacímét tárolni. "b" mutató "a" ra mutat.
- `int &r = a;`

  Egy egész tipusú referenciát deklarál, ami hasonló a mutatóhoz, de nem ugyan az, a referencia úgymond egy állnév, pontosabban egy már létező változóhoz egy másik név.
- `int c[5];`

  Ez egy egész tipusú 5 elemű tömb.
- `int (&tr)[5] = c;`

  Ez egy referenciája a "c" 5 elemű tömbnek (Az összes elemnek).
- `int *d[5];`

  A d tömbben minden egyes tag egy mutató.
- `int *h ();`

  Az int tipusú változó visszatérési tipusát tartalmazó függvény.
- `int *(*l) ();`

  Egy egész tipusra mutató mutatót visszaadó függvény.

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

  Egy egész tipusút afo és két egész tipusút kapó függvényre mutató mutatót visszaadó, egész tipust kapó függvény
- ```
  int (*(*z) (int)) (int, int);
  ```

  Egy egész típust visszaadó és két egész típust kapó függvényre mutató mutatót visszaadó, egész típust kapó függvényre

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

A következő programban egy alsó háromszögmátrixot hozunk létre.

a kód:

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

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

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

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


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

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

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

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

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

Magyarázat: Szokás zerint includoljuk a szükséges include-kat. A fő függvényben az első sora az "int nr=5" itt adjuk meg, hogy 5 sorunk legyen a kimeneten. A "double **tm", sorral foglalunk le tárhelyet a memóriában. Az első ifben megtaláljuk a malloc függvéynt ami dinamikus memória foglaló, ezzel nr számú double ** mutatót foglalunk le, ha null értéket ad vissza az azt jelzi ,hogy nincs elég hely a foglaláshoz. A következő if lefoglalja a mátrix sorait, az első sornak egy double * mutatót foglal le, a másodiknak 2 , a harmadiknak 3 , nr ig. A 3. for ciklussal megadjuk a mátrix elemeit. Az "i" a matrix sorai, a "j" pedig a benne lévő mutatók. a "tm[i][j]=i*(i+1)/2+j; érjük el azt, hogy az elemek mindig egyel nőjenek. A 4. for ciklus pedig a kííratás. Ezek után már csak annyit csinálunk, hogy a 3 sort megváltoztatjuk, mert így is ki lehet íratni. A legvégén pedig a free()-vel felszabadítjuk a lefoglalt memóriát, ezzel megelőzve a memóriafolyást.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

A feladat lényege , hogy egy szöveget titkosítsunk Exor-ral(XOR). Az XOR a "kizáró vagy". A szöveget az alábbi módon titkosítjuk: Az eredeti sszöveg bájtjaihoz rendelünk titkosító kulcs bájtjtokat. Aztán X.cOR-t műveletet végzunk rajta. Az XOR-t művelet úgy müködik, hogy ha a bitek azonosak (1,1;0,0) akkor 0 ad vissza értéknek, ha pedig külöbzözőek (1,0;0,1) akkor 1 et ad vissza, és így minden bitpáron elvégezve ezt megkapunk egy titkosított szöveget.

Kód:

```
#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);

    }
}
```

A kód magyarázata: Először is beinclude-oljuk szükséges include-kat. Aztán két állandó változót definiálunk a #define parancsal. Ezeknek az értéke nem változik. Az első állandó a MAX_KULCS az értéke 100. A második pedig a BUFFER_MERET 256, ez nekünk a beolvasásnál fog kelleni. A fő fügvényben egy-egy char tipusú tömb méreteivé tesszük a 2 állandót. Ezek után 2 változót hozunk be, a kulcs_index, ami a kulcsunk aktuális elemét tárolja, és az olvasott_bajtok ami a beolvasott bájtok összegét tárolja. A kulcs_merete változóban a kulcs méretét adjuk meg a "strlen()" függvény segítségével, amit mi adunk meg egyik argumentumként. Az strncpy függvény pedig a kulcs kezeléséhez kell. Ezután a while ciklusban beolvassuk a buffer tömbe a bemenetet, a while ciklus addig fut, ameddig van mit beolvasni. A read függvényel lépünk ki a ciklusból. A while cikluson belül a forciklusban végig megyünk az összes bájton és végre hajtjuk a titkosítást.

A futtatás a következő: A fordítás: gcc fajlnev.c -o fajlnev miután lefut, utánna futtatjuk: ./fajlnev 56789012 (ez a kulcs) titkosítando.txt (ide írjuk a titkosítandó txt fajl nevét, relíciós jelek között) > titkos.szoveg (titkosított fajlneve). A titkos szöveget a more titkos.szoveg parancsal nézhetjük meg.

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!

Az alábbi feladatban a 3.2 feladatban lévő titkosítóhoz írunk egy programot ami feltöri a titkosított szöveget. A program alapműködése ugyan azon az elven alapszik, mint a 3.2 mivel ugyan így XOR- al alakítjuk vissza a szöveget. A lényeg, hogy a kulcsot amivel titkosítottunk azt ismerjük, mert ezzel a kulcsal tudjuk feltörni. Úgy működik, hogy a titkosított bájtokat össze exortáljuk a kulcsal, és így újra az eredeti bájtokat kapjuk. A feladatban a 3.2 ben titkosított azöveget és a kulcsot fogjuk használni, ugyanis erre épül a program.

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)
{
  return 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;

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

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

//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);
    exor(kulcs,KULCS_MERET,titkos,p-titkos);
  }
return 0;
```

```
}
```

A kód az UDPROG repójából van. Elsőnek is definiáljuk az állandókat és az include-kat. Az állandók közül most is a buffer a beolvasáshoz szükséges, a kulcs mérete megint a kulcsot tartalmazó tömbhöz kell ami az előzőleg használt kód miatt 8. A fő függvény előtt találunk függvényeket. Az átlagos szóhossz és a tiszta lehet függvény a törés gyorsaságát segítik elő. Az átlagos szóhossz megadja az szóhossz atlagat még a tiszta lehet pedig a gyakori magyar szavak figyeli. A void exor () fügvény megkap egy kulcsot, a méretét, a tiktos szövegetnek a tömbjét és annak a méretét.És itt a forciklusban a kulcsot össze exortálja a titkos szöveggel. Az exor_tores függvény meghívja az exor függvényt is vissza adja a tiszta szöveget. A fő függvényben láthatjuk deklarációk után a titkos szöveg beolvasását.Utánna a program megnézi az összes lehetséges permutációt és a megoldást kííratja a kimenetre, ezzel a kóddal a 3.2 programot használva fel tudjuk törni a szöveget.

4.5. Neurális OR, AND és EXOR kapu

R

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

```
library(neuralnet)

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

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

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

plot(nn.or)

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


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

4.6. Hiba-visszaterjesztéses perceptron

C++

A perceptron a mesterséges inteligenciának olyan, mint az agynak a neuron. A program képes feldolgozni és megtanulni a bemenetet, ami 0,1 ből áll.

Kód:

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

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

```
    Perceptron* p = new Perceptron(3, size, 256, 1);

    double* image = new double[size];

    for(int i {0}; i<png_image.get_width(); ++i)
         for(int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;

    double value = (*p) (image);

    std::cout << value << std::endl;

    delete p;
    delete [] image;
}
```

A kód magyarázata: két headere van szükségünk az "mlp.hpp" és a "png++/png.hpp" -re, ezek a megjeleníté miatt kellenek nekünk és ebbe van a perceptron elve is. A fő fügvényünk elején lefoglaljuk a tárhelyet a képnek és megadjuk a méreteit. Következik a perceptron létrehozása és a megfelelő értékek hozzá adása. A "double* image = new double[size];" sorral a végélétrehozunk egy size méretű képet és utánna feltöltjük a megadott képpel. a delete parancsokkal töröljük a perceptront és a képet.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Megoldás videó:

Megoldás forrása:

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

Megoldás videó:

Megoldás forrása:

5.3. Biomorfok

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

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

Tanulságok, tapasztalatok, magyarázat...

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!

Megoldás forrása:

Megoldás videó:

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.