Univerzális programozás

Írd meg a saját programozás tankönyvedet!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert Ã©s Hetes, Bence	2019. március 11.	

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/RakeTheRape/BHAKSZ/blob/master/turing/

Tanulságok, tapasztalatok, magyarázat...

A program ciklusokat használva dolgoztatja meg a procit. Sokszor véletlenül készítünk végtelen ciklusokat. Az alábbi példa azt az esetet mutatja be ahol csak 1 magot használ 100%-on.

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

A következő példa már minden magot használ 100%-on.

```
#include <omp.h>

int main()
{
#pragma omp parallel
  {
  for(;;);
  }

}
```

Ennek a programnak a futtatásához szükséges a "-fopenmp" használata.

Utolsó példánk mikor a program 0%-ban használja ki a procit. Ezt sleep parancssal érhetjük el a legegyszerűbben.

```
#include <unistd.h>

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

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

A T100-as programot felhasználva készítsük most el az alábbi T1000-set, amelyben a Lefagy-ra épőlő Lefagy2 már nem tartalmaz feltételezett, csak csak konkrét kódot:

```
Program T1000
{

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

  boolean Lefagy2(Program P)
  {
     if(Lefagy(P))
      return true;
     else
      for(;;);
  }

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

Mit for kiírni erre a `T1000(T1000)` futtatásra?

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true
- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

akkor most hogy fog működni? Sehogy, mert ilyen `Lefagy` függvényt, azaz a T100 program nem is létezik.

Tanulságok, tapasztalatok, magyarázat...

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: https://github.com/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

Elsőnek is segédváltozóval oldjuk meg a problémát. Azaz létrehozunk még egy változót amiben tárolhatjuk az értéket.

```
#include <stdio.h>

int main()
{
  printf("segédváltozó csere\n\n");
int a=5;
int b=7;

  printf("eredeti: a=%d b=%d\n",a,b);
int c=a;
a=b;
b=c;

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

  return 0;
}
```

Ezután segédváltozó nélkül, szimpla összeadás kivonással oldjuk meg.

```
#include <stdio.h>

int main()
{
  printf("valtozocsere valtozo nelkul\n\n");
int a=5;
int b=7;
  printf("eredeti: a=%d b=%d\n",a,b);
a=a+b;
b=a-b;
a=a-b;
  printf("felcserelt: a=%d b=%d\n",a,b);
  return 0;
}
```

Végül exor-módszerrel is megoldjuk.

```
#include <stdio.h>

int main()
{
  printf("Csere exorral\n\n");
  int a=5;
  int b=7;
  printf("Eredeti értékek: a=%d b=%d\n",a,b);
  a=a^b;
  b=a^b;
  a=a^b;
```

```
    printf("Felcserélt értékek: a=%d b=%d\n", a,b);
    return 0;
}
```

2.4. Labdapattogás

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

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

Megoldás forrása:https://github.com/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

Labdapattogtatást egyszerű feltételvizsgálatokkal oldhatjuk meg, felvesszük a terminál méretét (initscr() function-nal), a labda kezdő helyét, gyorsaságát. Miután megadtuk a kezdő adatokat, "if" függvényekkel vizsgáljuk, hogy a labda az ablak széléhez ért-e, és irányt változtatunk. A labda gyorsaságának csökkentéséhez használhatjuk a usleep() függvényt, kép frissitéshez pedig a refresh()-t.

```
#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 );
  clear();

  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;
}
```

Ha nagyon kalandvágyóak vagyunk if nélkül is elérhetjük ezt.

```
#include <iostream>
#include <iomanip>
#include <unistd.h>

using namespace std;

int rajz(int x,int y, int h, int w)
{
    for(int i=1;i<=x;i++)
    {
  cout<<"X";
  for(int j=1;j<=w;j++)
  {
    cout<<" ";
  }
  cout<<" X"<<endl;
    }
    cout<<"X";
    for(int i=0;i<y;i++)
    {
  cout<<" ";
    }
    cout<<"o";
    for(int i=y;i<w;i++)
    {
  cout<<" ";
```

```
    }
    cout<<"X"<<endl;
    for(int i=x;i<h;i++)
    {
  cout<<"X";
  for(int j=1;j<=w;j++)
  {
    cout<<" ";
  }
  cout<<" X"<<endl;
    }

    return 0;
}

int main()
{
    int x=0,y=0;
    int width,height;

    cout<<"Add meg a terulet szelesseget es magassagat, amin szeretned, ↩
        hogy pattogjon a labda! \n";
    cin>>height>>width;
    while(true)
    {
  system("clear");
  cout<<" ";
  for(int i=0;i<width+1;i++)
  {
    cout<<"X";
  }
  cout<<endl;
  rajz(abs(height-(x++%(height*2))),abs(width-(y++%(width*2))),height, ↩
     width);
  cout<<" ";
  for(int i=0;i<width+1;i++)
  {
    cout<<"X";
  }
  cout<<endl;
  usleep(50000);
    }
    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 videó:

Megoldás forrása: https://github.com/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

Az 'int' méretét, és hogy hány bitet foglal, bitshifteléssel nézzük meg. A 'while' függvényben folyamatosan balra lépegetünk, és minden lépésnél növeljük a 'hossz' változót, amig a 'wat' változó nulláig nem ér.

```
#include <stdio.h>
int
main (void)
{
  int hossz = 0;
  int wat = 1;
  do
    ++hossz;
  while (wat <<= 1);
  printf ("szohossz: %d bites\n", hossz);
  return 0;
}
```

Az 'int' 32 bites szó.

BogoMips

A BogoMips egy algoritmus ami a Linux kernelben méri fel a processzor sebességét az ú.n. 'busy-loop' konfigurálásához. A 'busy-loop' annyit jelent, hogy egy process folytonosan egy feltételt vizsgál, amíg az igazat ad vissza értékül, BogoMips azt mutatja hány mp-ig áll a proci, tehát nem csinál semmit.

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

void
delay (unsigned long long int loops)
{
unsigned long long int i;
for (i = 0; i < loops; i++);
}

int
main (void)
{
unsigned long long int loops_per_sec = 1;
unsigned long long int ticks;

printf ("Calibrating delay loop..");
fflush (stdout);
```

```
while ((loops_per_sec <<= 1))
{
ticks = clock ();
delay (loops_per_sec);
ticks = clock () - ticks;

printf ("%llu %llu\n", ticks, loops_per_sec);

if (ticks >= CLOCKS_PER_SEC)
{
loops_per_sec = (loops_per_sec / ticks) * CLOCKS_PER_SEC;

printf ("ok - %llu.%02llu BogoMIPS\n", loops_per_sec / 500000,
(loops_per_sec / 5000) % 100);

return 0;
}
}

printf ("failed\n");
return -1;
}
```

Bitshifteléssel megyünk a while ciklusban végig a 2 hatványain. A ticks-ben tároljuk mennyi processzoridőt használt a CPU eddig, majd a delay() függvénynek átadjuk loops_per_sec változót (aminek a bitjei mindig odébb vannak eggyel tolva), ahol elszámolunk 0-tól a változó végéig. Ezután megint lekérjük a processzoridőt kivonva az előző ticks-ben tárolt CPU időt, így megkapjuk, mennyi ideig tartott a elszámolni a loops_per_sec változó végéig. Majd megnézzük if-el, hogy nagyobb vagy egyenlő a kapott ticks, mint a CLOCKS_PER_SEC aminek az értéke 1 millió és ha ez igaz, akkor kiszámoljuk, hogy milyen érték kell ahhoz,hogy a ciklusértékéket megkapjuk, ezzel meghatározva a CPU sebességét.

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/RakeTheRape/BHAKSZ/tree/master/turing

Tanulságok, tapasztalatok, magyarázat...

A Page Rank az interneten található oldalakat rangsorolja. Kezdetben minden oldalnak van egy szavazati pontja és ha az egyik linkeli a másikat, akkor a linkelt oldal megkapja a linkelő pontját. Tehát egy oldal akkor lesz előkelőbb helyen egy google kereséskor, ha minél több másik oldal linkel rá, illetve ezen oldalakra is minél többen linkelnek, annál jobb minőségűnek fog számítani egy linkelése vagy szavazata.

```
#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;
    }
```

Az L nevű többdimenziós tömbben vannak rögzítve mátrix formájában az adatok a linkelésekről, melyik oldalra melyik oldal linkel és mennyit. A végtelen ciklusban nullázzuk PR összes elemét, majd rögtön hozzáadjuk az L mátrix és PRv vektor szorzatainak értékét. Ezután a távolság metódusunkban végigmegyünk a PR és PRv vektorokon és egy változóban eltároljuk ezek különbségének a négyzetét (hogy ne legyen negatív) és gyököt vonva visszaadjuk az értéket, amely ha kisebb mint 0.00000001, akkor kilépünk a végtelen ciklusból, ellenkező esetben pedig PRv tömböt feltöltjük PR elemeivel. Végül kiiratjuk az értékeket.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás videó: https://youtu.be/xbYhp9G6VqQ

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

A Brun tétel azt mondja, hogy az ikerprímszámok reciprokából képzett összege konvergál egy számhoz. Ezt határt Brun konstansnak nevezzük. Ezzel ellentétben a prímszámok a végtelen felé tartanak.

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

A Monty Hall probléma egy amerikai televíziós vetélekedőben jelent meg, ahol a műsor végén a játékosnak 3 ajtó közül kellett választania. A nyeremény csak az egyik ajtó mögött volt. A játékos választása után a műsorvezető kinyitott egy üres ajtót és feltette a kérdést, hogy fenntartja-e a választását a játékos vagy egy másik ajtót választ. A Monty Hall probléma kérdése, hogy számít-e, hogy a játékos megváltoztatja-e a választását. Józan ésszel gondolkodva nem számít, mivel a maradék két ajtó közül az egyik mögött van a nyeremény, így 50-50% az esélye annak, hogy nyerünk. A feladvány bizonyítása több matematikai professzoron is kifogott, köztük a világhírű Erdős Pálon is, akit csak a számítógépes szimuláció győzött meg, ami alapján számít, hogy másik ajtót választunk, ugyanis ekkor megduplázódik az esélyünk a nyerésre. Amikor először választunk ajtót, akkor 1/3 az esélye annak, hogy eltaláljuk a nyertes ajtót és 2/3, hogy nem. Ezután a játékvezető kinyit egy ajtót, amelyik üres és ha nem változtatunk a döntésünkön, továbbra is 1/3 lesz annak az esélye, hogy nyertünk. Viszont mivel már csak 2 ajtó van a játékban ezért ha változtatunk, akkor 2/3 lesz az esélyünk a nyerésre.

```
kiserletek_szama=1000000
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)
```

Először eltároljuk, hogy hány kisérletet végezzünk el, majd a kiserlet és jatekos vektorokban kisorsolunk 1 és 3 között véletlenszerűen számokat. A műsorvezető vektorát beállítjuk a kísérletek számával. Ezután egy for ciklussal végigmegyünk minden kísérleten és ha kiserlet i-edig értéke megyegyezik a a játékos i-edig találatával, az jelenti, hogy eltalálta a játékos a nyereményt és a mibol vektorba az a két érték kerül be amelyeket a játékos nem választott. (Ez a két érték az üres ajtókat jelenti ebben az esetben.)

Ha a játékos nem találta el elsőre a kiserlet vektorban található számot, akkor a mibol vektorba már csak 1 egy érték kerülhet, az amelyik nem a nyeremény és nem a játékos által kiválasztott érték. Ezután a musorvezeto vektorba berakjuk a mibol vektorban található számot, illetve ha két érték van benne akkor a kettőből az egyiket véletlenszerűen.

Ezután értkezik a kiértékelés. A nemvaltoztatesnyer vektorba kerülnek azok az esetek, amikor elsőre eltalálja a játékos a megfelelő ajtót. Megint végigmegyünk a kísérleteken és a holvalt vektorba azok vagy az az érték kerül az 1, 2 és 3 közül amely nem egyenlő a műsorvezető és a játékos által választottal vagyis ekkor ha váltana a játékos akkor nyerne. A változtat vektorba pedig a holvalt vektor elemei közül az egyiket rakjuk át.

A valtoztatesnyer vektorba pedig azok az értékek kerülnek, amelyek a kiserlet vektorba és a változtat vektor-

ba találhatóak, vagyis ekkor az az ajtó a nyertes ,amelyiket másodjára választanánk. Eztuán pedig kiiratjuk az esetek számait:

```
[1] "Kiserletek szama: 1000000"
> length(nemvaltoztatesnyer)
[1] 333590
> length(valtoztatesnyer)
[1] 666410
> length(nemvaltoztatesnyer)/length(valtoztatesnyer)
[1] 0.5005777
> length(nemvaltoztatesnyer)+length(valtoztatesnyer)
[1] 1000000
```

3. fejezet

Helló, Chomsky!

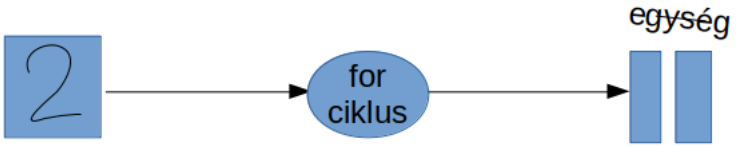
3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...



Az unáris számrendszer, azaz egyes számrendszer, lényege, hogy a számot egységekkel jelezzül, pl a 2-őt két db vonallal, és igy tovább. Az alábbi program bekér egy számot, majd a 'for' ciklussal annyiszor írjuk ki az egységet (jelen esetben 'I').

```
int main ()
{
  int x;
  printf ("Adjon meg egy szamot:");
  scanf ("%d", &x);
  for (int i = 0; i < x; i++)
    {
      printf ("I");
    }
  return 0;
}
```

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.

Megoldás videó:

Megoldás forrása:

```
        #include <stdio.h>
int main ()
{
  // Printing to screen.
  printf ("Hello World\n");
}
```

C99-et használva lefordul. :)

```
batf41@batf41-VirtualBox:~/Asztal$ gcc -o c89 -std=c99 c89.c
batf41@batf41-VirtualBox:~/Asztal$
```

C89-el már nem. :(

```
batf41@batf41-VirtualBox:~/Asztal$ gcc -o c89 -std=c89 c89.c
c89.c: In function ‘main’:
c89.c:4:7: error: C++ style comments are not allowed in ISO C90
      // Printing to screen.
       ^
c89.c:4:7: error: (this will be reported only once per input file)
batf41@batf41-VirtualBox:~/Asztal$
```

3.4. Saját lexikális elemző

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

Elsőnek is készítünk egy változót, amelyben majd taroluk a számok mennyiségét. Ezt majd növeljük minden szám beírása után. Majd a számjegyeket 0-9-ig definiáljuk 'digit' név alatt.

```
%{
#include <stdio.h>
int realnumbers = 0;
%}
digit [0-9]
```

A következő részben megadjuk a programnak a mintát amit keresnie kell.

```
%%
{digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
```

A '{digit}*' rész azt jelenti hogy bármennyi digit, tehát szám lehet.

```
{digit}*(\.{digit}+)?
```

A '\.' részen a pont önmagában annyit tesz hogy bármilyen karakterre rá lehet illeszteni. Viszont jelen esetben nekünk a '\' jellel le kell "védenünk". Így a program a valós számoknál lévő pontot fogja értelmezni.

A '\.' után jönnek a számjegyek, a '+' pedig annyit jelent, hogy legalább egy digitnek lennie kell a pont után.

```
(\.{digit}+)?
```

Ezután kiírjuk a számot amit a program felismert, 'atof'-al pedig a double tipusba konvertált változatát is.

```
{++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
```

Végül lehívjuk a lexert, és szépen elvégez nekünk minden maradékot.

```
int
main ()
{
 yylex ();
 printf("The number of real numbers is %d\n", realnumbers);
 return 0;
}
```

A teljes kód így néz ki:

```
%{
#include <stdio.h>
int realnumbers = 0;
%}
```

```
digit [0-9]
%%
{digit}*(\.{digit}+)? {++realnumbers;
    printf("[realnum=%s %f]", yytext, atof(yytext));}
%%
int
main ()
{
 yylex ();
 printf("The number of real numbers is %d\n", realnumbers);
 return 0;
}
```

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

3.6. A források olvasása

Hogyan olvasod, hogyan értelmezed természetes nyelven az alábbi kódcsipeteket? Például

```
if(signal(SIGINT, jelkezelo)==SIG_IGN)
    signal(SIGINT, SIG_IGN);
```

Ha a SIGINT jel kezelése figyelmen kívül volt hagyva, akkor ezen túl is legyen figyelmen kívül hagyva, ha nem volt figyelmen kívül hagyva, akkor a jelkezelo függvény kezelje. (Miután a **man 7 signal** lapon megismertem a SIGINT jelet, a **man 2 signal** lapon pedig a használt rendszerhívást.)

Bugok
Vigyázz, sok csipet kerülendő, mert bugokat visz a kódba! Melyek ezek és miért? Ha nem megy ránézésre, elkapja valamelyiket esetleg a splint vagy a frama?

---

i.
```
if(signal(SIGINT, SIG_IGN)!=SIG_IGN)
    signal(SIGINT, jelkezelo);
```

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

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

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

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

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

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

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

Megoldás forrása:

Megoldás videó:

Tanulságok, tapasztalatok, magyarázat...

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

Megoldás videó: https://youtu.be/ZexiPy3ZxsA, https://youtu.be/AJSXOQFF_wk

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
- egészre mutató mutató
- egész referenciája

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

Mit vezetnek be a programba a következő nevek?

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4. fejezet

Helló, Caesar!

4.1. int *** háromszögmátrix

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat...

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

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.