Univerzális programozás

Így neveld a programozód!

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

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Ã©s Bátfai, Margaréta	2020. március 9.	

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	A Brun tételes feladat kidolgozása.	nbatfai

Ajánlás

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

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

Tartalomjegyzék

Ábrák jegyzéke

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

Ne cifrázzuk: programok írása. Mik akkor a programok? Mit jelent az írásuk?

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: http://esr.fsf.hu/hacker-howto.html!
- Olvasgasd aztán 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
- C kapcsán a [KERNIGHANRITCHIE] könyv adott részei.
- C++ kapcsán a [BMECPP] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a The GNU C Reference Manual, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf`
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [BMECPP] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

- 21 - Las Vegas ostroma, https://www.imdb.com/title/tt0478087/, benne a Monty Hall probléma bemutatása.
- Kódjátszma, https://www.imdb.com/title/tt2084970, benne a kódtörő feladat élménye.

- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a bemutatása.
- , , benne a 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

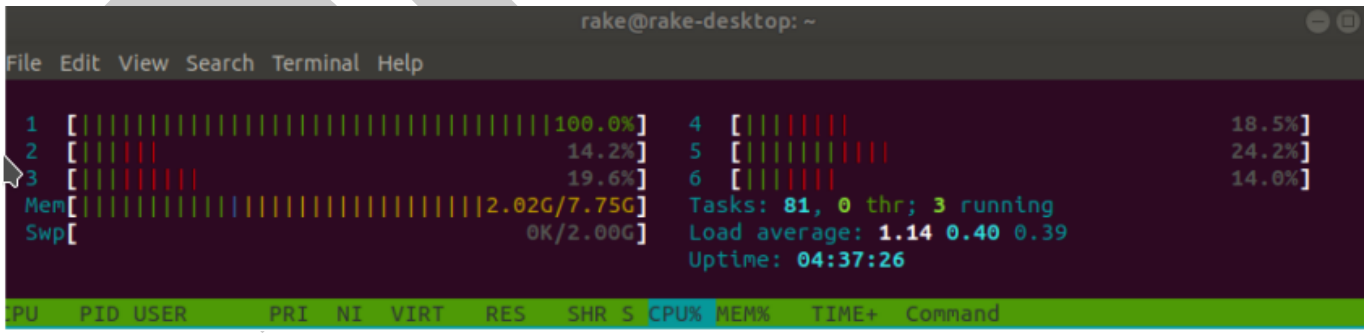
Írj olyan C végtelen ciklusokat, amelyek 0 illetve 100 százalékban dolgoztatnak egy magot és egy olyat, amely 100 százalékban minden magot!

Megoldás forrása: 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(;;);
}
```



2.1. ábra. 1 magot használ 100%-on

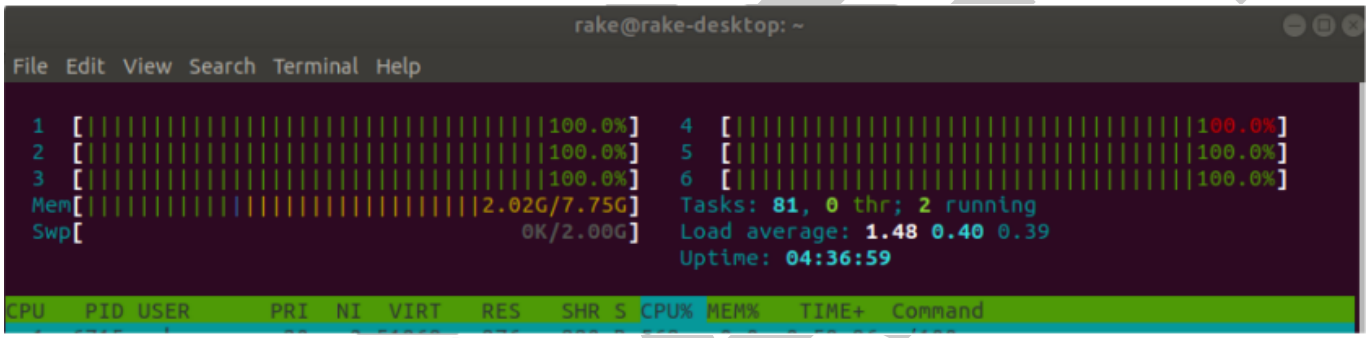
A következő példa már minden magot használ 100%-on.

```
#include <omp.h>

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

}
```

Ennek a programnak a futtatásához szükséges a "-fopenmp" használata, a multi threading miatt.

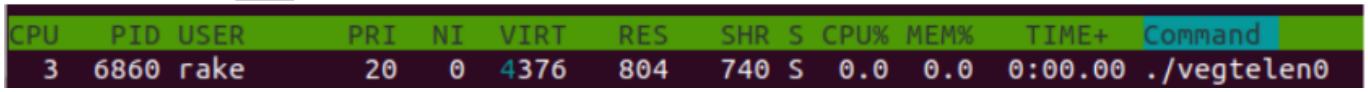


2.2. ábra. minden magot használ 100%-on

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);
}
```



2.3. ábra. 0%-ban használja ki a procit

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: /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 videó-

kon.)

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

Megoldás forrása:/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: /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. 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 vektorba 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
```

2.8. 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 természetes számok építőelemei a prímszámok. Abban az értelemben, hogy minden természetes szám előállítható prímszámok szorzataként. Például 12=2*2*3, vagy például 33=3*11.

Prímszám az a természetes szám, amely csak önmagával és eggyel osztható. Eukleidész görög matematikus már Krisztus előtt tudta, hogy végtelen sok prímszám van, de ma sem tudja senki, hogy végtelen sok ikerprím van-e. Két prím ikerprím, ha különbségük 2.

Két egymást követő páratlan prím között a legkisebb távolság a 2, a legnagyobb távolság viszont bármilyen nagy lehet! Ez utóbbit könnyű bebizonyítani. Legyen n egy tetszőlegesen nagy szám. Akkor szorozzuk össze n+1-ig a számokat, azaz számoljuk ki az 1*2*3*... *(n-1)*n*(n+1) szorzatot, aminek a neve (n+1) faktoriális, jele (n+1)!.

Majd vizsgáljuk meg az a sorozatot:

(n+1)!+2, (n+1)!+3,... , (n+1)!+n, (n+1)!+ (n+1) ez n db egymást követő azám, ezekre (a jól ismert bizonyítás szerint) rendre igaz, hogy

- (n+1)!+2=1*2*3*... *(n-1)*n*(n+1)+2, azaz 2*valamennyi+2, 2 többszöröse, így ami osztható kettővel
- (n+1)!+3=1*2*3*... *(n-1)*n*(n+1)+3, azaz 3*valamennyi+3, ami osztható hárommal
- ...
- (n+1)!+(n-1)=1*2*3*... *(n-1)*n*(n+1)+(n-1), azaz (n-1)*valamennyi+(n-1), ami osztható (n-1)-el
- (n+1)!+n=1*2*3*... *(n-1)*n*(n+1)+n, azaz n*valamennyi+n-, ami osztható n-el
- (n+1)!+(n+1)=1*2*3*... *(n-1)*n*(n+1)+(n-1), azaz (n+1)*valamennyi+(n+1), ami osztható (n+1)-el

tehát ebben a sorozatban egy prim nincs, akkor a (n+1)!+2-nél kisebb első prim és a (n+1)!+ (n+1)-nél nagyobb első prim között a távolság legalább n.

Az ikerprímszám sejtés azzal foglalkozik, amikor a prímek közötti távolság 2. Azt mondja, hogy az egymástól 2 távolságra lévő prímek végtelen sokan vannak.

A Brun tétel azt mondja, hogy az ikerprímszámok reciprokaiból képzett sor összege, azaz a (1/3+1/5)+ (1/5+1/7)+ (1/11+1/13)+... véges vagy végtelen sor konvergens, ami azt jelenti, hogy ezek a törtek összeadva egy határt adnak ki pontosan vagy azt át nem lépve növekednek, ami határ számot B_2 Brun konstansnak neveznek. Tehát ez nem dönti el a több ezer éve nyitott kérdést, hogy az ikerprímszámok halmaza végtelen-e? Hiszen ha véges sok van és ezek reciprokait összeadjuk, akkor ugyanúgy nem lépjük át a B_2 Brun konstans értékét, mintha végtelen sok lenne, de ezek már csak olyan csökkenő mértékben járulnának hozzá a végtelen sor összegéhez, hogy így sem lépnék át a Brun konstans értékét.

Ebben a példában egy olyan programot készítettünk, amely közelíteni próbálja a Brun konstans értékét. A repó bhax/attention_raising/Primek_R/stp.r mevű állománya kiszámolja az ikerprímeket, összegzi a reciprokaikat és vizualizálja a kapott részeredményt.

```
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program.  If not, see <http://www.gnu.org/licenses/>

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")
```

Soronként értelemezzük ezt a programot:

```
 primes = primes(13)
```

Kiszámolja a megadott számig a prímeket.

```
> primes=primes(13)
> primes
[1]  2  3  5  7 11 13
```

```
 diff = primes[2:length(primes)]-primes[1:length(primes)-1]
```

```
> diff = primes[2:length(primes)]-primes[1:length(primes)-1]
> diff
[1] 1 2 2 4 2
```

Az egymást követő prímek különbségét képzi, tehát 3-2, 5-3, 7-5, 11-7, 13-11.

```
idx = which(diff==2)
```

```
> idx = which(diff==2)
> idx
[1] 2 3 5
```

Megnézi a `diff`-ben, hogy melyiknél lett kettő az eredmény, mert azok az ikerprím párok, ahol ez igaz. Ez a `diff`-ben lévő 3-2, 5-3, 7-5, 11-7, 13-11 különbségek közül ez a 2., 3. és 5. indexűre teljesül.

```
t1primes = primes[idx]
```

Kivette a primes-ból a párok első tagját.

```
t2primes = primes[idx]+2
```

A párok második tagját az első tagok kettő hozzáadásával képezzük.

```
rt1plust2 = 1/t1primes+1/t2primes
```

Az 1/t1primes a t1primes 3,5,11 értékéből az alábbi reciprokokat képzi:

```
> 1/t1primes
[1] 0.33333333 0.20000000 0.09090909
```

Az 1/t2primes a t2primes 5,7,13 értékéből az alábbi reciprokokat képzi:

```
> 1/t2primes
[1] 0.20000000 0.14285714 0.07692308
```

Az 1/t1primes + 1/t2primes pedig ezeket a törteket rendre összeadja.

```
> 1/t1primes+1/t2primes
[1] 0.5333333 0.3428571 0.1678322
```

Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvénnyel.

```
sum(rt1plust2)
```

```
>   sum(rt1plust2)
[1] 1.044023
```

A következő ábra azt mutatja, hogy a szumma értéke, hogyan nő, egy határértékhez tart, a B_2 Brun konstanshoz. Ezt ezzel a csipettel rajzoltuk ki, ahol először a fenti számítást 13-ig végezzük, majd 10013, majd 20013-ig, egészen 990013-ig, azaz közel 1 millióig. Vegyük észre, hogy az ábra első köre, a 13 értékhez tartozó 1.044023.

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

A B_2 konstans közelítése

2.4. ábra. A B_2 konstans közelítése

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.

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

A program először a primeket számolja ki x-ig. Ezután az egymásmellett álló prímeket kivonja egymásból. Az 'idx' azt nézi, hogy ha a különbség kettő, akkor azok ikerprímek. 't1primes' változóba azokat a vizsgált számpároknak első elemét tároljuk melyeknek különbsége 3.

't2primes' változóba azt a vizsgált számpár második elemét mentjük, ahol a különbség 4. Ezután jön a matek része, a függvény reciprokokat hoz létre, azokat összeadja. Ezeket kiírjuk, és láthatjuk merre konvergál. A prímek felső határhoz konvergálnak.

Werkfilm

- https://youtu.be/VkMFrgBhN1g
- https://youtu.be/aF4YK6mBwf4

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

unary.cpp

Tanulságok, tapasztalatok, magyarázat...

Az unáris számrendszerben a (decimális) számok N darab 1-s számjeggyel vannak ábrázolva, ez az egyetlen számjegy, például ha a 2-st szeretnénk felírni az említett számrendszerben : 11, ha a 6-t: 111111. Az ábrán egy Turing Gép("végtelen memóriás", szalagot egy internált tábla segítségével feldolgozó számítógép) végzi 10-s számrendszerből az unárisba váltást.

Ahhoz, hogy a gép ezt végrehajtsa az szükséges, hogy ciklusokban addig vonjon ki számból 1-t, amíg az nulla nem lesz, és e művelet közben a tárba kell pakolnia a levont egyeseket.

A szám utolsó számjegyén kezdődik a művelet, ha ez 0 akkor 9-cel megyünk a "kék" állapotban, ha 9 akkor 8-cal megyünk, tehát mindig levonunk 1-t, addig megyünk amíg újra nem 0-hoz érünk(ha nem 0-val kezdődött a számjegy, akkor első 0-ig tart), annyi egyest tározunk el, amennyiszer a művelet végrehajtódott, ha 0-hoz értünk megismételjük ezt a többi számjeggyel is.

Ha írni akarnánk erre a feladatra (nyílván Turing gép megalkotására nem leszünk képesek tulajdonságai miatt) a következőképpen nézne ki:

```
#include <iostream>
#include <limits>

void convertDecimalToUnary(int val);


int main()
{
```

```
    int val, theOption;

    std::cout << "Please enter a value to convert to unary:\n";

    do
    {
        std::cin >> val;
        std::cin.clear();
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(),'\n');

    }while(std::cin.fail());

    convertDecimalToUnary(val);

    return 0;


}

void convertDecimalToUnary(int val)
{
    for(int i=0; i<val; i++)
    {
        std::cout << "1";
    }
    std::cout << "\n";
}
```

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:

Noam Chomsky amerika nyelvész nevéhez kötődik a generatív nyelvtan koncepciójának kidolgozása, a Chomsky-hierarchia megalkotása.

A formális nyelvek osztályozására szolgál a fentebb említett modell, ami a formális nyelv elemeit kifejező-erő alapján sorolja be, az erősebb osztályok a gyengébb osztályokat(azok elemei) generálására képesek. A generatív nyelvtan 4 alapköve:

- terminális szimbólumok: betűk, egy ábécének
- nem terminális jelek: egy diszjunkt ábécé, ami generálási segédeszközként van jelen
- kezdőszimbólum: egy kitüntett szimbólum

- helyettesítési szabályok

A generálás a következőképpen zajlik le: adott egy szó(mondatszimbólum, azaz a mi esetünkben az S változó) vizsgálatra, ennek a szónak a részszavait a már (akár általunk) meghatározott helyettesítési szabály alapján lecseréljük egy másik szóra, és ezt (műveletet) addig hajtjuk végre folyamatosan amíg a szavunk csak terminális jelekből áll.

A feladatban megadott nyelv nem környezetfüggetlen, ami annyit jelent, hogy nem lehet olyan hozzárendelési szabállyal legenerálni ami ilyen alakú:

H -> h

Tehát nincs olyan képzési szabálya aminek bal oldalán csak egyetlen egy nem-terminális jel van(H), a h pedig tetszőleges.

Környezetfüggő nyelvek amik a megadott nyelvet állítja elő:

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

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

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

!. aYbbcc (aY - aaX)
   aaXbbcc (Xb - bX)
   aabXbcc (Xb - bX)
   aabbXcc (Xc - Ybcc)
   aabbYbccc (bY - Yb)
   aabYbbccc (bY - Yb)
   aaYbbbccc (aY - aa)
   aaabbbccc
```

```
   A, B, C változók
   a, b, c konstansok
   A - aAB, A - aC, CB - bCc, cB - Bc, C - bc

!. A (A - aAB)
   aAB ( A - aC)
   aaCB (CB - bCc)
   aabCc (C - bc)
   aabbcc
```

```
!. A (A - aAB)
   aAB ( A - aAB)
   aaABB ( A - aAB)
   aaaABBB ( A - aC)
   aaaaCBBB (CB - bCc)
   aaaabCcBB (cB - Bc)
   aaaabCBcB (cB - Bc)
   aaaabCBBc (CB - bCc)
   aaaabbCcBc (cB - Bc)
   aaaabbCBcc (CB - bCc)
   aaaabbbCccc (C - bc)
   aaaabbbbcccc
```

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:

c89.c

Tanulságok, tapasztalatok, magyarázat...

Mielőtt delfiniálnánk a C utasítást fogalmát... Mi az a *B N F*?

A környezet-független szintaxisokat formalizáló szintaxis, azaz nyelvtanokat definiáló nyelvtan. Az ismertebb programozási nyelvek kulcsszavai../szintaxisa../.. ebben van megadva

Hogyan használjuk?

- Szarmaztatási szabályoknak halmazát leírjuk például így:
-

```
<egész szám> ::= <előjel><szam>
<előjel> ::= [-|a]
<szam> ::= <szamjegy>{<szamjegy>}
<szamjegy> ::= 0|1|2|3|4|5|6|7|8|9
```

•

```
<fájl> ::= <tartalom>
<tartalom> ::=<mező>
<mező> ::= <számok> | <karakterlánc>
<számok> := <számjegy>{<szamjegy>}
<szamjegy> ::= 0|1|2|3|4|5|6|7|8|9
<karakterlánc> ::=<karakter>
<karakter> ::= <betű> | <számjegy>
<betű> ::= A|B|...|Z | a|b|...|Z
```

Ezek alapján a C utasítás fogalma BNF-ben:

```
<utasítás>::=
            <összetett_utasítás>
            <kifejezés>; (értékadás pl, num=10)
            if(<kifejezés>) <utasítás>
            else if(<kifejezés>) <utasítás>
            else <utasítás>
            switch (<kifejezés)
            <egész_konstans_kifejezés : <utasítás>
            goto <azonosító>;
            <azonosító> : <utasítás>
            break; continue; return<kifejezés>;
            or(<kifejezés1><kifejezés2><kifejezés3>) <utasítás>
            while(<kifejezés>) <utasítás>
            do <utasítás> while<kifejezés>
            ; (üres utasítás, pl FORTRAN continue-ja)
```

C89 szabvánnyal nem leforduló kód

```
int main()
{
        //unsigned long long eternity = 1917;

        return 0;
}
```

```
cant0r@dev:~/Dev/bhax/src/Chomsky/BNF$ gcc -std=c89 c89.c -o c89
c89.c: In function ‘main’:
```

```
c89.c:3:2: error: C++ style comments are not allowed in ISO C90
  //unsigned long long eternity = 1917;
  ^
c89.c:3:2: error: (this will be reported only once per input file)
cant0r@dev:~/Dev/bhax/src/Chomsky/BNF$
```

C99 szabvánnyal leforduló kód

```
int main()
{
        //unsigned long long eternity = 1917;

        return 0;
}
```

```
cant0r@dev:~/Dev/bhax/src/Chomsky/BNF$ ls
c89.c
cant0r@dev:~/Dev/bhax/src/Chomsky/BNF$ gcc -std=c99 c89.c -o c99
cant0r@dev:~/Dev/bhax/src/Chomsky/BNF$
```

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:

c89.c

c89.c

FUTTATÁSHOZ SZÜKSÉGESEK
a flex és bison csomagok telepítése.

A lexer egy olyan program amit arra használunk, hogy szövegelemző/olvasó programokat generáljunk. Mik a szövegelemző/olvasó programok? Olyan programok amik forráskódot beolvasnak és felismerik a "magabbrendű" nyelvtani formációkat(token-ek) a beolvasott karakterek között.

A FLEX(Fast LEXical analyzer generator) ilyen előbb említett szkennereket alkot a megadott szótár alapján, minták specifikációját kell csak megadunk regurális kifejezésekkel.

FLEX használata

A programunk a következő:

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

A forráskód három fő részből áll, a dupla % jelek választják szét őket.

1. Definíciók helye, ezek garantáltan benne lesznek a majd generálásra kerülő C programba, tehát ide pakolhatunk olyan dolgokat, mint például változók.
2. Szabályok helye, megadjuk a regExp-t a feldolgozáshoz. Ebben a példában valós számokat keresünk, emiatt a következő szabályt írtuk fentebb: Várunk egy számjegyet amiből lehet tetszőleges darab('*' = 0 vagy több), a szám(ok) után pedig vagy lehet vagy nem lehet egy legalább 1 számjegy pontosan egy darab pont után.
3. Felhasználói kód helye

Fordítás:

lex -o lex.c lex.l

gcc -o lex lex.c -lfl

Működtetési példák A standard bemenetről várja karakterek sorozatát, egészen EOF-ig, mi a következő szöveget fogjuk beolvastatni:

```
Tulajdonképpen nincs is miért aggódnunk.
A kutatási eredményeink szerint még legalább 2 hónapig,
4 napig és 7 óráig
nem fog bekövetkezni kisugárzás.
Annak is 3-as a várható erőssége
```

```
skálán ami pontosan 2.13.
```

Lexer, ha nem követeljük meg a számjegy jelenlétét a . előtt.

Lexer, ha megkövetelük a számjegy jelenlétét a . előtt.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

l33t.l

l33t.c

Tanulságok, tapasztalatok, magyarázat...

A program lex szerinti szerkezetét és magát a flex csomagokat nem részletezném, ugyanis az előző feladatnál már bemutatásra került.

Itt a definíciók (%{*}%) helyénél megadtuk a l33t szövegre fordítás szabályait, egy karakterhez négy féle hozzáhasonló, őt ábrázolni próbáló karaktert vagy *karakterláncot* adtunk meg. Ezt a sajátos szótárat egy `l37td1c7` struktúrában tároljuk kényelmi okokra hivatkozva.

Ugyanitt persze behívtuk a működéshez szükséges headereket(pl rand() miatt stdlib.h, time.h) is.

A szabályok blokkja pedig megadja, hogyan járjunk el standard inputról beolvasott szöveg fordításakor.

Lényege: Végiglépkedünk az *input* karakterein, ha az éppen vizsgálat alá vetett karakter megtalálható a szótárunkban, akkor generálunk egy egész számot 1 és 100 között és ha ez a generált érték megfelel valamelyik esélynek(lásd lentebb) akkor helyette az esélyhez rendelt hasonmás karaktert küldjük a standard kimenetre, ellenben ő magát.

Az esélyek:

1. 90% annak az esélye, hogy a `l337d1c7.leet` 0. elemét iratjuk ki STDOUT-ra.
2. 4% annak az esélye, hogy a `l337d1c7.leet` 1. elemét iratjuk ki STDOUT-ra.
3. 3% annak az esélye, hogy a `l337d1c7.leet` 2. elemét iratjuk ki STDOUT-ra.
4. 3% annak az esélye, hogy a `l337d1c7.leet` 3. elemét iratjuk ki STDOUT-ra.

Ahhoz, hogy tudjuk, hogy volt e **SPECIÁLIS** karakter kiiratva egy egyszerű segédváltozót vezettünk be, aminek ha az értéke 0 marad (hamis) akkor nem volt **SPECIÁLIS** karakter STDOUT-ra küldve, tehát simán kiiratjuk a bevitt karaktert.

A forráskód és pár példa:

```
%{
#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"}}

  };

%}
%%
. {

    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;
}
```

MINTASZÖVEG

```
Szabad szoftver alatt értünk minden számítógépes programot
```

```
 és dokumentációt, amely kielégíti az alábbi feltételeket:

A szoftver bármilyen célra felhasználható.
Lehetőség van a szoftver működésének szabad
tanulmányozására és módosítására.
Szabadon terjeszthető, továbbadható.
Lehetőség van a szoftver továbbfejlesztésére
és a fejlesztés közreadására.

A szoftver tanulmányozásának, módosításának,
illetve továbbfejlesztésének előfeltétele
 a forráskód elérhetősége.
```

Egy futtatás után.

Több futtatás után.

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?

Megoldás forrása:

Megoldás videó:

wotan.cpp TESZTELÉSEK HELYE, SZEMRE ÁRTALMAS

signals.c

Tanulságok, tapasztalatok, magyarázat...

i.

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

Csakis akkor kezelje a `jelkezelo` függvény a SIGINT jelet, ha az (SIGINT) nincs ignorálva.

Ez a kódrész azoknál a programoknál fordul elő gyakran amelyek ezeket a szignálokat kezelik, vannak olyan esetek amikor a shell egy background processzus indítása előtt a SIGINT és SIGQUIT jeleket ignorálja, tehát gyakori, hogy a progrmaok ellenőrzik, hogy nincsenek-e ignorálva ezek a jelek.

Fontos lehet megjegyezni, hogy a `fork` meghívása után a gyerek ÖRÖKLI a szülejének a jelkezelőit, hiszen a szülő és a gyerek osztozkodik egy címtartományon(memória).

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

void jelkezelo(int a)
{
  printf("Helló!\n");
}

int main()
{
  /*for(;;)
  {
  usleep(60000);
    if(signal(SIGINT, jelkezelo) == SIG_IGN)
    {
      signal(SIGINT,SIG_IGN);
      break;
    }
  }*/
    if(signal(SIGINT,SIG_IGN) != SIG_IGN)
    {
      signal(SIGINT, jelkezelo);

    }
  for(;;)
  {

    usleep(60000);

  }
}
```

figyelem
A `signal` függvény tényleges használatához a `signal.h` használata szükséges. **man 2 signal**

Mit mond a splint?

```
cant0r@dev:~/Dev/bhax/src/Wotan$ splint wotan.c
Splint 3.1.2 --- 20 Feb 2018

wotan.c: (in function main)
wotan.c:39:18: Unrecognized identifier: jelkezelo
  Identifier used in code has not been declared. (Use -unrecog to  ↩
     inhibit
  warning)
wotan.c:39:3: Return value (type [function (int) returns void])  ↩
   ignored:
                signal(SIGINT, j...
  Result returned by function call is not used. If this is intended,  ↩
     can cast
  result to (void) to eliminate message. (Use -retvalother to inhibit  ↩
     warning)

Finished checking --- 2 code warnings
cant0r@dev:~/Dev/bhax/src/Wotan$
```

ii.

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

Egy for ciklus, ami 5-ször hajtódik végre 1-s inkrementációval, az inkrementáció alakja preorder, ami annyit jelent, hogy `++i` visszatérési értéke az új `(i+1)` érték. Ez nem módosítás az iterálások számán.

iii.

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

Egy for ciklus, ami 5-ször hajtódik végre 1-s inkrementációval, az inkrementáció alakja postorder, ami annyit jelent, hogy `i++` visszatérési értéke a régi `i` érték. Ez nem módosítás az iterálások számán.

iv.

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

Egy for ciklus, ami 5-ször végrehajtódik és a tömb értékeit lecseréli `i++` visszatérési értékeivel. Rendre 0,1,2,3,4 számok lesznek az adott tömb értékei.

FELTÉTELEZI
a program, hogy létezik `tomb` és annak elemeinek száma 5.

Mit mond a splint?

```
cant0r@dev:~/Dev/bhax/src/Wotan$ splint wotan.c
Splint 3.1.2 --- 20 Feb 2018

wotan.c: (in function main)
wotan.c:19:30: Unrecognized identifier: tomb
  Identifier used in code has not been declared. (Use -unrecog to ↩
     inhibit
  warning)
wotan.c:19:41: Expression has undefined behavior (left operand uses i, ↩
    modified
                  by right operand): tomb[i] = i++
  Code has unspecified behavior. Order of evaluation of function ↩
     parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining ↩
     evaluation
  order, then the result of the expression is unspecified. (Use - ↩
     evalorder to
  inhibit warning)

Finished checking --- 2 code warnings
```

v.

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

Egy for ciklus ami addig megy amíg `i` ciklusváltozó kisebb mint az `n` ÉS `(*d++)` visszatérés értéke érvényes. `d` és `s` értelemszerűen mutatók, de ahogy a szintaxisból kikövetkeztethető igazából tömbök első elemére(legleőször) mutató mutatók, a ()-en belüli kifejezés azonban csak akkor működik ha explicit módon létrehozunk egy-egy mutatót a tömbjeinkre `d` és `s` néven, nem használhatjuk csupán a tömb azonosítóját.

FELTÉTELEZZÜK
hogy létezik `n`, `d`, `s` változók.

Mit mond a splint ha léteznek a létezetlenek?

```
cant0r@dev:~/Dev/bhax/src/Chomsky/Wotan$ splint wotan.c
Splint 3.1.2 --- 20 Feb 2018

wotan.c: (in function main)
wotan.c:33:18: Right operand of && is non-boolean (int): i < n && (*d ↩
   ++ = *s++)
```

```
  The operand of a boolean operator is not a boolean. Use +ptrnegate ↩
     to allow !
  to be used on pointers. (Use -boolops to inhibit warning)
wotan.c:38:2: Return value (type int) ignored: putchar('\n')
  Result returned by function call is not used. If this is intended, ↩
     can cast
  result to (void) to eliminate message. (Use -retvalint to inhibit ↩
     warning)

Finished checking --- 2 code warnings
cant0r@dev:~/Dev/bhax/src/Chomsky/Wotan$
```

vi.

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

A STANDARD KIMENTRE kiírjuk az `f` függvény visszatérési értéket, ami legalább egy 4 bájtos egész általában, de mivel nincsen meghatározva, hogy milyen sorrendben lesznek kiértékelve! A C nyelv nem határozza ezt meg, így bugos a kód, de nagyon.

```
#include <stdio.h>

int f(int a, int b)
{
  printf("HELLÓ!: %d %d\n", a,b);
  return a;
}

int main()
{
  int a=0;
  printf("%d %d", f(a, ++a), f(++a, a));

  return 0;
}
```

```
cant0r@dev:~/Dev/bhax/src/Wotan$ ./wotan
HELLÓ!: 1 1
HELLÓ!: 2 2
2 1
cant0r@dev:~/Dev/bhax/src/Wotan$
```

FELTÉTELEZZÜK
hogy létezik `a` változó és `f` függvény.

Mit mond a splint?

```
cant0r@dev:~/Dev/bhax/src/Wotan$ splint wotan.c
Splint 3.1.2 --- 20 Feb 2018

wotan.c: (in function main)
wotan.c:30:18: Unrecognized identifier: f
  Identifier used in code has not been declared. (Use -unrecog to  ↩
     inhibit
  warning)
wotan.c:30:20: Unrecognized identifier: a
wotan.c:30:18: Argument 2 modifies <type <any>>, used by argument 3 ( ↩
   order of
    evaluation of actual parameters is undefined):
    printf("%d %d", f(a, ++a), f(++a, a))
  Code has unspecified behavior. Order of evaluation of function  ↩
     parameters or
  subexpressions is not defined, so if a value is used and modified in
  different places not separated by a sequence point constraining  ↩
     evaluation
  order, then the result of the expression is unspecified. (Use - ↩
     evalorder to
  inhibit warning)
wotan.c:30:29: Argument 3 modifies <type <any>>, used by argument 2 ( ↩
   order of
    evaluation of actual parameters is undefined):
    printf("%d %d", f(a, ++a), f(++a, a))

Finished checking --- 4 code warnings
```

vii.

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

Kiiratjuk STANDARD KIMENETRE az `f` függvény által visszatérített értéket(legalább 4 bájtos egész), az `f`(lehet akár makró is) argumentumként átadjuk `a`-t, az `a` változó értékét adjuk át, tehát az `f` csak egy másolatán fog dolgozni tudni, és kiiratjuk `a` által jelenleg hordozott értékét.

FELTÉTELEZZÜK
hogy létezik `a` változó és `f` függvény.

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

Kiiratjuk STANDARD KIMENETRE az `f` függvény által visszatérített értéket(legalább 4 bájtos egész), az `f`(lehet akár makró is) paramétere egy memóriacím, hacsak nem C++ nyelvről beszélünk(ekkor ugye behívtuk a standard C könyvtárat printf miatt), mert akkor referencia is lehet, és kiiratjuk `a` által jelenleg hordozott értékét.

FELTÉTELEZZÜK
hogy létezik `a` változó és `f` függvény.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

`blastFromThePast.tex`

`blastFromThePast.pdf`

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

Mi az Ar nyelv?

Egy futtatás után.

A megadott leírás után egyszerűen értelmezhetjük, egy kivétellel, az összetett elsőrendű formulákat.

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

A prímszámok száma végtelen.

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

Az ikerprímek száma végtelen.

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

A prímszámok száma véges.

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

A prímszámok száma végtelen.

Legutolsó kiértékelésünk hogyan jött ki? Egyszerűen pár trükköt alkalmaztunk. Legelőször kiemeltük a kvantorokat, a kvantorkiemelés szabályai szerint, majd az implikációs részformulát átalakítottuk konjunkciós részformulává, s eközben eltünt a negáció művelete a kétszeres tagadás szabálya miatt, így újra megkaptuk a legelső formulát. Ezek ekvivalens átalakítások voltak.

3.8. Deklaráció

Megoldás videó:

Megoldás forrása:

dec.cpp

test.c

tmp.c

Vezesd be egy programba (forduljon le) a következőket:

figyelem
BEVEZETÉSÜK A LISTA UTÁN TÖRTÉNIK MEG

- 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

```
int* getReactorPointer(int* reactorArray);
typedef int(*R)(int,int);
int getTrueReactorState(int a, int b);
R getReactorState(int);

int main()
{

  //egész
  int chernobyl = 1986;

  //egészre mutató mutató
  int* toChernobyl = &chernobyl;

  //C++ feature
  int& refToChernobyl = chernobyl;

  //egészek tömbje
  int reactorsOfCNPP[4] = {1,2,3,4};

  //egészek tömbjének referenciája
  int (&refToReactors)[4] = reactorsOfCNPP;

  //egészre mutató mutatók tömbje
  int * pointerArray[4];

  //egészre mutató mutatót visszaadó függvény
  int* myReactor = getReactorPointer(reactorsOfCNPP);

  //egészre mutató mutatót visszaadó függvényre mutató mutató
  int* (*pointerFUN)(int*) = getReactorPointer;

  //egészet visszaadó és két egészet kapó függvényre mutató mutatót ↩
     visszaadó, egészet kapó függvény
  R getTheTrueState = getReactorState(1986);

  return 0;

}
int* getReactorPointer(int* reactorArray)
{
```

```
  return reactorArray;
}
int getTrueReactorState(int a, int b)
{
  return -1986;
}

R getReactorState(int wishGranter)
{
  return getTrueReactorState;
}
```

Mit vezetnek be a programba a következő nevek?

test.c

tmp.c

Tanulságok, tapasztalatok, magyarázat...

- ```
  int a;
  ```

  Egy a jelölt egész típusú változót.
- ```
  int *b = &a;
  ```

  Egy egészre mutató mutatót b azonosítóval ami az előbbi a változót címét tartalmazza.
- ```
  int &r = a;
  ```

  Egy egészre mutató mutatót r névvel ami az a értékét mint mutatócím tartalmazza. Ha lefordítjuk(figyelmeztet) és futtajuk párszor észrevehetjük, hogy mindig változik az r értéke.
- ```
  int c[5];
  ```

  Egy 5 elemű, jelölt egészekből álló tömböt.
- ```
  int (&tr)[5] = c;
  ```

  Egy 5 elemű, jelölt egészekből álló tömbre mutató mutatót, ami a c tömbre mutat.
- ```
  int *d[5];
  ```

  Egy 5 elemű, egészekre mutató mutatókból álló tömböt.
- ```
  int *h ();
  ```

  Egy jelölt egésszel visszatérő, zéró paraméteres függvényre mutató függvénymutatót.
- ```
  int *(*l) ();
  ```

Egy jelölt egésszre mutató mutatóval visszatérő, zéró paraméteres függvényre mutató függvénymutató.

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

  Egy jelölt egésszel visszatérő 2 jelölt egészet váró függvényre mutató mutatóval visszatérő, 2 jelölt egészet váró függvény.

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

  Egy jelölt egésszel visszatérő, két jelölt egészt paraméterként váró függvényre mutató, jelölt egészet váró függvényre mutató mutató.

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

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

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

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

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

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

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

    }

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

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

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

    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;
}
```

A double ** háromszögmátrix a memóriában

4.1. ábra. A double ** háromszögmátrix a memóriában

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

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: https://youtu.be/XpBnR31BRJY

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

Tanulságok, tapasztalatok, magyarázat...

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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

A Mandelbrot halmaz a komplex síkon

5.1. ábra. A Mandelbrot halmaz a komplex síkon

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

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

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

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

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

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

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

Megoldás forrása:

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

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040  ↩
   -0.019473810573093663922605855987058021128l8  ↩
   -0.019473810572541341845642648422654019668l7  ↩
   0.798505756933826860155534177465597167611l  ↩
   0.79850575693437919611028519284445792436 6
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913l  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327099751215428l  ↩
   0.213538705180497528912653137922461610287 4
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

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


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

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

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

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

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

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

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

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

```
    {
      // k megy az oszlopokon

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

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

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

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

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

              ++iteracio;
            }

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

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

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

}
```

5.3. Biomorfok

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

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

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

A különbség a Mandelbrot halmaz és a Julia halmazok között az, hogy a komplex iterációban az előbbiben

a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

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

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

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

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

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

            ++iteracio;
          }
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása:

5.6. Mandelbrot nagyító és utazó Java nyelven

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

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

6. fejezet

Helló, Welch!

6.1. Első osztályom

Valósítsd meg C++-ban és Java-ban az módosított polártranszformációs algoritmust! A matek háttér teljesen irreleváns, csak annyiban érdekes, hogy az algoritmus egy számítása során két normálist számol ki, az egyiket elspájzolod és egy további logikai taggal az osztályban jelzed, hogy van vagy nincs eltéve kiszámolt szám.

Megoldás videó:

Megoldás forrása:

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

Python

Megoldás videó: https://youtu.be/j7f9SkJR3oc

Megoldás forrása: https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0 (/tensorflow-0.9.0/tensorflow/exa
https://progpater.blog.hu/2016/11/13/hello_samu_a_tensorflow-bol

Tanulságok, tapasztalatok, magyarázat...

8.2. Mély MNIST

Python

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

8.3. Minecraft-MALMÖ

Megoldás videó: https://youtu.be/bAPSu3Rndi8

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

Megoldás videó: https://youtu.be/z6NJE2a1zIA

Megoldás forrása:

9.2. 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...

9.3. 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...

10. fejezet

Helló, Gutenberg!

10.1. Juhász István: Magas Szintű Programozási Nyelvek 1

A való világ túlságosan bonyolult ahhoz, hogy tökéletesen lemásoljuk a mechanikai működését, mégis a programozás valamelyest ezzel a céllal jött létre. Az emberi gondolkodást, modellezést próbálja utánozni, ezzel segítve az embereket a problémák megoldásában.

Először is pár alapfogalmat tisztáznunk. A számítógépek programozására kialakult nyelveket három szintre tagoljuk:

- -Gépi nyelv: Ez a legalsó nyelv, ez áll a legközelebb a géphez, közvetlenül ezt képes értelmezni a processzor, általában kettes számrendszerben íródok, de van pl. hexadecimális, azaz tizenhatos számrendszeren alapuló gépi kód is. Az ember és a gép között ez a nyelv a gép felé áll.
- -Assembly szintű nyelv: Az assembly egy program nyelv, amit csaknem minden programozási nyelv előállít végeredményként. Szimbolikus gépi kódnak is szokták nevezni.
- -Magas szintű nyelv: Ezzel foglalkozik a programozó, ez áll a legközelebb az emberi nyelvhez, a könnyű érthetőség érdekében. A magas szintű programozási nyelveket át kell alakítanunk valamilyen módon, pl. interpreteres, azaz értelmező móddal, vagy fordítóprogram segítségével. A fordítóprogram gépi kódot állít elő.

A Fordító Működése: Ahhoz hogy lefordítsa a program a forráskódot, a következő lépéseket hajtja végre a fordítóprogram:

- -Lexikális elemzés: A lexikális elemzés során a forrásszöveget feldarabolja lexikális egységekre.
- -Szintaktikai elemzés: A szintaktikai elemzés folyamán ellenőrzi, hogy teljesülnek-e az adott nyelv szintaktikai szabályai.
- -Szemantikai elemzés
- -Kódgenerálás

Minden nyelv rendelkezik saját szabályokkal, ami természetesen változhat. Ezek a szabályok affajta használati utasítás, korlátozás, mit tehetünk és hogyan. „C” esetében felhozhatjuk példaként a C89 és a C99 szabványt. Sokféle fordító program és forráskód szerkesztő létezik, úgynevezett IDE-k, úgymint Code::Blocks, Visual Studio, NetBeans.

Utasításokból áll az algoritmus, azokat lépésenként végrehajtja, illetve ezek alapján áll össze a tárgykód. Utasításokat két részre oszthatjuk:

- Deklarációs utasítások
- Végrehajtandó utasítások

10.2. Kerninghan & Richie

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

Minden programozási nyelvben közös, hogy változók és állandók alkotják a programban feldolgozott alapvető adatobjektumokat. Deklaráláskor szükséges megadni a típusukat, nevüket, és általában ezeknek már létrehozásukkor értéket utalnak. Operátorokat arra használjuk, hogy megadjuk a gépnek mit is csináljon az adattal. A típus megadása fontos lépés, ez határozza meg milyen értéket tartalmaz, ad vissza, illetve milyeneket kaphat.

Most a „C” nyelvvel fogunk foglalkozni.

Az ANSI szabvánnyal kibővült a lehetőség a programozók számára. Minősítők kerültek bele, melyek teret adtak a programozóknak hogy más hosszúságú egészekkel is dolgozhassanak. Ilyen minősítők pl. a „signed” és „unsigned” forma, amely az előjel meglétét vagy hiányát mutatja.

Bevezették a

```
long double
```

adattípust, ezzel együtt a „short” előjelet is. A „short” 16bites határral bír, a „long” viszont 32bites.

C-vel foglalkozva kevés alapvető adattípust használunk:

```
char
  //egyetlen bájt, a gépi karakterkészlet egy elemét tárolja


int
  //egész szám, mérete általában a befogadó számítógép egészek ↩
     ábrázolásához használt mérete
```

```
float
  //egyszeres pontosságú lebegőpontos szám


double
  //kétszeres pontosságú lebegőpontos szám
```

Ezekehez társulnak minősíthető specifikáció, pl 'short' és 'long'

```
short int valami;
long int nagyValami;
```

10.3. BME Szoftverfejlesztás C++ Nyelven

A „C++" a „C" nyelv család tagja, sok mindent örökölt tőle, legtöbbet említett örökölt tulajdonsága a „hardverközelség". A „C++" egy Objektumorientált nyelv, a mai világban egyik legelterjedtebb nyelv, alapja sok operációs és embedded rendszernek.

Alapjai a „C"-re épül, azonban vannak lényeges különbségek a két nyelv között, példaként véve egy „C++" függvényt.

```
Void f(){}
```

A függvénynek nincs paramétere, azaz nem lehet paraméterrel hívni. Azonban ha „C"-ben is ezt szeretnénk elérni, módosítanunk kell a függvényt, és paraméterként „..." kell írnunk.

```
Void f(...){}
```

Új típus is felüti a fejét, a 'bool'. Ennek értéke True vagy False lehet. Ez segíti az olvashatóságot, azonban átalakíthatjuk 'int'-re is, 0 vagy 1 értékké alakul.

'C++' teret adott egy új lehetőségnek: 'függvény túlterhelés' lehetségessé vált. 'C++'-ban beazonosítjuk a függvényeket a nevük és argumentum listájuk alapján, ezáltal megnyílik a lehetőség hogy ugyanazon névvel ellátott, viszont különböző argumentum listával rendelkező függvényeket képezzünk, csakis a visszatérési érték, ami nem változhat.

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.

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. 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

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. 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.