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	Csontos, Róbert	2019. március 21.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás forrása:https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Turing/Vegtelen_Ciklus

Egy ciklust végtelen ciklusnak nevezünk, ha addig fut, amíg valamilyen külső behatásra meg nem áll. Külső behatás lehet például az ha elmegy az áram és így leáll a számítógép. Az is végtelen ciklus, amikor egy futó program ablakát nyitva tartja az operációs rendszer. Ez akkor áll meg amikor mi bezárjuk az ablakot. Lehet viszont olyan eset is, amikor egy hibából adódóan jön létre. Ilyenkor nem szándékos a végtelen futás, ám külső behatás nélkül ebben az esetben sem fog megállni.

Olyan programot írni ami egy szálat 100%-ra pörget több módon is lehet. Az egyik ilyen mód az alább látott. Amint a program belép a ciklusba megkezdődik a végtelen futás.

Most pedig nézzük a kódot. A jelen látott példában egy **while** ciklust használunk. Ennek a feltételében egy **true** érték látható. Ez a feltétel kimondja hogy addig fusson a ciklus amíg a feltétel igaz. Mivel ez viszont mindig igaz marad, a ciklusmagból soha nem fogunk kilépni. Látható továbbá hogy a kód elején van **#include stdbool.h**. Erre azért van szükség, mivel a C nyelv alapbón nem tud true értéket használni, ezért tehát meg kell hívni hozzá ezt a függvénykönyvtárat.

```
Végtelen ciklus 1 szál 100%
{

#include <stdbool.h>

int main()
{
  while (true) {
  ;
  }
  return 0;
}
}
```

A következő példa azt mutatja hogy egy végtelen ciklus hogyan néz akkor ha a program 0%-ban pörgeti a cpu-t az adott szálon. Az elv az előzőhöz hasonló. Annyi a különbség hogy ebben a kidolgozásban for ciklust használtam ezzel jelezve hogy szándékos a végtelen ciklus. Ahhoz hogy 0%-on pörögjön a cpu a **sleep** parancsot kell használni. Ehhez viszont szükség van az **unistd.h** függvénykönyvtárra. A sleepnél a szám azt határozza meg hogy hány másodpercig sleep-el, viszont a végtelenség miatt ez soha nem fog abba maradni.

```
Végtelen ciklus 1 szál 0%
{

#include <unistd.h>

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

Végül pedig az utolsó kód azt mutatja be amikor az összes szálon 100%-ra pörög a cpu. Ehhez az kell hogy a ciklus elé beillesszük a **#pragma omp parallel** parancsot. Ez biztosítja számunkra azt hogy a ciklus minden szálon fut. Viszont hogy ez működjön az is kell hogy a futtatásnál a **-fopenmp** kapcsolót is használjuk.

```
Végtelen ciklus összes szál 100%
{

#include <stdbool.h>

//gcc vegtelenossz.c -fopenmp -o vegossz

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

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

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

Megoldás 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.

A gondolatmenet az hogy kellene egy olyan program ami ellenőrzi hogy a megadott program lefagy-e vagy nem. Ilyen programot nem lehet készíteni. Elméletben azonban mégis készítsük el azt a programot amely képes erre. Ha lefagy a paraméterül kapott program akkor írja ki hogy lefagy, ha pedig nem akkor indítson egy végtelen ciklust.

A képzelt program elkészült. Most adjuk meg neki önmagát. Teszteljük hogy lefagy-e vagy nem. Mivel önmagát kapta meg és azt tudjuk hogy ez a program nem fagy le ezért indít egy végtelen ciklust, tehát lefagy. Ez viszont ellentmondás. Így tehát lehetetlen olyan programot készíteni ami eldönti minden kapott programról hogy lefagy-e vagy nem.

2.3. Változók értékének felcserélése

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

Megoldás forrása:https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/csere2.c

A feladat az volt hogy készítsünk egy olyan programot ami két változó értékét felcseréli bármilyen logikai utasítás és segédváltozó nélkül. Egy ilyen kódot úgy lehet elkészíteni ha matematikai módszereket alkalmazva cseréljük ki a két számot. Az alább látható kód ezt mutatja be. Bekérünk a felhasználótól két számot majd összeadás és kivonás segítségével felcseréljük a két számot. Ahhoz viszont hogy a standart outputra tudjunk írni és onnan olvasni szükség van az **stdio.h** függvénykönyvtárra.

```
Változó csere matematikai műveletekkel
{

#include <stdio.h>

int main()
```

```
{
  int a = 0;
  int b = 0;
  printf("Adja meg az a szamot: ");
  scanf("%d" , &a );
  printf("Adja meg a b szamot: ");
  scanf("%d" , &b);
  b = b-a;
  a = a+b;
  b = a-b;
  printf("a=%d%s",a,"\n");
  printf("b=%d%s",b,"\n");

}
}
```

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!

Megoldás forrása:https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Turing/Labda

Az alábbi program a standart outputon imitálja egy labda pattogását. Ehhez szüksége van több függvénykönyvtárra. A lentebb látott kódban láthatóak. Ezek közzül már van amelyiket néztük. Most az új a **curses.h**. Erre azért van szükség hogy a kódban látható WINDOW, initscr, getmaxyx és refresh használható legyen. A futtatás során pedig szükség lesz a **-lncurses** kapcsolóra. A részletesebb kód magyarázat pedig a kommentekben olvasható.

```
Labdapattogás if-el
{
#include <stdio.h>
#include <curses.h>
#include <unistd.h>

//gcc labdaif.c -o labda -lncurses

int
main ( void )
{
    WINDOW *ablak;    //létrehozza az ablakot
    ablak = initscr ();   //kiszámolja az ablak méretét

    int x = 0;
    int y = 0;

    int xnov = 1;
    int ynov = 1;
```

```
    int mx;
    int my;

    for ( ;; ) {

        getmaxyx ( ablak, my , mx );    //átadja az my-nak az ablak magasságát, mx-nek a szélességét

        mvprintw ( y, x, "O" );   //kiírja a labdát

        refresh ();     //meg kell hívni hogy kimenetet kapjunk a terminálra
        usleep ( 100000 );    //lassítja a labda mozgását micro second-be mérve
    clear();      //letisztítja az ablakot

        x = x + xnov;   // labda vízszintes koordinátája
        y = y + ynov;   // labda függőleges koordinátája

        if ( x>=mx-1 ) { // elerte-e a jobb oldalt?
            xnov = xnov * -1;
        }
        if ( x<=0 ) { // elerte-e a bal oldalt?
            xnov = xnov * -1;
        }
        if ( y<=0 ) { // elerte-e a tetejet?
            ynov = ynov * -1;
        }
        if ( y>=my-1 ) { // elerte-e a aljat?
            ynov = ynov * -1;
        }

    }

    return 0;
}

}
```

Nézzük az if nélüli verziót. Az **stdlib.h** könyvtárra szükség lesz az abs függvény használatához. A **define** sorokban határozzuk meg az ablak méretét melyben a labda pattogni fog. A putX egy függvény melyet mi készítettünk és a mainbe kerül majd meghívásra. A koordinátákat kell neki megadni és azok segítségével kiírja a labdát. A mainbe mikor meghívásra kerül akkor használjuk benne az abs függvényt. Erre azért van szükség hogy a távolságok ne lépjenek ki a képernyőről. A többi magyarázatot lásd a kód kommentelésében.

```
Labdapattogás if nélkül
{

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

```
#include<unistd.h>

#define SZEL 78   //meghatározza az ablak szélességét
#define MAG 22    //meghatározza az ablak magasságát

int putX(int x,int y)
{
int i;

for(i=0;i<x;i++)    // meghatározza a labda függőleges koordinátáját az ablakon belül
printf("\n");

for(i=0;i<y;i++)    // meghatározza a labda vízszintes koordinátáját az ablakon belül
printf(" ");

printf("X\n");    //kiírja a labdát és tesz egy új sort

return 0;
}

int main()
{
int x=0,y=0;    //a kezdeti koordinátákat beállítja nullára

while(1)    //végtelen ciklus indul
{
system("clear");    // az ablakot tisztítja
putX(abs(MAG-(x++%(MAG*2))),abs(SZEL-(y++%(SZEL*2))));    //kiszámolja a labda aktuális helyét és kiírja a labdát
usleep(50000);    //meghatározza a labda sebességét
}

return 0;
}
}
```

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 forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Turing/Szohossz_BogoMIPS

A program végig shifteli bitenként az **integer** változót és a végén kiírja hogy hány biten tárolja. Ehhez szükség van a **bitshift** operátorra. Ez egyessével balra shifteli az intet amíg az utolsó bit is nem lesz 0. Ekkor végeredményként kiírja hogy 32 biten tárolja.

```
Szóhossz
{
#include <stdio.h>
int main()
{
  int a=1;
  int n=1;
  while(a<<=1)
  {
    n+=1;
  }
  printf("Megoldas:%d%s",n,"\n");
}

}
```

2.6. Helló, Google!

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/pagerank.c

Amikor a Google megalakult a page-rank-ot használták fel arra hogy az oldalakat rangsorolják. Ez alapján állították fel a sorrendet az eredmények megjelenítésekor. A jobb értékkel rendelkező oldal feljebb került a listában így a relevánsabb adatok előrébb kerültek. A page-rank ugyanis azt csinálja hogy az egyes honlapokat állítja sorba az alapján hogy mennyire releváns a rajtuk szereplő információ. A rangsoroláshoz minden honlapnak kiszámít egy értéket és ezeket hasonlítja össze. Az egyes értékeket pedig úgy számolja ki hogy veszi az adott oldalra mutató linkek számát. Valamint összeadja azt hogy az oldalra mutató honlapokból hány link indul ki és ezeket összeadja. Végül a két értéket elosztja egymással. Az ehhez kapcsolódó kód lentebb látható a magyarázattal egybe.

```
Page-Rank
{
#include <stdio.h>
#include <math.h>

void kiir (double tomb[], int db)   //kiírja az eredmény tömböt, a db felel ↩
    azért hogy a ciklus hányszor fut le a függvényen belül
{
  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++)
  {
    if ((pagerank[i] - pagerank_temp[i])<0)
    {
      tav +=(-1*(pagerank[i] - pagerank_temp[i]));
    }
    else
    {
      tav +=(pagerank[i] - pagerank_temp[i]);
    }
  }
  return tav;
}

int main(void)
{
  double L[4][4] = {              //ebben tároljuk a linkmátrixot
  {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};    // ebben lesz benne a végeredmény
  double PRv[4] = {1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0, 1.0 / 4.0};   //az ↩
     egyes oldalak preztizse

  long int i,j;   // ciklus számláló
  i=0; j=0;

  for (;;)
  {
    for(i=0;i<4;i++)    //átmásolja a PR-be a PRv-t
      PR[i] = PRv[i];
    for (i=0;i<4;i++) //az L linkmátrixot összeszorozza a PR vektorral
    {
      double temp=0;
      for (j=0;j<4;j++)
        temp+=L[i][j]*PR[j];
      PRv[i]=temp;        //a mátrix szorzás eredményét eltárolja a PRv-ben
    }

    if ( tavolsag(PR,PRv, 4) < 0.00001)   //a feltétel ha teljesül akkor ↩
       belép a az if magjába
      break;                //ebben az esetben lép ki a végtelen ciklusból
  }
  kiir (PR,4);    //meghívja a kiir függvényt és kiírja az eredményt
  return 0;

}
```

```
}
```

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/brun.r

A feladat megoldásához meg kell értenünk mik is azok a prímszámok és az ikerprímek. A prímszámok olyan számok amiknek csak kát osztójuk van, az 1 és önmaga. Az ikerprímek azok is prímszámok, különlegességük hogy a két szám közötti távolság kettő. Ilyen számpárokról a mai napig nem tudni hogy végtelen sok van-e. A Brun tétel pedig ezekkel a számpárokkal foglalkozik. A tétel azt mondja ki hogy az ikerprímek reciprokának összege a Brun konstanshoz konvergál. Az alábbi kód ennek bemutatására készült.

```
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")
```

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/blob/master/Hello_Turing/monty.r

A Monty Hall egy valószínűségi problémát vizsgál. Az elv az hogy három lehetősg közzül kell választanunk úgy hogy egy jó választás van, a másik két esetbe rosszabbul járunk. Ha választottunk egyet utána a másik kettőből egyet megnézünk és kiderül hogy az egy rossz választás. Utána pdeig meg van a lehetőség arra hogy újra válasszunk. A kérdés az hogy megéri a két fent maradó lehetőségből a másikat választani vagy inkább maradjunk az eredeti választásnál. Az elvi megoldás az hogy megéri váltani mivel akkor nagyobb valószínűséggel választjuk a jó megoldást. Ez viszont ellent mond az emberi gondolkodásnak ezért ez paradoxonhoz vezet. A kód pedig erre a problémára ad egy szemléltetést.

```
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)
```

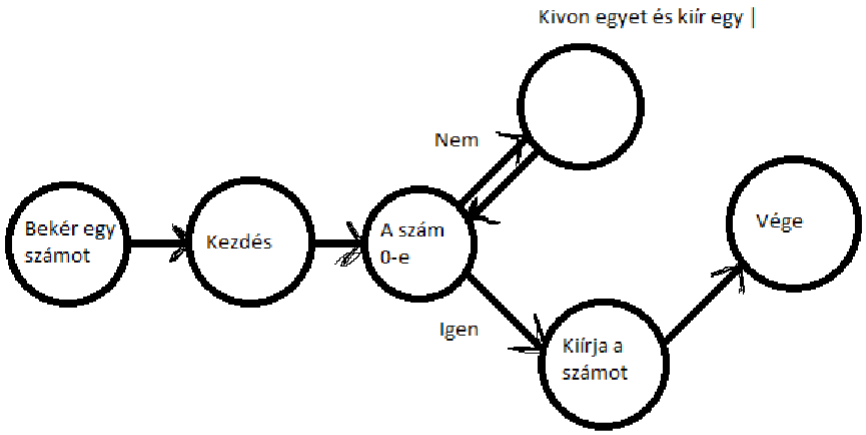
3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

A decimálisból unárisba váltó Turing gép tizes számrendszerből vált egyesbe. Az unáris számrendszer csak arra alkalmas hogy természetes számokat ábrázoljon mivel csak egy karakter létezik benne. Mondjuk egy | karakter. Ezzel pedig nem tudunk mást leírni csak "egyszerű"(természetes) számokat. Tehát mondjuk a 6 ábrázolása a következő képpen néz ki:||||||.



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 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.

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/Hivatkozasi_nyelv

A feladat az volt hogy mutassuk meg mi a különbség a C89 és C99 verzió között. Ezek a C programozási nyelv két különböző változata. A C99-es egy újabb verzió a 89-hez képest. Ebből adódóan találhatóak benne újítások melyek még a C89-ben nem szerepelnek. Ilyen például az hogy a 89-es verzióban még volt lehetséges az hogy a for ciklus fejében deklaráljuk a futó változót. Ebben a verzióban még a ciklus előtt

deklarálni kellett és a ciklus fejben csak értéket adtunk neki. A 99-es verzióban viszont már lehetséges volt a fejben deklarálni a futó változót. A futtatás során pedig a **-std=c89** és **-std=c99** kapcsolóval lehet kényszeríteni a fordítót hogy ezeket a verziókat használja.

```
C89
{

#include <stdio.h>

//gcc -std=c89 c89.c -o c89

int main()
{
  int i;
  for (i = 0; i < 5; i++)
  {
    printf("%d ",i+1);
  }
  printf("\n");
  return 0;
}
}
```

```
C99
{

#include <stdio.h>

//gcc -std=c99 c99.c -o c99

int main()
{
  for (int i = 0; i < 5; i++)
  {
    printf("%d ",i+1);
  }
  printf("\n");
  return 0;
}
}
```

3.4. Saját lexikális elemző

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/Lexik%C3%A1lis_eler

A program bekér egy szöveget a felhasználótól. Ezen lefuttatja a lexert és a szövegben található összes számot "kiemeli" és átalakítja számmá(tehát számként fogja kezelni nem pedig szövegként). A kilépéskor - melyhez a (ctrl + d) kombinációt használjuk - pedig kiírja hogy hányszám szerepel a szövegbe. Ennek a feladatnak az a lényege hogy ne a saját kódunkat használjuk hanem a lexert használjuk tehát óriások vállán álljunk.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/l33t

A leet egy olyan program amely arra való hogy a beírt szavakban egyes karaktereket kicserél más karakterekre de úgy hogy a szöveg továbbra is olvasható marad. Ennek az online beszélgetésekben van nagy haszna. Hiszen ha egy olyan szót akarunk beírni ami az adott oldal szűrőin nem megy át akkor egy kis módosítással ez megkerülhető. Például ha egy szóban szerepel a betű akkor azt ha kicseréljük mondjuk @ jelre akkor az még érthető ám a szűrő már átengedi és nem fogja cenzúrázni.

A program kódja megadja hogy milyen karaktereket mikre lehet cserélni. Egy tömbben tároljuk az erre vonatkozó információkat. Valamint azt is eltároljuk hogy a beírt szöveg milyen hosszú és ezután végig megyünk a teljes szövegen és átírjuk benne a karaktereket if-ek segítségével.

```
%{
  #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;
}
```

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 fenti kódhoz képest itt az egyenlőség vizsgálata le lett cserélve a nem egyenlőre. Ez a kicsi változás pedig elég ahhoz hogy az eredmény az ellentéte legyen az eredetinek.

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

Itt egy for ciklus fog ötször lefutni. A ++i-nek köszönhetően az i értéke előbb fog nőni és csak utána fog kiértékelődni.

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

Itt egy for ciklus fog ötször lefutni. A i++-nak köszönhetően az i értéke előbb fog kiérnőni és csak utána fog nőni.

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

Ebben a példában is egy for ciklus fog ötször lefutni. Viszont itt már egy tömbben el is tároljuk az i++ értékeit. Első helyen egy memória szemét lesz utána viszont az i++ értékei kerülnek tárolásra.

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

Ebben a kód részletben a for ciklus feltételében egy összekapcsolt feltétel van. Viszont ez a feltétel hibás. Hiszen (*d++ = *s++) ebben a részben értékadó egyenlőség van nem pedig összehasonlító. Így tehát ez nem ad vissza logikai értéket ezért nem lehet eldönteni hogy a feltétel teljesül vagy nem.

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

Az f() függvény értékét íratjuk ki. Ám mivel ebben a kód részletben nem látjuk magát a függvényt azt viszont igen hogy a megadott értékek felcserélet˝ok így arra lehet következtetni hogy az f()-ben valamilyen kommutativ dolog van.

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

Itt a printf-el két számot íratunk ki decimális formában. Az egyik az f() által visszaadott érték a másik meg az a paraméter amit a függvénynek átadtunk.

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

Két szám kiiratása történik. Egy memóriacím általi érték és az a változó.

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Hello_Chomsky/forras_olvasas

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

Az Ar egy logikában használatos nyelv mely a matematikai nyelvet foglalja magába.

\text = szöveg kiírása

\forall = univerzális kvantor

\exists = egzisztenciális kvantor

\wedge = konjukció

\vee = diszjunkció

\neg = negáció

\supset = implikáció

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

  Minden x-re létezik olyan y hogy x kisebb mint y és y prím.

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

  Minden x-re létezik olyan y hogy x kisebb mint y és y prím és y kettőre  ↩
     következője is prím.

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

  Létezik y esetén minden x, ha x prím akkor x kisebb mint y.

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

  Létezik y esetén minden x, ha y kisebb mint x akkor x nem prím.
```

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

Mit vezetnek be a programba a következő nevek?

- `int a;`

  egész
- `int *b = &a;`

  egy mutató ami egy memóriacímre mutat
- `int &r = a;`

  egész referenciája
- `int c[5];`

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

  egészek tömbjének referenciája (nem az első elemé)
- `int *d[5];`

  egészre mutató mutatók tömbje

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

  egészre mutató mutatót visszaadó függvény
- ```
  int *(*l) ();
  ```

  egészre mutató mutatót visszaadó függvényre mutató mutató
- ```
  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

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/H%C3%A1romsz%C3%B6gm%

Ebben a programban alsó háromszögmátrixot hozunk létre. Hogy ezt létre tudjuk hozni először is tisztáznunk kell hogy mi az az alsó háromszögmátrix. Ehhez pedig tudnunk kell hogy mi az a mátrix. A mátrix egy olyan vektor melynek két dimenziója van. Tehát rendelkezik sorokkal és oszlopokkal is. Az alsó háromszögmátrix pedig egy olyan speciális fajtája ennek amelyben a főátló feletti összes érték 0, valamint a mátrix négyzetes.

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

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

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

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

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

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

```
            return -1;
        }

    }

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

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

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

    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
    *(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;
}
```

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/C_titkos%C3%ADt%C3%B3

Ez a fajta titkosító módszer nagyon régi és elég egyszerű elven alapul. Van egy titkosítandó szövegünk és egy kulcsunk amivel titkosítunk. A módszerlényege hogy a kulcsot - ami lehet egy akármilyen karakterlánc

- átváltjuk bitekre és a szöveget is. Ezután a szövegen és a kulcson végrehajtja a kizáró vagyot. Ez úgy működik hogy a szöveg alá helyezzük a kulcs bitjeit és ahol a bitek megegyeznek oda a titkosítás során 0-t írunk ahol különböznek oda pedig 1-est. A művelet végrehajtása után pedig egy titkosított szöveget kapunk ami értelmezhetetlen. Ahhoz hogy ismét olvashatóvá váljon a kulcs segítségével vissza kell fejteni az eredeti szöveget.

Az alábbi programban erre a titkosítóra látunk egy konkrét kódot. Úgy működik hogy a program megkapja a kulcsot és a titkosítandó szöveget. A szöveget elkezdi beolvasni a bufferbe és a kulcs segítségével elkezdi a titkosítást. Mikor végzett az adott résszel, akkor azt kiírja egy eredmény fájlba és beolvassa a szöveg következő részét a bufferbe. Ezt a folyamatot addig ismétli amíg a teljes szöveget nem titkosítja.

```
#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);

    }
}
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/Java_titkos%C3%ADt%C3%B3

Az előző feladathoz hasonlóan ez a program is az EXOR-os titkosítást végzi el. A különbség annyi hogy ez a kód java nyelven van írva és tartalmaz obijektum orientált részeket. Ilyen például a class. További különbség hogy ez a kód a titkosítandó szöveget nem egy fájlból olvassa be hanem a felhasználónak kell begépelni a standart inputon keresztül. A program további működési elve ugyanaz.

```
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();

        }

    }

}
```

4.4. C EXOR törő

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

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/C_t%C3%B6r%C5%91

Ebben a részben az előző két feladat ellentéte szerepel. AMíg ott a titkosítás volt a cél addig ennél a feladatnál a feltörés a cél. A program megkapja a titkosított szöveget és elkezdi keresni a feloldáshoz szükséges kulcsot, amely 5 karakter hosszú és az angol abc betűit tartalmazza. A törés elve a brute force módszer. Egyesével kipróbál minden lehetséges kulcsot amíg meg nem találja a megfelelőt. Amikor végzett a töréssel akkor pedig teszteli is az eredményt. A tiszta_lehet azt vizsgálja hogy a megadott kulcsszavak szerepelnek-e a tiszta szöbvegbe, ha igen akkor valószínűleg helyes a törés.

```
#define MAX_TITKOS 4096
#define OLVASAS_BUFFER 256
#define KULCS_MERET 5
#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;
  char kod[28]= {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o ↩
     ','p','q','r','s','t','u','v','w','x','y','z'};
```

```
  // 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 <= 28; ++ii)
    for (int ji = 0; ji <= 28; ++ji)
      for (int ki = 0; ki <= 28; ++ki)
    for (int li = 0; li <= 28; ++li)
        for (int mi = 0; mi <= 28; ++mi)
      {
        kulcs[0] = kod[ii];
        kulcs[1] = kod[ji];
        kulcs[2] = kod[ki];
        kulcs[3] = kod[li];
        kulcs[4] = kod[mi];

        if (exor_tores (kulcs, KULCS_MERET, titkos, p - titkos))
      {
          printf
      ("Kulcs: [%c%c%c%c%c]\nTiszta szoveg: [%s]\n",
       kod[ii], kod[ji], kod[ki], kod[li], kod[mi], titkos);
        return 0;
      }

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

  return 0;
}
```

4.5. Neurális OR, AND és EXOR kapu

R

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/Neur%C3%A1lis

Tanulságok, tapasztalatok, magyarázat...

```
library(neuralnet)

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

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

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

plot(nn.or)

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


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

orand.data <- data.frame(a1, a2, OR, AND)

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

plot(nn.orand)

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



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

exor.data <- data.frame(a1, a2, EXOR)

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

plot(nn.exor)

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

```
a1      <- c(0,1,0,1)
a2      <- c(0,0,1,1)
EXOR    <- c(0,1,1,0)

exor.data <- data.frame(a1, a2, EXOR)

nn.exor <- neuralnet(EXOR~a1+a2, exor.data, hidden=c(6, 4, 6), linear. ↩
   output=FALSE, stepmax = 1e+07, threshold = 0.000001)

plot(nn.exor)

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

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása: https://github.com/nagyrobert1996/Prog1/tree/master/Caesar/Perceptron

Ennél a feladatnál egy kép segítségével a bináris osztályozást fogjuk "megtanítani" a számítógépnek. A bináris osztályozás jól bemutatható 1-kel és 0-kal. Az egyik osztályba az 1-ek a másik osztályba a 0-k fognak tartozni.

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.