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	2019. április 2.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

Ábrák jegyzéke

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás videó:

Megoldás forrása:

A következő program egy magot 100 százalékban dolgoztat meg:

```
#include <stdio.h>

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

Ez a program egy egyszerű while ciklus, aminek ha a feltétele teljesül, akkor nem csinál semmit. Ebben az esetben a feltétel 1, ami 'igaz' vagy angolul 'true' boolean típusú értékkel bír, tehát az adott ciklus egy végtelen ciklus, azaz sosem áll le, kivéve ha arra kényszerítjük a CTRL+C betűkombinációval, hiszen az 'igaz' értéke mindig igaz.

A következő program minden magot 100 százalékban dolgoztat meg:

```
            ]
// compile: gcc name.c -fopenmp

#include <stdio.h>

int main()
{
    #pragma omp parallel
    while(1){};
}
```

Ez az előző program módosítása. Ugyanabból a while ciklusból áll némi változtatással. A #pragma feladata, hogy a gép vagy az operációs rendszerre vonatkozó specifikus utasítást adjon a compilernek, azaz megmondja hogy a compiler tegyen valamit vagy felülírjon valamilyen általánosnak vett utasítást. Itt a #pragma szerepe a felülírás. Normális esetben, mint azt felül is láthattuk, ez a program #pragma nélkül csak 1 processzort dolgoztat meg 100 százalékon, de az omp parallel utsítás azt mondja a compilernek, hogy egyszerre több szálon fusson a program, azaz több processzor dolgozzon rajta egy időben, aminek az eredménye, hogy ez a program az összes processzort 100 százalékosan megdolgoztatja.

A következő program 0 százalékon dolgoztat meg egy magot:

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

        int main()
        {
            while(1)
            {
                usleep(1000);
            }
        }
```

Ez a program a while ciklusban a usleep(1000) utasítást végzi el. A usleep(seconds_t usec) az unistd.h fájlban definiált függvény. Adott usec mennyiségű mikroszekundumig megszakítja a program működését.

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!

Megoldáás videó:

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

```
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 épülő Lefagy2 már nem tartalmaz feltételezett, csak csak konkrét kódot:

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

- Ha T1000 lefagyó, akkor nem fog lefagyni, kiírja, hogy true

- Ha T1000 nem fagyó, akkor pedig le fog fagyni...

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

A megoldási probléma lényege hogy nem tudunk olyan programot írni, amely eldönti egy másik programról, hogy az lefagy-e vagy sem. Ahhoz hogy a T100 eldöntse, hogy egy program lefagy-e vagy sem, elősször le kell futtatnia azt és megvizsgálni a következményeket. Ha a prgoram lefutott, a T100 kiírja hogy a programunk lefutott, de ha a program lefagy, akkor a T100 is le fog fagyni, mert a döntés előtt le kell futtatnia azt. Ha a fenti T1000 programot önmagára futtatjuk, akkor két eshetőség adódik: Ha a programunk nem fagy le, akkor bekövetkezik a végtelen ciklus a T1000-ben és így lefagy. Ha pedig lefagy a programunk, akkor mivel a program lefagy ezért a T1000 is lefagy.

2.3. Változók értékének felcserélése

Írj olyan C programot, amely felcseréli két változĂł értékét, bármifĂŠle logikai utasítás vagy kifejezés nasználata nélkül!

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

```
#include <stdio.h>

int main()
{
    int a = 10;
    int b = 4;
    printf("a = %d\n", a);
    printf("b = %d\n", b);

    a = a - b;  a = 10 - 4 / 6
    b = a + b;  b = 6 + 4 / 10
    a = b - a;  a = 10 - 6 / 4

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

A fenti C program kiírja két változó értékét, felcseréli őket, majd kiírja a változók új értékeit. Az int a = 10 és int b = 4 megadják a és b értékeit. Először a-t egyenlővé tesszük a - b -vel (itt így a = 10 - 4 = 6). Majd b-t egyenlővé tesszük a + b -vel, de mivel a = a - b, ezért b = a + b = a - b + b = a (b = 10 - 4 + 4 = 10). Ekkor a-t egyenlővé tesszük b - a -val. Itt b = a és a = a - b -ként van definiálva, tehát a = b - a = a - (a - b) = a - a + b = b (a = 6 + 4 - (10 - 4) = 4) A printf() függvény kiírja a paraméteréül kapott stringet. Például a fent látható printf("a = %d\n", a) azt fogja kiírni hogy a = 10, majd egy új sort kezd. Ebből "a = %d\n" a string rész, amiben a %d egy format specifier. Feladata, hogy megmondja, a %d helyére egy decimális egész szám (itt: a) fog kerülni, amit a string rész utáni vesszővel elválasztott helyre kell írni. Ugyanitt a \n egy escape sequence, ami azt mondja a printf függvénynek, hogy a \n helyére egy új sort írjon.

Ezek után megkezdődik a változók felcserélése a fenti műveletek segítségével. Miután az kész, a program ismét kiírja a változók immár új felcserélt értékeit.

2.4. Labdapattogás

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

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

Labdapattogtatás if nélkül:

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

int write_x(x,y)
{
    int xi,yi;

    for(xi=0;xi<x;xi++)
    {
        printf("\n");
    }

    for(yi=0;yi<y;yi++)
    {
        printf(" ");
    }
    printf("X\n");
    return 0;
}

int main()
{

    int width = 90;
    int height = 25;
    long int x=0,y=0;

    while(1)
    {
        system("cls");
        write_x(abs(height - (x++ % (height * 2 ))), abs(width - (y ↩
            ++ % (width*2))));
        usleep(10);
    }

    return 0;
```

```
        }
```

A program Windows alatt egy adott 'pályán' pattogtat egy x-el jelölt 'labdát' a konzolon. Két függvényből áll, a már megszokott int main() és egy int write_x(x,y) függvényből. A main függvényben meg van adva a pálya hossza, magassága és a labda koordinátái, amelyek alapesetben nullával egyenlőek. Ezek után a program egy végtelen ciklusban először törli a képernyőt a system("cls") függvénnyel, majd meghívja a write_x() függvényt két abszolút értékre. Egy szám abszolút értékét az abs() függvénnyel számoljuk ki, ami a math.h fájlban van meghatározva. Ha a write_x() lefutott, akkor egy ideig megszűnik a munkavégzés és előről kezdődnek a ciklusban definiált lépések.

Az int write_x(x,y) itt egy a felhasználó által megadott függvény. Az előtte lévő int szó azt jelenti, hogy a függvény egy integer, azaz egész szám típusú értéket fog visszaadni, amit a return 0 paranccsal teszünk meg (return 0 azt jelenti, hogy a program probléma nélkül lefutott. Ha nem nullát ad vissza, akkor futás közben hiba történt.). A zárójelek közötti x és y számok a függvény paraméterei, tehát ezekkel fog dolgozni. A számokat függvényhívásnál kell megadni. A függvényen belül két for ciklus található. Az első feladata, hogy a labda függőleges helyzetének megfelelő új sort írjon ki, a másodiké pedig, hogy úgyanúgy a labda vízszintes pozíciójának megfelelő szóközt írasson ki. Ha ez megtörtént, akkor a labda jelenlegi koordinájaihoz érkezett a kurzor. A for ciklusok befejeződnek és printf()-el kiíratunk egy X-et.

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:

```
            // BHAX BogoMIPS
// 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
//
```

```
// This program is based on
//
// - Linus Torvalds's original code (https://mirrors.edge.kernel.org/pub/ ↩
   linux/kernel/v1.0/linux-1.0.tar.gz init/main.c)
// - and Jeff Tranter's standalone version (archive.debian.org/debian/pool/ ↩
   main/s/sysutils/sysutils_1.3.8.5.1.tar.gz).
//
// See also UDPROG
//

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

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

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

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

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

      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;
}
```

2.6. Helló, Google!

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

Megoldás videó:

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

void kiir (double tomb[], int db);
double tavolsag(double pagerank[], double pagerank_temp[], int db);

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};

  for (;;)
  {
    for(int i=0; i<4; i++)
        {
      PR[i] = PRv[i];
    }

    for (int i=0; i<4; i++)
        {
      double temp = 0.0;

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

    if(tavolsag(PR, PRv, 4) < 0.000001)
        {
      break;
    }
```

```
  kiir(PR, 4);

  return 0;
}

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

double tavolsag(double pagerank[], double pagerank_temp[], int db)
{
  double dist = 0.0;

  for(int i=0; i<db; i++)
  {
    dist += (pagerank[i] - pagerank_temp[i]) * (pagerank[i] - pagerank_temp ↩
        [i]);
  }

  return sqrt(dist);
}
```

Ez a program egy 4 honlapból álló hálózatra kiszámolja a négy lap Page-Rank értékét.

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

```
#   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")
```

A Brun tétel szerint az ikerprímek reciprokösszege egy Brun-konstans nevű értékhez konvergál. Az ikerprímek olyan prímpárok, melyek különbsége 2. Egy prímpár pedig egy olyan prím, ami egy másik prímtől 2-nél nagyobb, vagy kisebb. Példaképp az 1 és a 3 számok. Ekkor 1/1 + 1/3 = 1.333 . A Brun-konstans megközelítő értéke 1,90216 . Itt a program első felében a primes(x) függvényt definiáljuk, ami egyrészt kiírja x-ig a prímszámokat, másrészt kiszámolja az ikerprímek reciprokösszegeit. Azután a második részben kiplottolhatjuk a függvény által kiszámolt eredményeket. Viggo Brun, norvég matematikus bizonyította be a tételt 1919-ben.

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

```
#    An illustration written in R for the Monty Hall Problem
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
```

```
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>
#
#    https://bhaxor.blog.hu/2019/01/03/erdos_pal_mit_keresett_a_nagykonyvben ↩
    _a_monty_hall-paradoxon_kapcsan
#

kiserletek_szama=10000000
kiserlet = sample(1:3, kiserletek_szama, replace=T)
jatekos = sample(1:3, kiserletek_szama, replace=T)
musorvezeto=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

    if(kiserlet[i]==jatekos[i]){

        mibol=setdiff(c(1,2,3), kiserlet[i])

    }else{

        mibol=setdiff(c(1,2,3), c(kiserlet[i], jatekos[i]))

    }

    musorvezeto[i] = mibol[sample(1:length(mibol),1)]

}

nemvaltoztatesnyer= which(kiserlet==jatekos)
valtoztat=vector(length = kiserletek_szama)

for (i in 1:kiserletek_szama) {

    holvalt = setdiff(c(1,2,3), c(musorvezeto[i], jatekos[i]))
    valtoztat[i] = holvalt[sample(1:length(holvalt),1)]

}

valtoztatesnyer = which(kiserlet==valtoztat)


sprintf("Kiserletek szama: %i", kiserletek_szama)
length(nemvaltoztatesnyer)
length(valtoztatesnyer)
```

```
length(nemvaltoztatesnyer)/length(valtoztatesnyer)
length(nemvaltoztatesnyer)+length(valtoztatesnyer)
```

A Monty Hall probléma keretében arról van szó, hogy adott három lehetőség (doboz, ajtó stb.), ebből egy nyerő és kettő nem. Nekünk a feladatunk hogy ezekből válasszunk egyet. Ilyenkor az esély hogy a nyerő dobozt választjuk egy a háromból. Választásunk után egy második fél kinyit a maradék két dobozból egy olyat, ami biztosan nem nyerő, majd felteszi nekünk a kérdést, hogy maradunk-e az előző döntésünknél, vagy változtatunk rajta. Ekkor a helyes válasz az, hogy változtassunk, ugyanis így 2/3 esélyünk van ara, hogy jó dobozt választunk szemben az előző 1/3-al.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

```
#include <stdio.h>

int main()
{
  int szam;
  scanf("%d",&szam);
  for(szam; szam != 0; --szam)
  {
    printf("|");
  }
  printf("\n");
  return 0;
}
```

Ez a program tizes számrendszerű számokat vált egyes számrendszerbe.

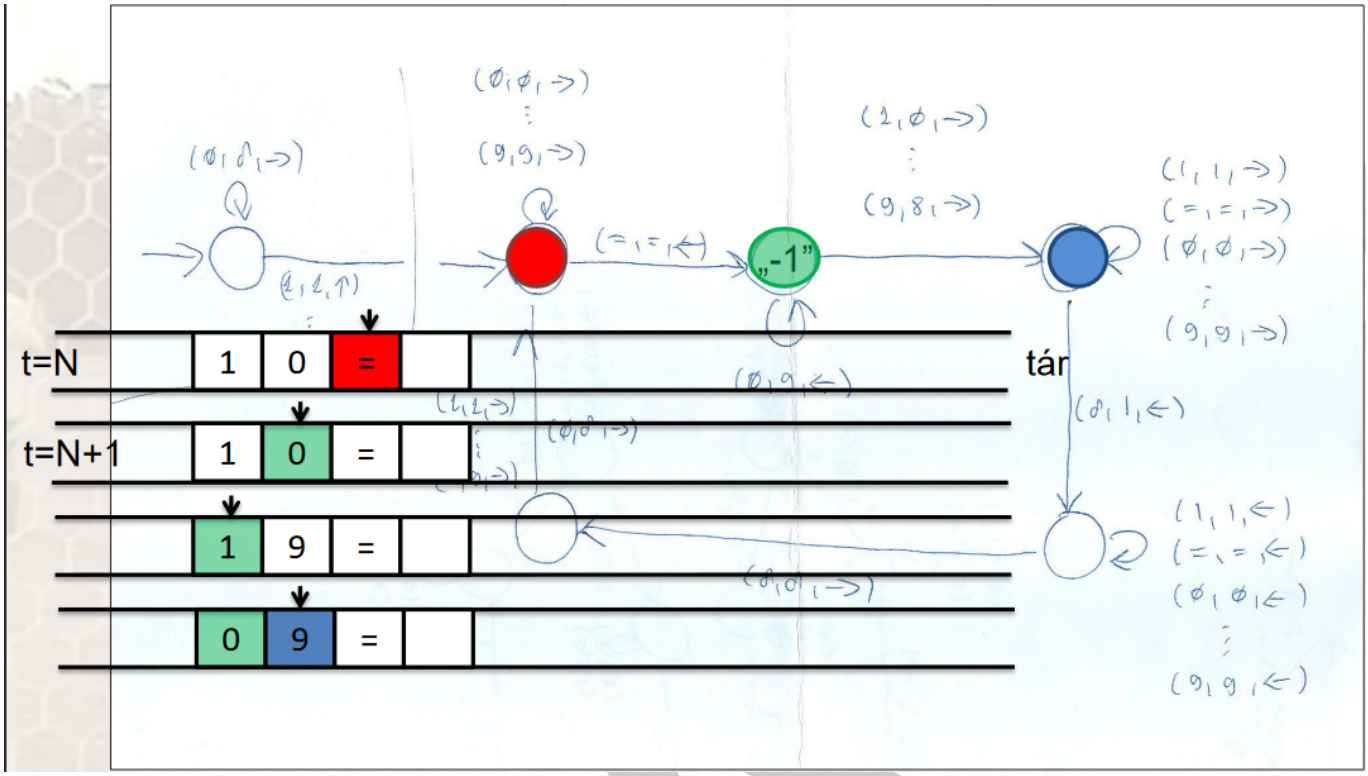
Deklarálunk egy integert, amit beszkennelünk a

```
scanf(string, &input)
```

függvény segítségével. A scanf() hasonlóan a printf() függvényhez két, vagy több argumentummal használandó, ahol az első egy karakterlánc, a többi pedig a stringben található "%" + betű kombinációjú jelek helyettesítési értéke.

A program következő részében egy for ciklus található, ahol a bekért szám értékét csökkentjük egyenként nulláig úgy, hogy minden egyes csökkentés után kiírunk egy "|" jelet. Ezzel elérjük, hogy lényegében átváltunk tizes számrendszerből egyes vagy unáris számrendszerbe.

A program állapotmenetgráfja:



3.1. ábra. Sample

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

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

Legenyek: S, X, Y változók, a,b,c konstansok

Legyenek megadva a következő következtetési szabályok: S -> abc, S -> aXbc, Xb -> bX, Xc -> Ybcc, bY -> Yb, aY -> aaX, aY -> aa

Ekkor a helyettesítéi szabályok alkalmazásával eljutunk $a^n b^n c^n$ -hez, tehát mindig ugyanannyi a, b és c változóink lesznek. Például induljunk ki S-ből:

```
S (S -> aXbc)
aXbc (Xb -> bX)
abXc (Xc -> Ybcc)
abYbcc (bY -> Yb)
aYbbcc (aY -> aa)
aabbcc
```

Egy másik ilyen nyelv: Legyenek A,B,C változók, a,b,c konstansok és legyenek megadva a következő helyettesítési szabályok: A -> aAB, A -> aC, CB -> bCc, cB -> Bc, C -> bc Most induljunk ki A-ból:

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

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.

```
      #include <stdio.h>
// compiles in C99, not in C89
// -std=C99

int main()
{
int n = 5;
for (int i = 0; i < n; ++i) {};
}
```

A fenti kód a C nyelv C99-es és újabb verzióiban lefordul, de azelőttiben, mint a C89 nem, mert a C99-es verzió megjelenése előtt a for ciklusok fejlécében nem lehetett változót inicializálni. Adott verzióval úgy lehet programot fordítani, hogy ha (gcc és C99-es verzió esetén) a következőképpen adjuk meg az utasítást: gcc programnev.c -std=C99

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!

```
      %{
      #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 fenti program megszámolja az input stringben lévő valós számokat.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

```
/*
Forditas:
$ lex -o l337d1c7.c l337d1c7.l

Futtatas:
$ gcc l337d1c7.c -o l337d1c7 -lfl
(kilépés az input vége, azaz Ctrl+D)

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 <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <ctype.h>

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

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

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

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

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

%}
%%
. {
```

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


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

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

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

      found = 1;
      break;
    }

  }

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

}
%%
int
main()
{
srand(time(NULL)+getpid());
yylex();
  return 0;
}
```

Ez egy lexer által írt program, ami ha lefut, az input stringben betűnkénti helyettesítést végez. A program elején egy struktúra van megadva, ami lehetővé teszi a saját típusunk deklarálását. Itt a megadott cipher típushoz tartozik egy c karakter és egy leet karakterre mutató mutatók négyelemű tömbje. Ezután inicializáljuk a cipher típusú l337d1c7 tömbünket. Ezzel véget ér a definícók része.

A program következő részében egy for ciklus található, ami végigmegy a l337d1c7 tömbön és megkeresi a betűt, amit inputba megkapott a program, majd generál egy véletlenszerű számot 1-től 100-ig és az eredménytől függően kiválasztja az adott c karakterhez tartozó leet helyettesítési értéket.

Lefordítani a "lex -o l337d1c7.c l337d1c7.l" utasítással kell, ahol l337d1c7.c a forrásfájlunk, l337d1c7.l oedig a lexer által írt programunk. A futtatás a "gcc l337d1c7.c -o l337d1c7 -lfl" paranccsal történik.

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);
```

```
        for(i=0; i<5; ++i) /* Legyen i egyenlő 0, ha i kisebb 5-nél ↩
           akkor inkrementálja egyel, majd nézze meg i értékét újra.
    Mivel itt i nincs deklarálva, ezért ez a program nem fog lefutni. */
    for(i=0; i<5; i++) /* Legyen i egyenlő 0, ha i kisebb 5-nél akkor ↩
       nézze meg i értékét újra, majd inkrementálja egyel.
    Mivel itt i nincs deklarálva, ezért ez a program nem fog lefutni. */
    for(i=0; i<5; tomb[i] = i++) /* Legyen i egyenlő 0, ha i kisebb 5-nél ↩
        a tomb nevű tömb i-edik eleme legyen egyenlő
    i-vel, inkrementálja i-t, majd folytassa a for ciklust.
    Mivel itt i nincs deklarálva, ezért ez a program nem fog lefutni. */
```

```
    for(i=0; i<n && (*d++ = *s++); ++i); /* Legyen i egyenlő 0, ha i ↩
       kisebb n-nél és a s mutató egyenlő az s mutatóval,
    inkrementálja őket, majd i-t is inkrementálja és folytassa a for ↩
       ciklust.
    Mivel itt i nincs deklarálva, ezért ez a program nem fog lefutni. */
    printf("%d %d", f(a, ++a), f(++a, a)); /* a printf() függvény kiírja ↩
       az f(a, ++a) és f(++a, a) értékeket egymás mellé,
    ahol f() egy két paraméterű függvény.
    Mivel f() függvény mnincs deklarálva ezért ez a program nem fog ↩
       lefutni. */
    printf("%d %d", f(a), a); /* a printf() függvény kiírja f(a) és a ↩
       értékeket egymás mellé, ahoz az f() egy egy paraméterű
    függvény. */
    printf("%d %d", f(&a), a); /* a printf() függvény kiírja f(&a) és a ↩
       értékeket ahol az f() egy egy paraméterű függvény, &a
    pedig az a változó címe.
    Az utolsó 3 sor kód magában nem fog lefutni, mert sem az f() függvény ↩
        sem a változó nincs deklarálva. */
```

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)) $

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

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

Bármely x-re létezik olyan y, hogy x kisebb mint y vagy x egy prímszám

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

Bármely x-re létezik olyan y, hogy x kisebb mint y vagy y prím vagy y + 2 prím

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

Létezik olyan y, hogy bármely x vagy prím, vagy kisebb mint y

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

Létezik olyan y, hogy bármely x-re igaz, hogy y kisebb mint x, amiből következik, hogy x nem prím

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

- ```
  int a;
  ```
- ```
  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);
  ```

```
#include <iostream>
int main()
{
  int a; //Ez egy egész szám típusú változó.
  int *b = &a;  //Ez egy a egész számra mutató mutató. Értéke a referenciája.
  int &r = a;  //Ez egy egész szám referenciája. Értéke a.
  int c[5];  //Ez egy 5 elemű egészek tömbje.
  int (&tr)[5] = c;  //Ez egy 5 elemű egészek tömbjének a referenciája.
  int *d[5];  //Ez egy egészre mutató mutatók tömbje.
  int *h ();  //Ez egy egészre mutató mutatót visszaadó függvény
  int *(*l) ();  //Ez egy egészre mutató mutatót visszaadó függvényre mutató mutató
  int (*v (int c)) (int a, int b);  //Ez egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény
  int (*(*z) (int)) (int, int);  //Ez egy függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre
}
```
```

4. fejezet

Helló, Caesar!

4.1. int ** háromszögmátrix

Megoldás videó:

```
#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;
}
```

Ez a program a malloc() és a free() függvényeket felhasználva helyet foglal egy alsó háromszög mátrixnak a szabad tárban, majd szabaddá teszi azt a helyet. A malloc() függvény feladata a megadott számú bájtoknak való memóriablokk lefoglalása. Visszatérítési értéke egy void típusú pointer. Ebben a programban két malloc() függvény van, mindkettő egy egy if függvényben. Ez azért van, hogyha az allokáció sikertelen, tehát nullpointer a visszatérési értéke, akkor a program befejeződjön. A malloc() által foglalt memóriaterület magától nem fog felszabadulni, ezért ezt a program végén nekünk kell megtenni a free() függvénnyel.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

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

#define MAX_KULCS 100
#define BUFFER_MERET 256

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

```
        char kulcs[MAX_KULCS];
        char buffer[BUFFER_MERET];

        int kulcs_index = 0;
        int olvasott_bajtok = 0;

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

        while ((olvasott_bajtok = read (0, (void *) buffer, BUFFER_MERET)))
            {
            for (int i = 0; i < olvasott_bajtok; ++i)
          {

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

          }

            write (1, buffer, olvasott_bajtok);

            }
        }
```

A XOR kódoló mögötti alapötlet, hogy a fájl bájtjait össze XOR-ozzuk a kulcs bájtjaival. Ekkor az eredmény egy titkosított fájl, aminek az eredetijét visszakaphatjuk, ha az eredményt ismét össze XOR-ozzuk a kulccsal. A XOR jelentése kizáró vagy, elvégezni az ^ operátorral lehet. A XOR B eredménye csak akkor igaz, ha vagy csak A, vagy csak B igaz. Minden más esetben hamis. Tehát:

```
    1 ^ 1 = 0
    0 ^ 1 = 1
    1 ^ 0 = 1
    0 ^ 0 = 0
```

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

```
public class ExorTitkosító {

    public ExorTitkosító(String kulcsSzöveg,
            java.io.InputStream bejövőCsatorna,
            java.io.OutputStream kimenőCsatorna)
            throws java.io.IOException {
```

```
        byte [] kulcs = kulcsSzöveg.getBytes();
        byte [] buffer = new byte[256];
        int kulcsIndex = 0;
        int olvasottBájtok = 0;

        while((olvasottBájtok =
                bejövőCsatorna.read(buffer)) != -1) {

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

                buffer[i] = (byte)(buffer[i] ^ kulcs[kulcsIndex]);
                kulcsIndex = (kulcsIndex+1) % kulcs.length;

            }

            kimenőCsatorna.write(buffer, 0, olvasottBájtok);

        }

    }

    public static void main(String[] args) {

        try {

            new ExorTitkosító(args[0], System.in, System.out);

        } catch(java.io.IOException e) {

            e.printStackTrace();

        }

    }

}
```

Ez az előző feladatnak a Java nyelven megvalósított változata.

A legnagyobb különbség, hogy mivel a C-vel ellentétben a Java egy objektum orientált nyelv, mindent osztályokba osztunk. Itt két osztályunk van. Ezek az ExorTitkosító és a main. Az ExorTitkosító osztályon belül megadunk egy ugyanolyan nevű objektumot, amely kaphat három argumentumot, a kulcsszöveget és a bemeneti illetve a kimeneti csatornát.

Mivel a Java-ban létezik külön byte típus, ezért mivel itt bájtokkal dolgozunk, itt felhasználjuk.

Ezután megkezdődik a while-ba ágyazott for ciklus segítségével a kódtörés, majd a kimenő csatornára írja ki az eredményt.

A main függvényen belül megadjuk, hogy az első parancssori argumentum legyen a kulcsszöveg, a bemeneti csatorna a System.in és kimeneti a System.out.

4.4. C EXOR törő

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

A következő program megtöri a titkosított fájlunkat.

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

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

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

    return (double) titkos_meret / sz;
}

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

    double szohossz = atlagos_szohossz (titkos, titkos_meret);

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

}

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

    int kulcs_index = 0;

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

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

    }

}

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

    exor (kulcs, kulcs_meret, titkos, titkos_meret);

    return tiszta_lehet (titkos, titkos_meret);

}

int
main (void)
{

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

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

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

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

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

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

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

    return 0;
}
```

4.5. Neurális OR, AND és EXOR kapu

R

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

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

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

```
#   You should have received a copy of the GNU General Public License
#   along with this program.  If not, see <http://www.gnu.org/licenses/>
#
# https://youtu.be/Koyw6IH5ScQ

library(neuralnet)

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

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

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

plot(nn.or)

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


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

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

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

plot(nn.orand)

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



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

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

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

plot(nn.exor)

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

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

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

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

plot(nn.exor)

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

Egy neurális hálózat neurális "sejtekből" álló hálózat.

Ezeknek a sejteknek megvan a matematikai modelljük. Minden ilyen sejtnek megvannak a bemeneti és kimeneti kapcsolatai. Egy bemeneti függvény, egy aktivációs függvény és a kimenet.

A bemeneti kapcsolatokon keresztül érkező adatoknak mindnek megvan a maga súlya.

A bemeneti függvényen keresztül kiszámoljuk a beérkező adatok súlyozott átlagát.

Ezután jön az aktivációs függvény, ami a súlyozott összeget egy a 0 és az 1 közé eső értékké fogja átváltani.

majd a kimenetet továbbadja a kimeneti kapcsolatain keresztül, amelyek további "sejtekhez" vannak kapcsolódva.

A program maga három részből áll. Ezek sorban az OR (vagy), OR és mellé AND (és), és az XOR (kizáró vagy) logikai kapukat írják le.

A program először betölti a neuralnet nevű könyvtárat, majd például az OR esetén feltölti az a1(0,1,0,1) és a2(0,0,1,1,) változókat, illetve megadjuk, hogy ezekre az értékekre milyen hatással lenne az OR művelet elvégzése. A műveletet fentről lefelé végezzük el. Tehát 0 OR 0, 1 OR 0, 0 OR 1 és 1 OR 1. Így megkapjuk az OR(0,1,1,1) változót. Ezzel megadjuk a hálónak a szabályokat és ezután elkezdi tanítani magát, kiszámolja magának a súlyokat.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

```
class Perceptron
{
public:
```

```
  Perceptron ( int nof, ... )
  {
    n_layers = nof;

    units = new double*[n_layers];
    n_units = new int[n_layers];

    va_list vap;

    va_start ( vap, nof );

    for ( int i {0}; i < n_layers; ++i )
      {
        n_units[i] = va_arg ( vap, int );

        if ( i )
          units[i] = new double [n_units[i]];
      }

    va_end ( vap );

    weights = new double**[n_layers-1];

#ifndef RND_DEBUG
    std::random_device init;
    std::default_random_engine gen {init() };
#else
    std::default_random_engine gen;
#endif

    std::uniform_real_distribution<double> dist ( -1.0, 1.0 );

    for ( int i {1}; i < n_layers; ++i )
      {
        weights[i-1] = new double *[n_units[i]];

        for ( int j {0}; j < n_units[i]; ++j )
          {
            weights[i-1][j] = new double [n_units[i-1]];

            for ( int k {0}; k < n_units[i-1]; ++k )
              {
                weights[i-1][j][k] = dist ( gen );
              }
          }
      }
  }

  Perceptron ( std::fstream & file )
  {
```

```
    file >> n_layers;

    units = new double*[n_layers];
    n_units = new int[n_layers];

    for ( int i {0}; i < n_layers; ++i )
      {
        file >> n_units[i];

        if ( i )
          units[i] = new double [n_units[i]];
      }

    weights = new double**[n_layers-1];

    for ( int i {1}; i < n_layers; ++i )
      {
        weights[i-1] = new double *[n_units[i]];

        for ( int j {0}; j < n_units[i]; ++j )
          {
            weights[i-1][j] = new double [n_units[i-1]];

            for ( int k {0}; k < n_units[i-1]; ++k )
              {
                file >> weights[i-1][j][k];
              }
          }
      }
  }


  double sigmoid ( double x )
  {
    return 1.0/ ( 1.0 + exp ( -x ) );
  }


  double operator() ( double image [] )
  {

    units[0] = image;

    for ( int i {1}; i < n_layers; ++i )
      {

#ifdef CUDA_PRCPS

        cuda_layer ( i, n_units, units, weights );
```

```
#else

        #pragma omp parallel for
        for ( int j = 0; j < n_units[i]; ++j )
          {
            units[i][j] = 0.0;

            for ( int k = 0; k < n_units[i-1]; ++k )
              {
                units[i][j] += weights[i-1][j][k] * units[i-1][k];
              }

            units[i][j] = sigmoid ( units[i][j] );

          }

#endif

      }

    return sigmoid ( units[n_layers - 1][0] );

  }

  void learning ( double image [], double q, double prev_q )
  {
    double y[1] {q};

    learning ( image, y );
  }

  void learning ( double image [], double y[] )
  {
    //( *this ) ( image );

    units[0] = image;

    double ** backs = new double*[n_layers-1];

    for ( int i {0}; i < n_layers-1; ++i )
      {
        backs[i] = new double [n_units[i+1]];
      }

    int i {n_layers-1};

    for ( int j {0}; j < n_units[i]; ++j )
      {
        backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ( units[i][ ↩
           j] ) ) * ( y[j] - units[i][j] );
```

```
      for ( int k {0}; k < n_units[i-1]; ++k )
        {
          weights[i-1][j][k] += ( 0.2* backs[i-1][j] *units[i-1][k] );
        }

    }

  for ( int i {n_layers-2}; i >0 ; --i )
    {

      #pragma omp parallel for
      for ( int j =0; j < n_units[i]; ++j )
        {

          double sum = 0.0;

          for ( int l = 0; l < n_units[i+1]; ++l )
            {
              sum += 0.19*weights[i][l][j]*backs[i][l];
            }

          backs[i-1][j] = sigmoid ( units[i][j] ) * ( 1.0-sigmoid ( units ↩
             [i][j] ) ) * sum;

          for ( int k = 0; k < n_units[i-1]; ++k )
            {
              weights[i-1][j][k] += ( 0.19* backs[i-1][j] *units[i-1][k]  ↩
                 );
            }
        }
    }

  for ( int i {0}; i < n_layers-1; ++i )
    {
      delete [] backs[i];
    }

  delete [] backs;

}

~Perceptron()
{
  for ( int i {1}; i < n_layers; ++i )
    {
      for ( int j {0}; j < n_units[i]; ++j )
        {
          delete [] weights[i-1][j];
        }
```

```
        delete [] weights[i-1];
      }

    delete [] weights;

    for ( int i {0}; i < n_layers; ++i )
      {
        if ( i )
          delete [] units[i];
      }

    delete [] units;
    delete [] n_units;

  }

  void save ( std::fstream & out )
  {
    out << " "
        << n_layers;

    for ( int i {0}; i < n_layers; ++i )
      out << " " << n_units[i];

    for ( int i {1}; i < n_layers; ++i )
      {
        for ( int j {0}; j < n_units[i]; ++j )
          {
            for ( int k {0}; k < n_units[i-1]; ++k )
              {
                out << " "
                    << weights[i-1][j][k];

              }
          }
      }

  }

private:
  Perceptron ( const Perceptron & );
  Perceptron & operator= ( const Perceptron & );

  int n_layers;
  int* n_units;
  double **units;
  double ***weights;

};
```

A fenti osztály egy hiba-visszaterjesztéses perceptron definíciója. Az így definiált perceptronnak helyet foglalhatunk úgy, hogy Perceptron* p = new Perceptron(int layer, int neuron1, int neuron2, ..., int neuronN) Ahol a layer a rétegek számát jelöli és az azutáni egész számok pedig az egyes rétegekhez rendelt neuronok számát.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

```
// mandelpngt.c++
// 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/>.
//
// Version history
//
//  Mandelbrot png
//  Programozó Páternoszter/PARP
//   https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↩
   _01_parhuzamos_prog_linux
//
//  https://youtu.be/gvaqijHlRUs
//
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
```

```
#define ITER_HAT 32000

void
mandel (int kepadat[MERET][MERET]) {

    // Mérünk időt (PP 64)
    clock_t delta = clock ();
    // Mérünk időt (PP 66)
    struct tms tmsbuf1, tmsbuf2;
    times (&tmsbuf1);

    // számítás adatai
    float a = -2.0, b = .7, c = -1.35, d = 1.35;
    int szelesseg = MERET, magassag = MERET, iteraciosHatar = ITER_HAT;

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
            // c = (reC, imC) a rács csomópontjainak
            // megfelelő komplex szám
            reC = a + k * dx;
            imC = d - j * dy;
            // z_0 = 0 = (reZ, imZ)
            reZ = 0;
            imZ = 0;
            iteracio = 0;
            // z_{n+1} = z_n * z_n + c iterációk
            // számítása, amíg |z_n| < 2 vagy még
            // nem értük el a 255 iterációt, ha
            // viszont elértük, akkor úgy vesszük,
            // hogy a kiinduláci c komplex számra
            // az iteráció konvergens, azaz a c a
            // Mandelbrot halmaz eleme
            while (reZ * reZ + imZ * imZ < 4 && iteracio < iteraciosHatar)
            {
                // z_{n+1} = z_n * z_n + c
                ujreZ = reZ * reZ - imZ * imZ + reC;
                ujimZ = 2 * reZ * imZ + imC;
                reZ = ujreZ;
                imZ = ujimZ;
```

```
                ++iteracio;

            }

            kepadat[j][k] = iteracio;
        }
    }

    times (&tmsbuf2);
    std::cout << tmsbuf2.tms_utime - tmsbuf1.tms_utime
              + tmsbuf2.tms_stime - tmsbuf1.tms_stime << std::endl;

    delta = clock () - delta;
    std::cout << (float) delta / CLOCKS_PER_SEC << " sec" << std::endl;

}

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

    if (argc != 2)
    {
        std::cout << "Hasznalat: ./mandelpng fajlnev";
        return -1;
    }

    int kepadat[MERET][MERET];

    mandel(kepadat);

    png::image < png::rgb_pixel > kep (MERET, MERET);

    for (int j = 0; j < MERET; ++j)
    {
        //sor = j;
        for (int k = 0; k < MERET; ++k)
        {
            kep.set_pixel (k, j,
                           png::rgb_pixel (255 -
                                           (255 * kepadat[j][k]) / ITER_HAT ↩
                                              ,
                                           255 -
                                           (255 * kepadat[j][k]) / ITER_HAT ↩
                                              ,
                                           255 -
                                           (255 * kepadat[j][k]) / ITER_HAT ↩
                                              ));
        }
    }
```

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

}
```

A Mandelbrot halmaz egy nevezetes alakzat a komplex számsíkon. A komplex számok azon számok, amelyek az a+bi alakot veszik fel, ahol i a -1 gyöke. Az i-t nem értelmezzük, de a segítségével megkaphatjuk a negatív számok gyökeit. Példaként a -2 gyöke egyenlő a kettő gyökének az i-szeresével.

A program kiszámolja, hogy mely értékek tartoznak a mandelbrot halmazba, majd lerajzolja őket. 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.

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

```
// 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  ↩
   -0.019473810573093663922605855987058021128 18  ↩
   -0.019473810572541341845642648422654019668 7  ↩
   0.798505756933826860155534177465597167611 1  ↩
   0.79850575693437919611028519284445792436 6
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913 1  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327097512154281  ↩
   0.213538705180497528912653137922461610287 4
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
```

```
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.


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

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

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

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

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

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

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

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

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

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

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

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

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

              ++iteracio;
            }

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

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

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

}
```

A fenti program ugyanazt csinálja, mint az előző, de felhasználja az std::complex osztályt.

Az std::complex osztály egy az complex.h fájlban definiált osztály amit a komplex számok reprezentálására és az azokkak való műveletek elvégzésére találtak ki. A komplex osztállyal való deklaráció a következőképpen néz ki:

```
std::complex<double> z_1 = (0,0)
```

Amivel például a $z_1 = 0 + 0*i$ komplex szám van megadva.

Az osztályon belül definiálva vannak a komplex osztállyal végzett műveletek, mint az összeadás, kivonás, osztás és szorzás, vagy egyéb a komplex számokkal kapcsolatos műveletek, mint a valós vagy a képzeletbeli rész visszadadása, az abszolútérték megadása. Ezeket úgy lehet felhasználni, mint más nem komplex számokkal, például, ha definiáltuk a z_1 és z_2 komplex számokat elég azt leírnunk, hogy z_1 + z_2 . Egy z nevű komplex szám valós részének a kiírása:

```
std::cout << std::real(z)
```

5.3. Biomorfok

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

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

```
// 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
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// 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;

}
```

A fenti program Julia halmazt ábrázol. A különbség a Jula és a Mandelbrot halmazok között, hogy az előbbiben a c változó, a befutott rácson egy pont, itt viszont egy állandó

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.

A JDK (Java Development Kit) forrásaiban a Sun programozói is pontosan így csinálták meg, mint ahogy itt van.

```
#include "polargen.h"

double
PolarGen::kovetkezo ()
{
  if (nincsTarolt)
    {
      double u1, u2, v1, v2, w;
      do
  {
    u1 = std::rand () / (RAND_MAX + 1.0);
    u2 = std::rand () / (RAND_MAX + 1.0);
    v1 = 2 * u1 - 1;
    v2 = 2 * u2 - 1;
    w = v1 * v1 + v2 * v2;
  }
      while (w > 1);

      double r = std::sqrt ((-2 * std::log (w)) / w);

      tarolt = r * v2;
      nincsTarolt = !nincsTarolt;

      return r * v1;
```

```
    }
  else
    {
      nincsTarolt = !nincsTarolt;
      return tarolt;
    }
}
```

Egy számítási lépés két normális eloszlású számot állít elő, így csak a minden páros számű meghíváskor kell számolnunk. Azt, hogy mikor van a páratlanadik illetve a párosadik meghívás, a nincsTarolt nevű változó mondja meg.

Itt a polargen.h fájlban inicializált kovetkezo() függvény van definiálva.

A polargen.h includált fájl tartalma egy PolarGen nevű osztályt ír le. Két privát változót, egy és 3 publikus függvény tartozik bele, amelyek közül a ~PolarGen a destruktor, a PolarGen() pedig a konstruktor.

```
#ifndef POLARGEN__H
#define POLARGEN__H

#include <cstdlib>
#include <cmath>
#include <ctime>

class PolarGen
{
public:
  PolarGen ()
  {
    nincsTarolt = true;
    std::srand (std::time (NULL));
  }
   ~PolarGen ()
  {
  }
  double kovetkezo ();

private:
  bool nincsTarolt;
  double tarolt;

};

#endif
```

6.2. LZW

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

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

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

} BINFA, *BINFA_PTR;

BINFA_PTR
uj_elem ()
{
  BINFA_PTR p;

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

extern void kiir (BINFA_PTR elem);
extern void szabadit (BINFA_PTR elem);

int
main (int argc, char **argv)
{
  char b;

  BINFA_PTR gyoker = uj_elem ();
  gyoker->ertek = '/';
  BINFA_PTR fa = gyoker;

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

```
        fa->bal_nulla->ertek = 0;
        fa->bal_nulla->bal_nulla = fa->bal_nulla->jobb_egy = NULL;
        fa = gyoker;
      }
    else
      {
        fa = fa->bal_nulla;
      }
  }
      else
  {
    if (fa->jobb_egy == NULL)
      {
        fa->jobb_egy = uj_elem ();
        fa->jobb_egy->ertek = 1;
        fa->jobb_egy->bal_nulla = fa->jobb_egy->jobb_egy = NULL;
        fa = gyoker;
      }
    else
      {
        fa = fa->jobb_egy;
      }
  }
    }

  printf ("\n");
  kiir (gyoker);
  extern int max_melyseg;
  printf ("melyseg=%d", max_melyseg);
  szabadit (gyoker);
}

static int melyseg = 0;
int max_melyseg = 0;

void
kiir (BINFA_PTR elem)
{
  if (elem != NULL)
    {
      ++melyseg;
      if (melyseg > max_melyseg)
  max_melyseg = melyseg;
      kiir (elem->jobb_egy);
      for (int i = 0; i < melyseg; ++i)
  printf ("---");
      printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem->ertek ↩
         ,
        melyseg);
      kiir (elem->bal_nulla);
```

```
      --melyseg;
    }
}

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

A program a következőt csinálja: Ha van egy 00011101110 sorozatom, akkor először felírjuk, a gyökeret (/), majd elkezdjük "szétszedni" a számsorunkat. Ha az új szám 0, akkor mindig balra írom, ha 1, akkor jobbra. Az első számjegy 0, nullám még nincs, tehát felírom. A második is 0, de az már van, ezért továbbmegyek, de megjegyzem magamnak a 0-át. A következő 0, és így a 00-nál tartunk, ami még nincs. Felírom, majd továbmegyek. A következő számjegy 1, ami szintén nincs, tehát felírom. Utána ismét 1, ami már van, továbbmegyek: 11, felírom. És így tovább kapjuk a 01, 110 számokat.

Az így kapott eredmény szemléltetése:

```
                /
       0              1
     0   1              1
                       0
```

6.3. Fabejárás

Egy fa posztorder bejárása:

A Postorder fabejárás lényege, hogy először a bal ágakat járjuk be, majd a jobb és végül a gyökeret írjuk ki. A fenti 00011101110 példa alapján a preorder bejárás eredménye: 0, 1, 0, /, 0, 1, 1

```
void printPostorder(BINFA_PTR elem)
{
     if (elem == NULL)
        return;


     printPostorder(elem->ball_nulla);


     printPostorder(elem->jobb_egy);
```

```
     printf("%d ", elem->ertek);
}
```

Preorder:

A Preorder fabejárás lényege, hogy először a gyökeret íratjuk ki, majd a bal ágat és végül a jobb ágat. A fenti 00011101110 példa alapján a preorder bejárás eredménye: /, 0, 0, 1, 1, 1, 0

```
void printPreorder(BINFA_PTR elem)
{
     if (elem == NULL)
          return;


     printf("%d ", elem->ertek);

     printPreorder(elem->ball_nulla);

     printPreorder(elem->jobb_egy);
}
```

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!

A következő program az előző C++-ban megvalósított változata. A Csomopont felel meg a feladat Node-jának illetve LZWBinfa a Tree-nek.

```
// z3a2.cpp
//
// Együtt támadjuk meg: http://progpater.blog.hu/2011/04/14/ ↩
   egyutt_tamadjuk_meg
// LZW fa építő 3. C++ átirata a C valtozatbol (+mélység, atlag és szórás)
// Programozó Páternoszter
//
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail ↩
   .com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
```

```
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthetõ illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszõleges) késõbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1, http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2, csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3, http://progpater.blog.hu/2011/03/05/ ↩
   labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// 0.0.4, z.cpp: a C verzióból svn: bevezetes/C/ziv/z.c átírjuk C++-ra
//        http://progpater.blog.hu/2011/03/31/ ↩
   imadni_fogjatok_a_c_t_egy_emberkent_tiszta_szivbol
// 0.0.5,   z2.cpp: az fgv(*mut)-ok helyett fgv(&ref)
// 0.0.6,   z3.cpp: Csomopont beágyazva
//          http://progpater.blog.hu/2011/04/01/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2
// 0.0.6.1  z3a2.c: LZWBinFa már nem barátja a Csomopont-nak, mert annak  ↩
   tagjait nem használja direktben
// 0.0.6.2  Kis kommentezést teszünk bele 1. lépésként (hogy a kicsit  ↩
   lemaradt hallgatóknak is
//    könnyebb legyen, jól megtűzdeljük további olvasmányokkal)
//    http://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg
//    (majd a 2. lépésben "beletesszük a d.c-t", majd s 3. lépésben a  ↩
   parancssorsor argok feldolgozását)

#include <iostream> // mert olvassuk a std::cin, írjuk a std::cout  ↩
   csatornákat
#include <cmath> // mert vonunk gyököt a szóráshoz: std::sqrt
#include <fstream> // fájlból olvasunk, írunk majd
```

```
/* Az LZWBinFa osztályban absztraháljuk az LZW algoritmus bináris fa építését. Az osztály
 definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ez lesz a
 beágyazott Csomopont osztály. Miért ágyazzuk be? Mert külön nem szánunk neki szerepet, ezzel
 is jelezzük, hogy csak a fa részeként számiolunk vele.*/

class LZWBinFa
{
public:
    /* Szemben a bináris keresőfánkkal (BinFa osztály)
     http://progpater.blog.hu/2011/04/12/imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3
     itt (LZWBinFa osztály) a fa gyökere nem pointer, hanem a '/' betüt tartalmazó objektum,
     lásd majd a védett tagok között lent: Csomopont gyoker;
     A fa viszont már pointer, mindig az épülő LZW-fánk azon csomópontjára mutat, amit az
     input feldolgozása során az LZW algoritmus logikája diktál:
     http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
     Ez a konstruktor annyit csinál, hogy a fa mutatót ráállítja a gyökérre . (Mert ugye
     labopon, blogon, előadásban tisztáztuk, hogy a tartalmazott tagok, most "Csomopont gyoker"
     konstruktora előbb lefut, mint a tagot tartalmazó LZWBinFa osztály konstruktora, éppen a
     következő, azaz a fa=&gyoker OK.)
     */
    LZWBinFa (): fa(&gyoker) {}

    /* Tagfüggvényként túlterheljük a << operátort, ezzel a célunk, hogy felkeltsük a
     hallgató érdeklődését, mert ekkor így nyomhatjuk a fába az inputot: binFa << b; ahol a b
     egy '0' vagy '1'-es betű.
     Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett paraméterként"
     kapott ) példány, azaz annak a fának amibe éppen be akarjuk nyomni a b betűt a tagjai
     (pl.: "fa", "gyoker") használhatóak a függvényben.

     A függvénybe programoztuk az LZW fa építésének algoritmusát tk.:
     http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

     a b formális param az a betű, amit éppen be kell nyomni a fába: */
    void operator<<(char b)
    {
        // Mit kell betenni éppen, '0'-t?
        if (b == '0')
```

```
        {
            /* Van '0'-s gyermeke az aktuális csomópontnak?
             megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
            if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
            {
                // elkészítjük, azaz példányosítunk a '0' betű akt. ↩
                   parammal
                Csomopont *uj = new Csomopont ('0');
                // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
                // jegyezze már be magának, hogy nullás gyereke mostantól ↩
                   van
                // küldjük is Neki a gyerek címét:
                fa->ujNullasGyermek (uj);
                // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
                fa = &gyoker;
            }
            else // ha van, arra rálépünk
            {
                // azaz a "fa" pointer már majd a szóban forgó gyermekre ↩
                   mutat:
                fa = fa->nullasGyermek ();
            }
        }
        // Mit kell betenni éppen, vagy '1'-et?
        else
        {
            if (!fa->egyesGyermek ())
            {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }
    /* A bejárással kapcsolatos függvényeink (túlterhelt kiir-ók, atlag, ↩
       ratlag stb.) rekurzívak,
     tk. a rekurzív fabejárást valósítják meg (lásd a 3. előadás "Fabejárás ↩
       " c. fóliáját és társait)

     (Ha a rekurzív függvénnyel általában gondod van => K&R könyv megfelel ↩
       ő része: a 3. ea. izometrikus
     részében ezt "letáncoltuk" :) és külön idéztük a K&R álláspontját :)
     */
    void kiir (void)
    {
        // Sokkal elegánsabb lenne (és más, a bevezetésben nem kibontandó ↩
```

```
            reentráns kérdések miatt is, mert
        // ugye ha most két helyről hívják meg az objektum ilyen ↩
            függvényeit, tahát ha kétszer kezd futni az
        // objektum kiir() fgv.-e pl., az komoly hiba, mert elromlana a ↩
            mélység... tehát a mostani megoldásunk
        // nem reentráns) ha nem használnánk a C verzióban globális ↩
            változókat, a C++ változatban példánytagot a
        // mélység kezelésére: http://progpater.blog.hu/2011/03/05/ ↩
            there_is_no_spoon
        melyseg = 0;
        // ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát, ↩
             akkor a
        // sztenderd out-ra nyomjuk
        kiir (&gyoker, std::cout);
    }
    void szabadit (void)
    {
        szabadit (gyoker.egyesGyermek());
        szabadit (gyoker.nullasGyermek());
        // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a ↩
            szabad tárban (halmon).
    }

    /* A változatosság kedvéért ezeket az osztálydefiníció (class LZWBinFa ↩
       {...};) után definiáljuk,
     hogy kénytelen légy az LZWBinFa és a :: hatókör operátorral minősítve ↩
        definiálni :) l. lentebb */
    int getMelyseg (void);
    double getAtlag (void);
    double getSzoras (void);

    /* Vágyunk, hogy a felépített LZW fát ki tudjuk nyomni ilyenformán: std ↩
       ::cout << binFa;
     de mivel a << operátor is a sztenderd névtérben van, de a using ↩
        namespace std-t elvből
     nem használjuk bevezető kurzusban, így ez a konstrukció csak az ↩
        argfüggő névfeloldás miatt
     fordul le (B&L könyv 185. o. teteje) ám itt nem az a lényeg, hanem, ↩
        hogy a cout ostream
     osztálybeli, így abban az osztályban kéne módosítani, hogy tudjon ↩
        kiírni LZWBinFa osztálybelieket...
     e helyett a globális << operátort terheljük túl, */
    friend std::ostream& operator<< (std::ostream& os, LZWBinFa& bf)
    {
        bf.kiir(os);
        return os;
    }
    void kiir (std::ostream& os)
    {
        melyseg = 0;
```

```
        kiir (&gyoker, os);
    }

private:
    class Csomopont
    {
    public:
        /* A paraméter nélküli konstruktor az elepértelmezett '/' "gyökér- ↩
           betűvel" hozza
         létre a csomópontot, ilyet hívunk a fából, aki tagként tartalmazza ↩
             a gyökeret.
         Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu"  ↩
            tagba, a két
         gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is  ↩
            megteszi. */
        Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
        ~Csomopont () {};
        // Aktuális csomópont, mondd meg nékem, ki a bal oldali gyermeked
        // (a C verzió logikájával műxik ez is: ha nincs, akkor a null megy ↩
            vissza)
        Csomopont *nullasGyermek () const {
            return balNulla;
        }
        // Aktuális csomópon,t mondd meg nékem, ki a jobb oldali gyermeked?
        Csomopont *egyesGyermek () const {
            return jobbEgy;
        }
        // Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a bal  ↩
           oldali gyereked!
        void ujNullasGyermek (Csomopont * gy) {
            balNulla = gy;
        }
        // Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a jobb ↩
            oldali gyereked!
        void ujEgyesGyermek (Csomopont * gy) {
            jobbEgy = gy;
        }
        // Aktuális csomópont: Te milyen betűt hordozol?
        // (a const kulcsszóval jelezzük, hogy nem bántjuk a példányt)
        char getBetu() const {
            return betu;
        }

    private:
        // friend class LZWBinFa; /* mert ebben a valtozatban az LZWBinFa  ↩
           metódusai nem közvetlenül
        // a Csomopont tagjaival dolgoznak, hanem beállító/lekérdező  ↩
           üzenetekkel érik el azokat */

        // Milyen betűt hordoz a csomópont
```

```
        char betu;
        // Melyik másik csomópont a bal oldali gyermeke? (a C változatból " ↩
           örökölt" logika:
        // ha hincs ilyen csermek, akkor balNulla == null) igaz
        Csomopont *balNulla;
        Csomopont *jobbEgy;
        // nem másolható a csomópont (ökörszabály: ha van valamilye a  ↩
           szabad tárban,
        // letiltjuk a másoló konstruktort, meg a másoló értékadást)
        Csomopont (const Csomopont &);
        Csomopont & operator=(const Csomopont &);
    };

    /* Mindig a fa "LZW algoritmus logikája szerinti aktuális"  ↩
       csomópontjára mutat */
    Csomopont *fa;
    // technikai
    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;
    // szokásosan: nocopyable
    LZWBinFa (const LZWBinFa &);
    LZWBinFa & operator=(const LZWBinFa &);

    /* Kiírja a csomópontot az os csatornára. A rekurzió kapcsán lásd a  ↩
       korábbi K&R-es utalást...*/
    void kiir (Csomopont* elem, std::ostream& os)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió  ↩
           leállítása
        if (elem != NULL)
        {
            ++melyseg;
            kiir (elem->egyesGyermek(), os);
            // ez a postorder bejáráshoz képest
            // 1-el nagyobb mélység, ezért -1
            for (int i = 0; i < melyseg; ++i)
                os << "---";
            os << elem->getBetu() << "(" << melyseg - 1 << ")" << std::endl ↩
               ;
            kiir (elem->nullasGyermek(), os);
            --melyseg;
        }
    }
    void szabadit (Csomopont * elem)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió  ↩
           leállítása
        if (elem != NULL)
        {
            szabadit (elem->egyesGyermek());
```

```
            szabadit (elem->nullasGyermek());
            // ha a csomópont mindkét gyermekét felszabadítottuk
            // azután szabadítjuk magát a csomópontot:
            delete elem;
        }
    }

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

    /* A fában tagként benne van egy csomópont, ez erősen ki van tüntetve, ↩
       Ő a gyökér: */
    Csomopont gyoker;
    int maxMelyseg;
    double atlag, szoras;

    void rmelyseg (Csomopont* elem);
    void ratlag (Csomopont* elem);
    void rszoras  (Csomopont* elem);

};

// Néhány függvényt az osztálydefiníció után definiálunk, hogy lássunk ↩
   ilyet is ... :)
// Nem erőltetjük viszont a külön fájlba szedést, mert a ↩
   sablonosztályosított tovább
// fejlesztésben az linkelési gondot okozna, de ez a téma már kivezet a ↩
   laborteljesítés
// szükséges feladatából: http://progpater.blog.hu/2011/04/12/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3

// Egyébként a melyseg, atlag és szoras fgv.-ek a kiir fgv.-el teljesen egy ↩
    kaptafa.

int LZWBinFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (&gyoker);
    return maxMelyseg-1;
}
double LZWBinFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double)atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBinFa::getSzoras (void)
```

```
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (&gyoker);

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

    return szoras;
}
void LZWBinFa::rmelyseg (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek());
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        rmelyseg (elem->nullasGyermek());
        --melyseg;
    }
}
void
LZWBinFa::ratlag (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyesGyermek());
        ratlag (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}
void
LZWBinFa::rszoras  (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
```

```
        rszoras (elem->egyesGyermek());
        rszoras (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

// teszt pl.: http://progpater.blog.hu/2011/03/05/ ↩
   labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// [norbi@sgu ~]$ echo "01111001001001000111"|./z3a2
// ------------1(3)
// ---------1(2)
// ------1(1)
// ---------0(2)
// ------------0(3)
// ---------------0(4)
// ---/(0)
// ---------1(2)
// ------0(1)
// ---------0(2)
// depth = 4
// mean = 2.75
// var = 0.957427
// a laborvédéshez majd ezt a tesztelést használjuk:
// http://

/* Ez volt eddig a main, de most komplexebb kell, mert explicite bejövő, ↩
   kimenő fájlokkal kell dolgozni
int
main ()
{
    char b;
    LZWBinFa binFa;

    while (std::cin >> b)
    {
        binFa << b;
    }

    //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ↩
       verziókban de, hogy izgalmasabb legyen
    // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

    std::cout << binFa; // ehhez kell a globális operator<< túlterhelése, ↩
       lásd fentebb
```

```
    std::cout << "depth = " << binFa.getMelyseg () << std::endl;
    std::cout << "mean = " << binFa.getAtlag () << std::endl;
    std::cout << "var = " << binFa.getSzoras () << std::endl;

    binFa.szabadit ();

    return 0;
}
*/

/* A parancssor arg. kezelést egyszerűen bedolgozzuk a 2. hullám kapcsolódó ↩
    feladatából:
 http://progpater.blog.hu/2011/03/12/hey_mikey_he_likes_it_ready_for_more_3
 de mivel nekünk sokkal egyszerűbb is elég, alig hagyunk meg belőle valamit ↩
    ...
 */

void usage(void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    // http://progpater.blog.hu/2011/03/12/ ↩
       hey_mikey_he_likes_it_ready_for_more_3
    // alapján a parancssor argok ottani elegáns feldolgozásából kb. ennyi  ↩
       marad:
    // "*((*++argv)+1)"...

    // a kiírás szerint ./lzwtree in_file -o out_file alakra kell mennie,  ↩
       ez 4 db arg:
    if (argc != 4) {
        // ha nem annyit kapott a program, akkor felhomályosítjuk erről a  ↩
           józetr:
        usage();
        // és jelezzük az operációs rendszer felé, hogy valami gáz volt...
        return -1;
    }

    // "Megjegyezzük" a bemenő fájl nevét
    char *inFile = *++argv;

    // a -o kapcsoló jön?
    if (*((*++argv)+1) != 'o')  {
        usage();
        return -2;
    }
```

```
  // ha igen, akkor az 5. előadásból kimásoljuk a fájlkezelés C++ változatát:
  std::fstream beFile (inFile, std::ios_base::in);
  std::fstream kiFile (*++argv, std::ios_base::out);

  unsigned char b; // ide olvassik majd a bejövő fájl bájtjait
  LZWBinFa binFa; // s nyomjuk majd be az LZW fa objektumunkba

  // a bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen írjuk, hogy meg tudjuk
  // majd nézni... :) l. az említett 5. ea. C -> C++ gyökkettes átírási példáit

  while (beFile.read ((char *) &b, sizeof (unsigned char))) {
// egyszerűen a korábbi d.c kódját bemásoljuk
// laboron többször lerajzoltuk ezt a bit-tologatást:
// a b-ben lévő bájt bitjeit egyenként megnézzük
      for (int i = 0; i < 8; ++i)
      {
    // maszkolunk
          int egy_e = b & 0x80;
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if megmondja melyik:
          if ((egy_e >> 7) == 1)
  // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW fa objektumunkba
              binFa << '1';
          else
  // különben meg a '0' betűt:
              binFa << '0';
          b <<= 1;
      }

  }

  //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi verziókban de, hogy izgalmasabb legyen
  // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

  kiFile << binFa; // ehhez kell a globális operator<< túlterhelése, lásd fentebb
  // (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, mégis megy, hurrá)

  kiFile  << "depth = " << binFa.getMelyseg () << std::endl;
  kiFile  << "mean = " << binFa.getAtlag () << std::endl;
  kiFile  << "var = " << binFa.getSzoras () << std::endl;

  binFa.szabadit ();
```

```
    kiFile.close();
    beFile.close();

    return 0;
}
```

Ebben a programban a gyökér csomópont kompozícióban van a fával. Egy tag kompozíciónak számít, ha a tag az osztály része, a tag egyszerre csak egy osztályba tartozik, ha az osztály befejeződik, akkor a tag is, illetve ha a tag nem tud az osztály létezéséről.

```
protected:
    Csomopont gyoker;
```

A gyoker nevű csomópont nem tud az LZWBinFa osztály létezéséről, csak az LZWBinFa osztályba tartozik, ha az osztály elpusztul, akkor a gyoker is elpusztul. Mivel a gyoker az LZWBinFa osztály része, az előzőek figyelembevétele után egyértelmű, hogy kompozícióban áll vele.

6.5. Mutató a gyökér

Írd át az előző forrást, hogy a gyökér csomópont ne kompozícióban, csak aggregációban legyen a fával!

Aggregációról beszélünk, ha: A tag az osztály része. A tag több mint egy osztály része lehet egy időben. A tag nem pusztul el, ha az osztály elpusztul. A tag nem tud az osztály létezéséről.

Mivel az aggregáció egyszerre csak egy osztály része lehet, ezért, ha a gyoker csomópontot a Csomopont osztályon belül deklaráljuk, akkor az már nem lehet kompozíció, csak aggregáció

```
// z3a2.cpp
//
// Együtt támadjuk meg: http://progpater.blog.hu/2011/04/14/ ↩
   egyutt_tamadjuk_meg
// LZW fa építő 3. C++ átirata a C valtozatbol (+mélység, atlag és szórás)
// Programozó Páternoszter
//
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail ↩
   .com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthetõ illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszõleges) késõbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1, http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2, csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3, http://progpater.blog.hu/2011/03/05/ ↩
   labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// 0.0.4, z.cpp: a C verzióból svn: bevezetes/C/ziv/z.c átírjuk C++-ra
//        http://progpater.blog.hu/2011/03/31/ ↩
   imadni_fogjatok_a_c_t_egy_emberkent_tiszta_szivbol
// 0.0.5,   z2.cpp: az fgv(*mut)-ok helyett fgv(&ref)
// 0.0.6,   z3.cpp: Csomopont beágyazva
//          http://progpater.blog.hu/2011/04/01/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2
// 0.0.6.1  z3a2.c: LZWBinFa már nem barátja a Csomopont-nak, mert annak  ↩
   tagjait nem használja direktben
// 0.0.6.2  Kis kommentezést teszünk bele 1. lépésként (hogy a kicsit  ↩
   lemaradt hallgatóknak is
//    könnyebb legyen, jól megtűzdeljük további olvasmányokkal)
//    http://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg
//    (majd a 2. lépésben "beletesszük a d.c-t", majd s 3. lépésben a  ↩
   parancssorsor argok feldolgozását)

#include <iostream> // mert olvassuk a std::cin, írjuk a std::cout  ↩
   csatornákat
#include <cmath> // mert vonunk gyököt a szóráshoz: std::sqrt
#include <fstream> // fájlból olvasunk, írunk majd

/* Az LZWBinFa osztályban absztraháljuk az LZW algoritmus bináris fa  ↩
   építését. Az osztály
 definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ↩
```

```
     ez lesz a
 beágyazott Csomopont osztály. Miért ágyazzuk be? Mert külön nem szánunk ↩
    neki szerepet, ezzel
 is jelezzük, hogy csak a fa részeként számiolunk vele.*/

class LZWBinFa
{
public:
    /* Szemben a bináris keresőfánkkal (BinFa osztály)
     http://progpater.blog.hu/2011/04/12/ ↩
        imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3
     itt (LZWBinFa osztály) a fa gyökere nem pointer, hanem a '/' betüt ↩
        tartalmazó objektum,
     lásd majd a védett tagok között lent: Csomopont gyoker;
     A fa viszont már pointer, mindig az épülő LZW-fánk azon csomópontjára ↩
        mutat, amit az
     input feldolgozása során az LZW algoritmus logikája diktál:
     http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
     Ez a konstruktor annyit csinál, hogy a fa mutatót ráállítja a gyökérre ↩
        . (Mert ugye
     labopon, blogon, előadásban tisztáztuk, hogy a tartalmazott tagok, ↩
        most "Csomopont gyoker"
     konstruktora előbb lefut, mint a tagot tartalmazó LZWBinFa osztály ↩
        konstruktora, éppen a
     következő, azaz a fa=&gyoker OK.)
     */
    LZWBinFa (): fa(&gyoker) {}

    /* Tagfüggvényként túlterheljük a << operátort, ezzel a célunk, hogy ↩
       felkeltsük a
     hallgató érdeklődését, mert ekkor így nyomhatjuk a fába az inputot: ↩
        binFa << b; ahol a b
     egy '0' vagy '1'-es betű.
     Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett ↩
        paraméterként"
     kapott ) példány, azaz annak a fának amibe éppen be akarjuk nyomni a b ↩
         betűt a tagjai
     (pl.: "fa", "gyoker") használhatóak a függvényben.

     A függvénybe programoztuk az LZW fa építésének algoritmusát tk.:
     http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

     a b formális param az a betű, amit éppen be kell nyomni a fába: */
    void operator<<(char b)
    {
        // Mit kell betenni éppen, '0'-t?
        if (b == '0')
        {
            /* Van '0'-s gyermeke az aktuális csomópontnak?
             megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
```

```
            if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
            {
                // elkészítjük, azaz példányosítunk a '0' betű akt. ↩
                   parammal
                Csomopont *uj = new Csomopont ('0');
                // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
                // jegyezze már be magának, hogy nullás gyereke mostantól ↩
                   van
                // küldjük is Neki a gyerek címét:
                fa->ujNullasGyermek (uj);
                // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
                fa = &gyoker;
            }
            else // ha van, arra rálépünk
            {
                // azaz a "fa" pointer már majd a szóban forgó gyermekre ↩
                   mutat:
                fa = fa->nullasGyermek ();
            }
        }
        // Mit kell betenni éppen, vagy '1'-et?
        else
        {
            if (!fa->egyesGyermek ())
            {
                Csomopont *uj = new Csomopont ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }
    /* A bejárással kapcsolatos függvényeink (túlterhelt kiir-ók, atlag, ↩
       ratlag stb.) rekurzívak,
     tk. a rekurzív fabejárást valósítják meg (lásd a 3. előadás "Fabejárás ↩
       " c. fóliáját és társait)

     (Ha a rekurzív függvénnyel általában gondod van => K&R könyv megfelel ↩
       ő része: a 3. ea. izometrikus
     részében ezt "letáncoltuk" :) és külön idéztük a K&R álláspontját :)
     */
    void kiir (void)
    {
        // Sokkal elegánsabb lenne (és más, a bevezetésben nem kibontandó ↩
           reentráns kérdések miatt is, mert
        // ugye ha most két helyről hívják meg az objektum ilyen ↩
           függvényeit, tahát ha kétszer kezd futni az
```

```
        // objektum kiir() fgv.-e pl., az komoly hiba, mert elromlana a mélység... tehát a mostani megoldásunk
        // nem reentráns) ha nem használnánk a C verzióban globális változókat, a C++ változatban példánytagot a
        // mélység kezelésére: http://progpater.blog.hu/2011/03/05/there_is_no_spoon
        melyseg = 0;
        // ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát, akkor a
        // sztenderd out-ra nyomjuk
        kiir (&gyoker, std::cout);
    }
    void szabadit (void)
    {
        szabadit (gyoker.egyesGyermek());
        szabadit (gyoker.nullasGyermek());
        // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a szabad tárban (halmon).
    }

    /* A változatosság kedvéért ezeket az osztálydefiníció (class LZWBinFa {...};) után definiáljuk,
     hogy kénytelen légy az LZWBinFa és a :: hatókör operátorral minősítve definiálni :) l. lentebb */
    int getMelyseg (void);
    double getAtlag (void);
    double getSzoras (void);

    /* Vágyunk, hogy a felépített LZW fát ki tudjuk nyomni ilyenformán: std::cout << binFa;
     de mivel a << operátor is a sztenderd névtérben van, de a using namespace std-t elvből
     nem használjuk bevezető kurzusban, így ez a konstrukció csak az argfüggő névfeloldás miatt
     fordul le (B&L könyv 185. o. teteje) ám itt nem az a lényeg, hanem, hogy a cout ostream
     osztálybeli, így abban az osztályban kéne módosítani, hogy tudjon kiírni LZWBinFa osztálybelieket...
     e helyett a globális << operátort terheljük túl, */
    friend std::ostream& operator<< (std::ostream& os, LZWBinFa& bf)
    {
        bf.kiir(os);
        return os;
    }
    void kiir (std::ostream& os)
    {
        melyseg = 0;
        kiir (&gyoker, os);
    }
```

```
private:
    class Csomopont
    {
    public:
        /* A paraméter nélküli konstruktor az elepértelmezett '/' "gyökér- betűvel" hozza
         létre a csomópontot, ilyet hívunk a fából, aki tagként tartalmazza a gyökeret.
         Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu" tagba, a két
         gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is megteszi. */
        Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
        ~Csomopont () {};
        // Aktuális csomópont, mondd meg nékem, ki a bal oldali gyermeked
        // (a C verzió logikájával műxik ez is: ha nincs, akkor a null megy vissza)
        Csomopont *nullasGyermek () const {
            return balNulla;
        }
        // Aktuális csomópon,t mondd meg nékem, ki a jobb oldali gyermeked?
        Csomopont *egyesGyermek () const {
            return jobbEgy;
        }
        // Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a bal oldali gyereked!
        void ujNullasGyermek (Csomopont * gy) {
            balNulla = gy;
        }
        // Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a jobb oldali gyereked!
        void ujEgyesGyermek (Csomopont * gy) {
            jobbEgy = gy;
        }
        // Aktuális csomópont: Te milyen betűt hordozol?
        // (a const kulcsszóval jelezzük, hogy nem bántjuk a példányt)
        char getBetu() const {
            return betu;
        }

    private:
        // friend class LZWBinFa; /* mert ebben a valtozatban az LZWBinFa metódusai nem közvetlenül
        // a Csomopont tagjaival dolgoznak, hanem beállító/lekérdező üzenetekkel érik el azokat */

        // Milyen betűt hordoz a csomópont
        char betu;
        // Melyik másik csomópont a bal oldali gyermeke? (a C változatból " örökölt" logika:
```

```
        // ha hincs ilyen csermek, akkor balNulla == null) igaz
        Csomopont *balNulla;
        Csomopont *jobbEgy;
        // nem másolható a csomópont (ökörszabály: ha van valamilye a szabad tárban,
        // letiltjuk a másoló konstruktort, meg a másoló értékadást)
        Csomopont (const Csomopont &);
        Csomopont & operator=(const Csomopont &);
    protected:
  Csomopont gyoker;
    };

    /* Mindig a fa "LZW algoritmus logikája szerinti aktuális" csomópontjára mutat */
    Csomopont *fa;
    // technikai
    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;
    // szokásosan: nocopyable
    LZWBinFa (const LZWBinFa &);
    LZWBinFa & operator=(const LZWBinFa &);

    /* Kiírja a csomópontot az os csatornára. A rekurzió kapcsán lásd a korábbi K&R-es utalást...*/
    void kiir (Csomopont* elem, std::ostream& os)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió leállítása
        if (elem != NULL)
        {
            ++melyseg;
            kiir (elem->egyesGyermek(), os);
            // ez a postorder bejáráshoz képest
            // 1-el nagyobb mélység, ezért -1
            for (int i = 0; i < melyseg; ++i)
                os << "---";
            os << elem->getBetu() << "(" << melyseg - 1 << ")" << std::endl;
            kiir (elem->nullasGyermek(), os);
            --melyseg;
        }
    }
    void szabadit (Csomopont * elem)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió leállítása
        if (elem != NULL)
        {
            szabadit (elem->egyesGyermek());
            szabadit (elem->nullasGyermek());
```

```
            // ha a csomópont mindkét gyermekét felszabadítottuk
            // azután szabadítjuk magát a csomópontot:
            delete elem;
        }
    }

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

    int maxMelyseg;
    double atlag, szoras;

    void rmelyseg (Csomopont* elem);
    void ratlag (Csomopont* elem);
    void rszoras  (Csomopont* elem);

};

// Néhány függvényt az osztálydefiníció után definiálunk, hogy lássunk ↩
   ilyet is ... :)
// Nem erőltetjük viszont a külön fájlba szedést, mert a ↩
   sablonosztályosított tovább
// fejlesztésben az linkelési gondot okozna, de ez a téma már kivezet a ↩
   laborteljesítés
// szükséges feladatából: http://progpater.blog.hu/2011/04/12/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3

// Egyébként a melyseg, atlag és szoras fgv.-ek a kiir fgv.-el teljesen egy ↩
    kaptafa.

int LZWBinFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (&gyoker);
    return maxMelyseg-1;
}
double LZWBinFa::getAtlag (void)
{
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double)atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;
```

```
    rszoras (&gyoker);

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

    return szoras;
}
void LZWBinFa::rmelyseg (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek());
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        rmelyseg (elem->nullasGyermek());
        --melyseg;
    }
}
void
LZWBinFa::ratlag (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyesGyermek());
        ratlag (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}
void
LZWBinFa::rszoras  (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyesGyermek());
        rszoras (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)
```

```
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

// teszt pl.: http://progpater.blog.hu/2011/03/05/ labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// [norbi@sgu ~]$ echo "01111001001001000111"|./z3a2
// ------------1(3)
// ---------1(2)
// ------1(1)
// ---------0(2)
// ------------0(3)
// ---------------0(4)
// ---/(0)
// ---------1(2)
// ------0(1)
// ---------0(2)
// depth = 4
// mean = 2.75
// var = 0.957427
// a laborvédéshez majd ezt a tesztelést használjuk:
// http://

/* Ez volt eddig a main, de most komplexebb kell, mert explicite bejövő, kimenő fájlokkal kell dolgozni
int
main ()
{
    char b;
    LZWBinFa binFa;

    while (std::cin >> b)
    {
        binFa << b;
    }

    //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi verziókban de, hogy izgalmasabb legyen
    // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

    std::cout << binFa; // ehhez kell a globális operator<< túlterhelése, lásd fentebb

    std::cout << "depth = " << binFa.getMelyseg () << std::endl;
    std::cout << "mean = " << binFa.getAtlag () << std::endl;
    std::cout << "var = " << binFa.getSzoras () << std::endl;
```

```
    binFa.szabadit ();

    return 0;
}
*/

/* A parancssor arg. kezelést egyszerűen bedolgozzuk a 2. hullám kapcsolódó ↩
    feladatából:
 http://progpater.blog.hu/2011/03/12/hey_mikey_he_likes_it_ready_for_more_3
 de mivel nekünk sokkal egyszerűbb is elég, alig hagyunk meg belőle valamit ↩
    ...
 */

void usage(void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    // http://progpater.blog.hu/2011/03/12/ ↩
       hey_mikey_he_likes_it_ready_for_more_3
    // alapján a parancssor argok ottani elegáns feldolgozásából kb. ennyi  ↩
       marad:
    // "*((*++argv)+1)"...

    // a kiírás szerint ./lzwtree in_file -o out_file alakra kell mennie,  ↩
       ez 4 db arg:
    if (argc != 4) {
        // ha nem annyit kapott a program, akkor felhomályosítjuk erről a  ↩
           júzetr:
        usage();
        // és jelezzük az operációs rendszer felé, hogy valami gáz volt...
        return -1;
    }

    // "Megjegyezzük" a bemenő fájl nevét
    char *inFile = *++argv;

    // a -o kapcsoló jön?
    if (*((*++argv)+1) != 'o')  {
        usage();
        return -2;
    }

    // ha igen, akkor az 5. előadásból kimásoljuk a fájlkezelés C++  ↩
       változatát:
    std::fstream beFile (inFile, std::ios_base::in);
    std::fstream kiFile (*++argv, std::ios_base::out);
```

```
  unsigned char b; // ide olvassik majd a bejövő fájl bájtjait
  LZWBinFa binFa; // s nyomjuk majd be az LZW fa objektumunkba

  // a bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen ↩
     írjuk, hogy meg tudjuk
  // majd nézni... :) l. az említett 5. ea. C -> C++ gyökkettes átírási ↩
     példáit

  while (beFile.read ((char *) &b, sizeof (unsigned char))) {
// egyszerűen a korábbi d.c kódját bemásoljuk
// laboron többször lerajzoltuk ezt a bit-tologatást:
// a b-ben lévő bájt bitjeit egyenként megnézzük
      for (int i = 0; i < 8; ++i)
      {
    // maszkolunk
          int egy_e = b & 0x80;
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ↩
       megmondja melyik:
          if ((egy_e >> 7) == 1)
  // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW fa ↩
     objektumunkba
              binFa << '1';
          else
  // különben meg a '0' betűt:
              binFa << '0';
          b <<= 1;
      }

  }

  //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ↩
     verziókban de, hogy izgalmasabb legyen
  // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

  kiFile << binFa; // ehhez kell a globális operator<< túlterhelése, lásd ↩
      fentebb
  // (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, ↩
      mégis megy, hurrá)

  kiFile  << "depth = " << binFa.getMelyseg () << std::endl;
  kiFile  << "mean = " << binFa.getAtlag () << std::endl;
  kiFile  << "var = " << binFa.getSzoras () << std::endl;

  binFa.szabadit ();

  kiFile.close();
  beFile.close();

  return 0;
```

```
}
```

6.6. Mozgató szemantika

Írj az előző programhoz mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva!

A Mozgató konstruktort az LZWBinFa osztályon belül kell definiálni. Mozgató konstruktor az előző programhoz:

```
// z3a2.cpp
//
// Együtt támadjuk meg: http://progpater.blog.hu/2011/04/14/ ↩
   egyutt_tamadjuk_meg
// LZW fa építő 3. C++ átirata a C valtozatbol (+mélység, atlag és szórás)
// Programozó Páternoszter
//
// Copyright (C) 2011, Bátfai Norbert, nbatfai@inf.unideb.hu, nbatfai@gmail ↩
   .com
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.
//
// Ez a program szabad szoftver; terjeszthetõ illetve módosítható a
// Free Software Foundation által kiadott GNU General Public License
// dokumentumában leírtak; akár a licenc 3-as, akár (tetszõleges) késõbbi
// változata szerint.
//
// Ez a program abban a reményben kerül közreadásra, hogy hasznos lesz,
// de minden egyéb GARANCIA NÉLKÜL, az ELADHATÓSÁGRA vagy VALAMELY CÉLRA
// VALÓ ALKALMAZHATÓSÁGRA való származtatott garanciát is beleértve.
// További részleteket a GNU General Public License tartalmaz.
//
// A felhasználónak a programmal együtt meg kell kapnia a GNU General
// Public License egy példányát; ha mégsem kapta meg, akkor
```

```
// tekintse meg a <http://www.gnu.org/licenses/> oldalon.
//
//
// Version history:
//
// 0.0.1, http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
// 0.0.2, csomópontok mutatóinak NULLázása (nem fejtette meg senki :)
// 0.0.3, http://progpater.blog.hu/2011/03/05/ ↩
   labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// 0.0.4, z.cpp: a C verzióból svn: bevezetes/C/ziv/z.c átírjuk C++-ra
//        http://progpater.blog.hu/2011/03/31/ ↩
   imadni_fogjatok_a_c_t_egy_emberkent_tiszta_szivbol
// 0.0.5,   z2.cpp: az fgv(*mut)-ok helyett fgv(&ref)
// 0.0.6,   z3.cpp: Csomopont beágyazva
//          http://progpater.blog.hu/2011/04/01/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_2
// 0.0.6.1  z3a2.c: LZWBinFa már nem barátja a Csomopont-nak, mert annak  ↩
   tagjait nem használja direktben
// 0.0.6.2  Kis kommentezést teszünk bele 1. lépésként (hogy a kicsit  ↩
   lemaradt hallgatóknak is
//    könnyebb legyen, jól megtűzdeljük további olvasmányokkal)
//    http://progpater.blog.hu/2011/04/14/egyutt_tamadjuk_meg
//    (majd a 2. lépésben "beletesszük a d.c-t", majd s 3. lépésben a  ↩
   parancssorsor argok feldolgozását)

#include <iostream> // mert olvassuk a std::cin, írjuk a std::cout  ↩
   csatornákat
#include <cmath> // mert vonunk gyököt a szóráshoz: std::sqrt
#include <fstream> // fájlból olvasunk, írunk majd

/* Az LZWBinFa osztályban absztraháljuk az LZW algoritmus bináris fa  ↩
   építését. Az osztály
 definíciójába beágyazzuk a fa egy csomópontjának az absztrakt jellemzését, ↩
    ez lesz a
 beágyazott Csomopont osztály. Miért ágyazzuk be? Mert külön nem szánunk  ↩
    neki szerepet, ezzel
 is jelezzük, hogy csak a fa részeként számiolunk vele.*/

class LZWBinFa
{
public:
  LZWBinFa(LZWBinFa&& other)
     : _data(nullptr)
    , _length(0)
  {

      _data = other._data;
      _length = other._length;

       other._data = nullptr;
```

```
      other._length = 0;
}

// ertekadas
LZWBinFa& operator=(LZWBinfa&& other)
{
    if (this != &other)
    {
          delete[] _data;

          _data = other._data;
          _length = other._length;

          other._data = nullptr;
         other._length = 0;
    }
    return *this;
}
  /* Szemben a bináris keresőfánkkal (BinFa osztály)
   http://progpater.blog.hu/2011/04/12/ ↩
      imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3
   itt (LZWBinFa osztály) a fa gyökere nem pointer, hanem a '/' betüt ↩
      tartalmazó objektum,
   lásd majd a védett tagok között lent: Csomopont gyoker;
   A fa viszont már pointer, mindig az épülő LZW-fánk azon csomópontjára ↩
      mutat, amit az
   input feldolgozása során az LZW algoritmus logikája diktál:
   http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor
   Ez a konstruktor annyit csinál, hogy a fa mutatót ráállítja a gyökérre ↩
      . (Mert ugye
   labopon, blogon, előadásban tisztáztuk, hogy a tartalmazott tagok, ↩
      most "Csomopont gyoker"
   konstruktora előbb lefut, mint a tagot tartalmazó LZWBinFa osztály ↩
      konstruktora, éppen a
   következő, azaz a fa=&gyoker OK.)
   */
  LZWBinFa (): fa(&gyoker) {}

  /* Tagfüggvényként túlterheljük a << operátort, ezzel a célunk, hogy ↩
     felkeltsük a
   hallgató érdeklődését, mert ekkor így nyomhatjuk a fába az inputot: ↩
      binFa << b; ahol a b
   egy '0' vagy '1'-es betű.
   Mivel tagfüggvény, így van rá "értelmezve" az aktuális (this "rejtett ↩
      paraméterként"
   kapott ) példány, azaz annak a fának amibe éppen be akarjuk nyomni a b ↩
       betűt a tagjai
   (pl.: "fa", "gyoker") használhatóak a függvényben.

   A függvénybe programoztuk az LZW fa építésének algoritmusát tk.:
```

```
 http://progpater.blog.hu/2011/02/19/gyonyor_a_tomor

 a b formális param az a betű, amit éppen be kell nyomni a fába: */
void operator<<(char b)
{
    // Mit kell betenni éppen, '0'-t?
    if (b == '0')
    {
        /* Van '0'-s gyermeke az aktuális csomópontnak?
         megkérdezzük Tőle, a "fa" mutató éppen reá mutat */
        if (!fa->nullasGyermek ()) // ha nincs, hát akkor csinálunk
        {
            // elkészítjük, azaz példányosítunk a '0' betű akt. ↩
               parammal
            Csomopont *uj = new Csomopont ('0');
            // az aktuális csomópontnak, ahol állunk azt üzenjük, hogy
            // jegyezze már be magának, hogy nullás gyereke mostantól ↩
               van
            // küldjük is Neki a gyerek címét:
            fa->ujNullasGyermek (uj);
            // és visszaállunk a gyökérre (mert ezt diktálja az alg.)
            fa = &gyoker;
        }
        else // ha van, arra rálépünk
        {
            // azaz a "fa" pointer már majd a szóban forgó gyermekre ↩
               mutat:
            fa = fa->nullasGyermek ();
        }
    }
    // Mit kell betenni éppen, vagy '1'-et?
    else
    {
        if (!fa->egyesGyermek ())
        {
            Csomopont *uj = new Csomopont ('1');
            fa->ujEgyesGyermek (uj);
            fa = &gyoker;
        }
        else
        {
            fa = fa->egyesGyermek ();
        }
    }
}
/* A bejárással kapcsolatos függvényeink (túlterhelt kiir-ók, atlag, ↩
   ratlag stb.) rekurzívak,
 tk. a rekurzív fabejárást valósítják meg (lásd a 3. előadás "Fabejárás ↩
    " c. fóliáját és társait)
```

```
     (Ha a rekurzív függvénnyel általában gondod van => K&R könyv megfelel ↩
        ő része: a 3. ea. izometrikus
     részében ezt "letáncoltuk" :) és külön idéztük a K&R álláspontját :)
     */
    void kiir (void)
    {
        // Sokkal elegánsabb lenne (és más, a bevezetésben nem kibontandó  ↩
           reentráns kérdések miatt is, mert
        // ugye ha most két helyről hívják meg az objektum ilyen  ↩
           függvényeit, tahát ha kétszer kezd futni az
        // objektum kiir() fgv.-e pl., az komoly hiba, mert elromlana a  ↩
           mélység... tehát a mostani megoldásunk
        // nem reentráns) ha nem használnánk a C verzióban globális  ↩
           változókat, a C++ változatban példánytagot a
        // mélység kezelésére: http://progpater.blog.hu/2011/03/05/ ↩
           there_is_no_spoon
        melyseg = 0;
        // ha nem mondta meg a hívó az üzenetben, hogy hova írjuk ki a fát, ↩
            akkor a
        // sztenderd out-ra nyomjuk
        kiir (&gyoker, std::cout);
    }
    void szabadit (void)
    {
        szabadit (gyoker.egyesGyermek());
        szabadit (gyoker.nullasGyermek());
        // magát a gyökeret nem szabadítjuk, hiszen azt nem mi foglaltuk a  ↩
           szabad tárban (halmon).
    }

    /* A változatosság kedvéért ezeket az osztálydefiníció (class LZWBinFa  ↩
       {...};) után definiáljuk,
     hogy kénytelen légy az LZWBinFa és a :: hatókör operátorral minősítve  ↩
        definiálni :) l. lentebb */
    int getMelyseg (void);
    double getAtlag (void);
    double getSzoras (void);

    /* Vágyunk, hogy a felépített LZW fát ki tudjuk nyomni ilyenformán: std ↩
       ::cout << binFa;
     de mivel a << operátor is a sztenderd névtérben van, de a using  ↩
        namespace std-t elvből
     nem használjuk bevezető kurzusban, így ez a konstrukció csak az  ↩
        argfüggő névfeloldás miatt
     fordul le (B&L könyv 185. o. teteje) ám itt nem az a lényeg, hanem,  ↩
        hogy a cout ostream
     osztálybeli, így abban az osztályban kéne módosítani, hogy tudjon  ↩
        kiírni LZWBinFa osztálybelieket...
     e helyett a globális << operátort terheljük túl, */
    friend std::ostream& operator<< (std::ostream& os, LZWBinFa& bf)
```

```
    {
        bf.kiir(os);
        return os;
    }
    void kiir (std::ostream& os)
    {
        melyseg = 0;
        kiir (&gyoker, os);
    }

private:
    class Csomopont
    {
    public:
        /* A paraméter nélküli konstruktor az elepértelmezett '/' "gyökér- betűvel" hozza
         létre a csomópontot, ilyet hívunk a fából, aki tagként tartalmazza a gyökeret.
         Máskülönben, ha valami betűvel hívjuk, akkor azt teszi a "betu" tagba, a két
         gyermekre mutató mutatót pedig nullra állítjuk, C++-ban a 0 is megteszi. */
        Csomopont (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
        ~Csomopont () {};
        // Aktuális csomópont, mondd meg nékem, ki a bal oldali gyermeked
        // (a C verzió logikájával műxik ez is: ha nincs, akkor a null megy vissza)
        Csomopont *nullasGyermek () const {
            return balNulla;
        }
        // Aktuális csomópon,t mondd meg nékem, ki a jobb oldali gyermeked?
        Csomopont *egyesGyermek () const {
            return jobbEgy;
        }
        // Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a bal oldali gyereked!
        void ujNullasGyermek (Csomopont * gy) {
            balNulla = gy;
        }
        // Aktuális csomópont, ímhol legyen a "gy" mutatta csomópont a jobb oldali gyereked!
        void ujEgyesGyermek (Csomopont * gy) {
            jobbEgy = gy;
        }
        // Aktuális csomópont: Te milyen betűt hordozol?
        // (a const kulcsszóval jelezzük, hogy nem bántjuk a példányt)
        char getBetu() const {
            return betu;
        }
```

```
    private:
        // friend class LZWBinFa; /* mert ebben a valtozatban az LZWBinFa ↩
           metódusai nem közvetlenül
        // a Csomopont tagjaival dolgoznak, hanem beállító/lekérdező ↩
           üzenetekkel érik el azokat */

        // Milyen betűt hordoz a csomópont
        char betu;
        // Melyik másik csomópont a bal oldali gyermeke? (a C változatból " ↩
           örökölt" logika:
        // ha hincs ilyen csermek, akkor balNulla == null) igaz
        Csomopont *balNulla;
        Csomopont *jobbEgy;
        // nem másolható a csomópont (ökörszabály: ha van valamilye a ↩
           szabad tárban,
        // letiltjuk a másoló konstruktort, meg a másoló értékadást)
        Csomopont (const Csomopont &);
        Csomopont & operator=(const Csomopont &);
    protected:
Csomopont gyoker;
    };

    /* Mindig a fa "LZW algoritmus logikája szerinti aktuális" ↩
       csomópontjára mutat */
    Csomopont *fa;
    // technikai
    int melyseg, atlagosszeg, atlagdb;
    double szorasosszeg;
    // szokásosan: nocopyable
    LZWBinFa (const LZWBinFa &);
    LZWBinFa & operator=(const LZWBinFa &);

    /* Kiírja a csomópontot az os csatornára. A rekurzió kapcsán lásd a ↩
       korábbi K&R-es utalást...*/
    void kiir (Csomopont* elem, std::ostream& os)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ↩
           leállítása
        if (elem != NULL)
        {
            ++melyseg;
            kiir (elem->egyesGyermek(), os);
            // ez a postorder bejáráshoz képest
            // 1-el nagyobb mélység, ezért -1
            for (int i = 0; i < melyseg; ++i)
                os << "---";
            os << elem->getBetu() << "(" << melyseg - 1 << ")" << std::endl ↩
               ;
            kiir (elem->nullasGyermek(), os);
            --melyseg;
```

```
        }
    }
    void szabadit (Csomopont * elem)
    {
        // Nem létező csomóponttal nem foglalkozunk... azaz ez a rekurzió ↩
           leállítása
        if (elem != NULL)
        {
            szabadit (elem->egyesGyermek());
            szabadit (elem->nullasGyermek());
            // ha a csomópont mindkét gyermekét felszabadítottuk
            // azután szabadítjuk magát a csomópontot:
            delete elem;
        }
    }

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

    int maxMelyseg;
    double atlag, szoras;

    void rmelyseg (Csomopont* elem);
    void ratlag (Csomopont* elem);
    void rszoras  (Csomopont* elem);

};

// Néhány függvényt az osztálydefiníció után definiálunk, hogy lássunk ↩
   ilyet is ... :)
// Nem erőltetjük viszont a külön fájlba szedést, mert a ↩
   sablonosztályosított tovább
// fejlesztésben az linkelési gondot okozna, de ez a téma már kivezet a ↩
   laborteljesítés
// szükséges feladatából: http://progpater.blog.hu/2011/04/12/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_3

// Egyébként a melyseg, atlag és szoras fgv.-ek a kiir fgv.-el teljesen egy ↩
    kaptafa.

int LZWBinFa::getMelyseg (void)
{
    melyseg = maxMelyseg = 0;
    rmelyseg (&gyoker);
    return maxMelyseg-1;
}
double LZWBinFa::getAtlag (void)
{
```

```
    melyseg = atlagosszeg = atlagdb = 0;
    ratlag (&gyoker);
    atlag = ((double)atlagosszeg) / atlagdb;
    return atlag;
}
double LZWBinFa::getSzoras (void)
{
    atlag = getAtlag ();
    szorasosszeg = 0.0;
    melyseg = atlagdb = 0;

    rszoras (&gyoker);

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

    return szoras;
}
void LZWBinFa::rmelyseg (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        if (melyseg > maxMelyseg)
            maxMelyseg = melyseg;
        rmelyseg (elem->egyesGyermek());
        // ez a postorder bejáráshoz képest
        // 1-el nagyobb mélység, ezért -1
        rmelyseg (elem->nullasGyermek());
        --melyseg;
    }
}
void
LZWBinFa::ratlag (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        ratlag (elem->egyesGyermek());
        ratlag (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)
        {
            ++atlagdb;
            atlagosszeg += melyseg;
        }
    }
}
```

```
void
LZWBinFa::rszoras  (Csomopont* elem)
{
    if (elem != NULL)
    {
        ++melyseg;
        rszoras (elem->egyesGyermek());
        rszoras (elem->nullasGyermek());
        --melyseg;
        if (elem->egyesGyermek() == NULL && elem->nullasGyermek() == NULL)
        {
            ++atlagdb;
            szorasosszeg += ((melyseg - atlag) * (melyseg - atlag));
        }
    }
}

// teszt pl.: http://progpater.blog.hu/2011/03/05/ ↩
   labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pedat
// [norbi@sgu ~]$ echo "01111001001001000111"|./z3a2
// ----------1(3)
// ---------1(2)
// ------1(1)
// ---------0(2)
// ----------0(3)
// ---------------0(4)
// ---/(0)
// ---------1(2)
// ------0(1)
// ---------0(2)
// depth = 4
// mean = 2.75
// var = 0.957427
// a laborvédéshez majd ezt a tesztelést használjuk:
// http://

/* Ez volt eddig a main, de most komplexebb kell, mert explicite bejövő,  ↩
   kimenő fájlokkal kell dolgozni
int
main ()
{
    char b;
    LZWBinFa binFa;

    while (std::cin >> b)
    {
        binFa << b;
    }

    //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi  ↩
```

```
        verziókban de, hogy izgalmasabb legyen
    // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

    std::cout << binFa; // ehhez kell a globális operator<< túlterhelése, lásd fentebb

    std::cout << "depth = " << binFa.getMelyseg () << std::endl;
    std::cout << "mean = " << binFa.getAtlag () << std::endl;
    std::cout << "var = " << binFa.getSzoras () << std::endl;

    binFa.szabadit ();

    return 0;
}
*/

/* A parancssor arg. kezelést egyszerűen bedolgozzuk a 2. hullám kapcsolódó feladatából:
 http://progpater.blog.hu/2011/03/12/hey_mikey_he_likes_it_ready_for_more_3
 de mivel nekünk sokkal egyszerűbb is elég, alig hagyunk meg belőle valamit ...
 */

void usage(void)
{
    std::cout << "Usage: lzwtree in_file -o out_file" << std::endl;
}

int
main (int argc, char *argv[])
{
    // http://progpater.blog.hu/2011/03/12/hey_mikey_he_likes_it_ready_for_more_3
    // alapján a parancssor argok ottani elegáns feldolgozásából kb. ennyi marad:
    // "*((*++argv)+1)"...

    // a kiírás szerint ./lzwtree in_file -o out_file alakra kell mennie, ez 4 db arg:
    if (argc != 4) {
        // ha nem annyit kapott a program, akkor felhomályosítjuk erről a júzetr:
        usage();
        // és jelezzük az operációs rendszer felé, hogy valami gáz volt...
        return -1;
    }

    // "Megjegyezzük" a bemenő fájl nevét
    char *inFile = *++argv;
```

```
  // a -o kapcsoló jön?
  if (*((*++argv)+1) != 'o')  {
      usage();
      return -2;
  }

  // ha igen, akkor az 5. előadásból kimásoljuk a fájlkezelés C++ ↩
     változatát:
  std::fstream beFile (inFile, std::ios_base::in);
  std::fstream kiFile (*++argv, std::ios_base::out);

  unsigned char b; // ide olvassik majd a bejövő fájl bájtjait
  LZWBinFa binFa; // s nyomjuk majd be az LZW fa objektumunkba

  // a bemenetet binárisan olvassuk, de a kimenő fájlt már karakteresen ↩
     írjuk, hogy meg tudjuk
  // majd nézni... :) l. az említett 5. ea. C -> C++ gyökkettes átírási ↩
     példáit

  while (beFile.read ((char *) &b, sizeof (unsigned char))) {
// egyszerűen a korábbi d.c kódját bemásoljuk
// laboron többször lerajzoltuk ezt a bit-tologatást:
// a b-ben lévő bájt bitjeit egyenként megnézzük
      for (int i = 0; i < 8; ++i)
      {
    // maszkolunk
          int egy_e = b & 0x80;
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if ↩
       megmondja melyik:
          if ((egy_e >> 7) == 1)
  // ha a vizsgált bit 1, akkor az '1' betűt nyomjuk az LZW fa ↩
     objektumunkba
              binFa << '1';
          else
  // különben meg a '0' betűt:
              binFa << '0';
          b <<= 1;
      }

  }

  //std::cout << binFa.kiir (); // így rajzolt ki a fát a korábbi ↩
     verziókban de, hogy izgalmasabb legyen
  // a példa, azaz ki lehessen tolni az LZWBinFa-t kimeneti csatornára:

  kiFile << binFa; // ehhez kell a globális operator<< túlterhelése, lásd ↩
      fentebb
  // (jó ez az OO, mert mi ugye nem igazán erre gondoltunk, amikor írtuk, ↩
      mégis megy, hurrá)
```

```
    kiFile  << "depth = " << binFa.getMelyseg () << std::endl;
    kiFile  << "mean = " << binFa.getAtlag () << std::endl;
    kiFile  << "var = " << binFa.getSzoras () << std::endl;

    binFa.szabadit ();

    kiFile.close();
    beFile.close();

    return 0;
}
```

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...

11. fejezet

Helló, Gutenberg!

11.1. Programozási alapfogalmak

[?]

11.2. Programozás bevezetés

[KERNIGHANRITCHIE]

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

The C Programming Language, Brian W. Kernighan, Dennis M. Ritchie

1.1 Indulás.

A könyv írói szerint minden program elsajátítása a "Helló Világ!" program megírásával történik. Ez C nyelven így néz ki:

```
include <stdio.h>

int main()
{
    printf("Hello World!"\n);
}
```

Ezt a forráskódot programnev.c néven kell elmenteni, futtatása operációs rendszertől függ. Lefordítása a "gcc programnev.c -nev" paranccsal történik, futtatása Linuxon "./nev" , Windows alatt "nev" a terminálból. A program eredménye a következőképpen fog kinézni: Hello World!

Egy C program függvényekből (function) és változókból (variable) áll. Egy függvény pedig állításokból (statement). Az állítások megmondják, hogy mit kell csinálni, a változók pedig a számításokhoz szükséges

értékeket tárolják. A munkánkat különböző könyvtárak segíthetik, amelyeket inkludálni kell. Ezt itt az első sor teszi. A legelső sor azt mondja a compilernek, hogy az stdio.h (standard input/output) fájl tartalmát másolja át a mi programfájlunkba.

Az argumentumok azok az értékek, amelyeket függvényhíváskor adunk meg. Például a "printf("Helló Világ!\n")" esetén az printf() függvény argumentuma a "Helló Világ!\n" karakterlánc. Az argumentumokat mindig egy zárójelbe tesszük.

1.2 Változók és aritmetikai kifejezések

A következő program két oszlopban kiírja nullától háromszázig huszassával haladva a hőmérsékletet fahrenheit fokban mérve, és mellé celsiusban a C = (5/9)(F-32) képletet használva, ahol a C a celsius, F a fahrenheitban mért hőmérséklet.

```
#include <stdio.h>

/* fahrenheit celsius tabla kiíratasa
fahr = 0, 20, ..., 300-ra */
main()
{
  int fahr, celsius;
  int lower, upper, step;

  lower = 0;   /* minimum homerseklet */
  upper = 300;   /* maximum homerseklet */
  step = 20;   /* lepesek nagysaga */

  /* 20-al noveljuk fahr erteket a maximum elereseig */
  fahr = lower;
  while (fahr <= upper) {
    celsius = 5 * (fahr - 32) / 9;
    printf("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
  }
}
```

A változókat, használatuk előtt, mindig deklarálni kell a "típus név" formában. Ezeket deklarációknak hívjuk. A C nyelvben több különböző változótípust ismerünk, mint az int (integer - valósszám), char (character - karakter), float (floating point - lebegőpontos szám). A változók méretét tekintve is beszélhetünk mutatókról (pointer), tömbökről (array), uniókról (union), vagy struktúrákról (structure)

A program azzal kezdődik, hogy elvégezzük a szükséges deklarációkat, illetve értékeket adunk a step, lower és upper értékeknek. Ezután következik egy while loop, miszerint amíg a fahr kisebb vagy egyenlő az upper számnál, addig számolja ki a fahrhoz tartozó celsius értéket és írassa ki. Végül növeljök a fahr értékét steppel.

A while loopon belül láthatjuk, hogy a printf() függvényt használhatjuk az output formázására is. A %d decimális egész szám helyét jelöli, melyeket a kiírandó karakterlánc után adunk meg. Azt, hogy milyen

értéket akarunk kiíratni a % jel utáni betűk és számok kombinációjával tudjuk meghatározni. Például %6d (minimum 6 karakter széles decimális integer), %6.2f (minimum 6 karakter széles, tizedes vessző után két számjeggyel rendelkező lebegőpontos szám). Ugyanitt láthatunk példát escape sequence-re is (/n és /t). Az escape sequence-ek olyan / után írt karakterek, amelyeknek speciális funkcióik vannak. Például a /n új sort ír, a /t pedig egy tabulátort.

1.3 A for állítás

Ebben a részben a következő kódrészletet láthatjuk:

```
#include <stdio.h>

main()
{
  int fahr;

  for(fahr = 0; fahr <= 300; fahr = fahr + 20)
    printf("%3d %6.1f\n", fahr, (5.0/9.0*(fahr - 32)));
}
```

Ez a program ugyanazt csinálja, mint az előző, csak rövidebben van leírva. A legnagyobb változás a legtöbb változó elhagyása, csak a fahr marad meg. A másik nagyobb változás a for ciklus használata a while helyett, illetve a celsius értéket itt a printf() függvényen belül számoltatjuk ki.

A

```
for (int mettol; mettol < meddig; lepes);
```

for ciklus mettol értéktől meddig értékig lepes lépésszámmal haladva elvégez valamilyen a ciklus belsejében lévő műveletet, vagy azok halmazát.

1.4 Szimbolikus konstansok

A konstansok olyan változók, amelyeknek az értékeit nem tervezzük megváltoztatni. Konstansokat a C nyelvben a következőképpen lehet definiálni:

```
#define NEV ertek
```

Ahol a NEV a konstans neve, hagyományosan nagybetűkkel írva a könnyű megkülönböztetés érdekében, ertek pedig a konstans értéke.

1.5 Karakter input és output

A C programunk a szöveget karakterláncként kezeli, legegyszerűbb függvények, amelyek a szövegekkel dolgoznak a getchar() és a putchar(). Híváskor a getchar() beolvassa a kapott szöveget és visszaadja azt értékként, tehát a

```
c = getchar();
```

beolvassa a kapott input szöveget és azt az értéket adja a c változónak.

A getchar() párja a putchar(string), amely kiírja egy adott string változó értékét,

A következő egyszerű program kiírja a beolvasott értéket a konzolra:

```
#include <stdio.h>

main()
{
  int c;

  c = getchar();
  while( c != EOF )
  {
    putchar(c);
    c = getchar();
  }
}
```

A program úgy működik, hogy beolvassa az input értékeket és amíg az nem egyenlő az EOF értékkel (end of file, az az érték, ami azt jelzi, hogy nincs több input)

Ugyanígy megszámoltathatjuk a beírt sorokat, ha a while ciklusba rekunk egy if fügvényt, ahol megnézzük, hogyha a beírt karakter egy új sor (/n), akkor egy nl = 0 változó értékét megnöveljük egyel, majd a program végén printf() -fel kiíratjuk azt.

1.6 Tömbök

Tömböket (array) a következőképpen adhatunk meg: int digits[10], ami például 10 egész szám típusú értéket tartalmazó digits nevű tömböt ad meg. A tömbök számozása nullától kezdődik, tehát hivatkozni az n-edik elemre digits[n-1] -ként kell. Ez fontos a for ciklusokkal való tömbök körbejárásánál, feltöltésénél, ahol ezért a számozást int i = 0 -tól kell kezdeni.

A függvényeket (function) általában a main után szoktuk definiálni a következőképpen:

```
vszt_ert fgv_nev ( parameterek )
{
  deklaraciok;
  utasitasok;
}
```

Minden függvénymegadást a visszatérési értékkel (return value) kell kezdeni, ami a függvény által visszaadott érték típusa, majd ezután jön a függvény neve és utána () jelek között a paraméterei, azaz azok az értékek, amelyekkel a függvényünk dolgozni fog. Ezután jön a függvény teste, amelyekbe azokat az értékeket kell írnunk, amelyekkel dolgozni fog a függvényünk, illetve az utasításokat, amelyeket el fogja végezni. Ha a visszatérési értékünk nem nulla, azaz nem semmit ad vissza (void), akkor a return utasítással adhatjuk meg, hogy mit is adjon vissza a függvényünk.

Bevett szokás még a prototipizálás, amely lényege, hogy még a main függvény előtt deklaráljuk, hogy milyen függvényeket fogunk használni. Tesszük ezt a függvény visszatérési értékének, nevének és paramétereinek a megadásával. Majd a main után definiáljuk, hogy mit is értünk a deklarált függvények alatt.

A függvények azért hasznosak, mert csak egyszer kell őket megírni, ha az megvan, akkor a compiler beilleszti a kódjukat a megfelelő helyre. Tehát oda, ahol meghívjuk őket.

1.8 Argumentumok, érték szerinti hívás

C-ben minden függvény érték szerint hívja meg alapesetben az értékeket, azaz, nem az eredeti értékeket módosítják, hanem azoknak csak egy másolatát. Ennek az ellentetje, a referencia szerinti hívás, ahol a függvény a paramétereket módosítja, nem csak azok segítségével adja meg a visszatérési értéket.

1.9 Karaktertömbök

C-ben a leggyakorib tömb a karaktertömb, azaz a karakterlánc, ami nevéhez hűen karakterek egy tömbje. Vegyük például a "hello\n" karakterláncot, ami áll lényegében a hello szóból és egy új sort jelképező \n jelből. A karaktertömbben, amiben ez a string el van tárolva, minden karakternek megvan a maga értéke, és minden karakter egy helyet foglal el (itt a "\n" egy karakternek minősül) plusz egy /0 karakter jelzi az adott karakterlánc végét. A %c és a %s formátum specifikációk egyenként egy karaktert és egy karakterláncot jelképeznek.

1.10 Scope

Hatókör (scope) szerint megkülönböztetünk lokális és globális értékeket. A lokális értékek csak abban a fügvényblokkban érhetők el, ahol azokat deklaráltuk, de a globális értékek mindenhol. A globális értékeket a main függvényen kívül szokták deklarálni. Fontos, hogyha egy blokkon belül definiálunk egy lokális változót, amelynek a neve megegyezik egy globális változóéval, akkor az adott blokkon belül a lokális változó felülírja a globálist. Explicit módon deklarálhatunk globális változót az extern kulcsszó használatával. Ha egy külső változót szeretnék használni egy függvénnyel, akkor annak a változónak a nevét tudatnuk kell a függvénnyel, ezt az extern kulcsszóval tehetjük meg.

2.1 Változók nevei

A változók névadásainak vannak bizonyos szabályai. Minden változónév betűkből és számokból áll. Az első karakternek betűnek kell lennie. Az aláhúzásjel (_) itt betűnek számít, de nem ajánlott velük változónevet kezdeni, mert a könyvtárakban sok folyamat azzal kezdődik.

A C nyelvben vannak bizonyos kulcsszavak (például: if, else, continue, int), amelyek nem lehetnek változónevek

Jó ha olyan változóneveket választunk, amiknek közük van a feladatukhoz és nem túl hosszúak.

2.2 Típusok és méretek

A következő alaptípusok léteznek C-ben: char (egy karakter tárolására alkalmas bájt.), int (integer, egész szám), float (lebegőpontos szám), double (lebegőpontos szám, de nagyobb méretű, mint a float)

Ezeken kívül még használhatóak a short és long kifejezések egész számokra, amelyek egyenként 16 és 32 bit méretűek. Megadásuk nem kötelező.

A signed és az unsigned a char és int típusokkal használható, unsigned típusok mindig vagy 0 vagy pozitív értéket vesznek fel, signed értékek ezzel szemben lehetnek negatív számok is, a maximálisan felvehető pozitív értékük a felére minusz egyre csökken, tehár mivel a char 255 bitet vehet fel unsigned értékként, a maximális pozitív értéke signed értékként 127 bit. Minimum értéke -127.

2.3 Konstansok

A long és unsigned kulcsszavakon kívül egyenként felhasználhatjuk az L és U betűket is. Azaz az 1234 egy integer szám, az 123456789L egy long integer. A kettőt kombinálhatjuk UL-ként.

Értéket megadhatunk nyolcas és hexadecimális számrendszerben is. Példaként a 31 számot leírhatjuk úgy hogy 037 (37_8), vagy 0x1f ($1F_{16}$)

Az escape sequence egy / jel és egsy betű kombinácoiója, ami egy speciális karaktert jelöl. Példaként a már látott '\n' egy új sort jelöl. Az összes C-beli escape sequence:

```
\a  \\ alert jel, sipolas
\b  \\ backspace
\f  \\ formfeed, lapdobas
\n  \\ uj sor
\r  \\ kocsi vissza
\t  \\ horizontalis tab
\v  \\ vertikalis tab
\\  \\ \
\?  \\ ?
\'  \\ '
\"  \\ "
\ooo  \\ oktalis szam
\xhh  \\ hexadecimalis szam
```

A konstans kifejezés egy olyan kifejezés, amely csak konstansokat tartalmaz. Például

```
#define MAXLINE 1000
```

Egy MAXLINE nevű 1000 értékkel rendelkező konstanst definiál. A string konstans egy "" jelek közötti karakterlánc. Fontos tudni, hogy a stringet csak a "" jelek határozzák meg, tehát a "ab" "cd" egymás mellet egyenlő lesz "abcd"-vel, mindegy mennyi szóköz van közöttük.

Egy másik fajta konstans az enumerációs konstans (enumeration constant), amire egy példa:

```
enum boolean { NO, YES }
```

Ami egy boolean nevű enum konstanst definiál. Az első értékhez (itt: NO) a 0 tartozik, a másodikhoz pedig az 1 és így tovább további elemek esetén.

2.4 Deklarációk

Minden változót deklarálni kell használat előtt. Egy tipikus deklaráció:

```
char c, d;
```

Ami egy c és egy d nevű char típusú változók deklarálása. A változókat lehet külön-külön és egyben is deklarálni. Természetesen a különbözü típusú változókat külön kell deklarálnunk.

Az inicializáció a deklaráció azon altípusa, amikor értéket is adunk a változónak. Például:

```
char c = "x", d = "y";
```

Itt megadjuk, hogy c legyen "x" betű és d legyen "y".

deklarálhatunk a const kulcsszóval is, ha konstant kívánunk deklarálni. A folyamat ugyanaz, csak a típus elé oda kell írnunk, hogy const.

2.5 Aritmetikai műveletek

Binér (kétváltozós) aritmetikai operátorok a +,-,*,/,%. Ezek közül a % a modulus operátor. Feladata megmondani, hogy a % b esetén mennyi az a-nak b-vel való osztása eseténi maradék. Nem lehet float vagy double-lel alkalmazni. Precedencia szerint + és - azonos rangú, felettük a *, /, és % operátorok foglalnak helyet.

2.6 Relációs és logikai operátorok

Relációs operátorok a <, <=, > és >= . Egyenlőség operátorok az == és !=, ahol ! nemet jelent. Precedenciájuk az aritmetikai operátorok alatt van.

Logikai operátorok az && (és) és || (vagy)

2.7 Típus átváltások

Különböző függvények léteznek típusok átváltásaira, például az

```
atoi(str)
```

Átváltja str-t egész számmá. Egyéb hasonló függvények a lower() és upper() amelyek egyenként csupa kis vagy nagybetűkké váltanak egy stringet.

2.8 Inkrementálás és dekrememtálás

Az inkrementálás annyit tesz, mint egyel megnövelni valami értékét (operátora: ++). Ellentetje a dekrementálás (--). Az inkrementálás a következőképpen jelölendő: n++, vagy ++n . A dekrementálás: n-- vagy --n . A különbség aközött, hogy az érték elé írjuk (prefix) vagy után (postfix), hogy a prefixes alakban n használatba kerülése előtt inkrementáljuk/dekrementáljuk az értékét, postfix alakban pedig használat után.

2.9 Bitműveletek

A C nyelv 6 bitműveletet tartalmaz: & (ÉS), | (inklúzív VAGY), ^ (exkluzív/kizáró VAGY), << (left shift), >> (right shift) és ~ (komplementer) . A left és a right shift balra, vagy jobbra tolja az operátor bal oldalán lévő érték bitjeit az operátor jobb oldalán lévő számmal. (x << 2; x bitjeit kettővel balra tolja.)

2.10 Kifejezések és a megfeleltetés operátor

Megfeleltetni az egyenlőség jellel kell. Ha op egy operátor, kif1 és kif2 kifejezések akkor

```
kif1 op= kif2
```

Megegyezik a következővel:

```
kif1 = kif1 op kif2
```

2.11 Feltételes kifejezések

```
z = (a > b) ? a : b;
```

A fenti egy feltételes kifejezés. Jelentése: Ha a nagyobb b-nél, akkor z legyen a, egyébként legyen b. Ha a kérdőjel előtti kifejezés igazságértéke igaz, akkor a kettőpont előtti érték kerül felhasználásra, egyébként az azutáni.

2.12 Precedencia

Ez a rész a precedenciával, azaz a különböző operátorok végrehajtási sorrendjével. Elösször a zárójelek közötti kifejezéseket kell kiértékelni, aztán a kivonás, összeadás, a sizeof operátor, osztás, szorzás. Majd a relációs operátorok, aztán a bitműveletek, a logikai operátorok. A következő a feltételes kifejezés ?: operátor, majd végül a megfeleltető operátorok.

3.1 Utasítások és blokkok

Egy kifejezés utasítás lesz, ha pontosvessző követi. A {} jelek utasításcsoportokat blokkokra osztanak.

3.2 If-else

Az if-else utasítás a következőképpen néz ki:

```
if (kifejezes)
  utasitasok
else
  utasitasok
```

Működése a következő: Az if() részben lévő kifejezés igazságértékét megnézve, ha az érték igaz, teljesíti ai if alatti utasításokat, majd átugorja az else alattiakat. Ha az érték hamis, akkor az if alattiakat ugorja át és az else alatti utasításokat végzi el. Egy soros utasítások esetén a () jelek nem szükségesek az if() és else() után.

3.3 Else-if

```
if (kifejezes)
  utasitasok1
else if (kifejezes)
  utasitasok2
.......
else if (kifejezes)
  utasitasokX
else
  utasitasokx+1
```

Az else-if utasytás hasonlóan működik az if-else-hez, a különbség az else utasítások számában rejlik, amiből annyit írhatunk, amennyire szükségünk van. A kiértékelés fentről lefelé halad, tehát ha az utolsó előtti kifejezés sem igaz, akkor az utolsó else-hez tartozó utasításokat fogja elvégezni a program.

3.4 Switch

```
switch (kifejezes)
  case konstans: utasitasok
  case konstans: utasitasok
  default: utasitasok
```

A switch utasítás úgy működik, hogy a switch-beli kifejezés értéke lesz összevetve a case kulcsszavak utáni konstansokkal. Ha egyezés van, akkor az adott konstans utáni utasítások lesznek teljesítve. Ha nincs, akkor a default (alapértelmezett) utasításokra kerül a sor. Fontos, hogy ha egyezés van, akkor a többi case konstansaival való összevetés nem lesz átugorva, csak ha használjuk a break parancsot.

3.5 While és for

```
while (kifejezes)
  utasitas
```

Ez egy egyszerű while ciklus, ha a kifejezés igaz, az utasítások teljesülnek. Tipikusan az egyik utasítás a kifejezés módosításával van kapcsolatban, egyébként ha az igaz, akkor végtelen ciklusunk születik.

```
for(i = 0; i < n; i++)
  utasitasok
```

Ez a for ciklus i = 0 értéket vizsgálja. Addig folytatódik a ciklus, amíg teljesül az első pontosvessző utáni kifejezés. A második pontosvessző utáni utasítás minden egyes ciklus után (az utasitasok végrehajtása után) végrehajtódik.

3.6 Do while

```
do
  utasitasok
while (kifejezes);
```

A do-while ciklus hasonló a while ciklushoz, de azzal ellentétben itt mindenképp végrehajtódnak az utasítások egyszer. Majd aztán lesz megvizsgálva a releváns kifejezés igazságértéke. Fontos, hogy a while ciklussal ellentétben a do-while után kell a pontosvessző.

3.7 Break és continue

A break utasítás feladata a jelenlegi ciklusból való kilépés. A continue feladata a következő iteráció elkezdése.

3.8 Goto és label

A goto és a label utasítások használata szorosan összefügg. A goto leggyakoribb használata a beágyazott ciklusokból való kilépés. A label-t mindig egy kettőspont követi, a goto utasítást pedig a label neve, ahová szeretnénk "elugrani".

11.3. Programozás

[BMECPP]

A C++ nem objektumorientált tulajdonságai

2.1 A C és a C++ nyelv

2.1.1 Függvényparaméterek és visszatérési érték

C-ben, ha egy függvénynek nem afunk paramétert, akkor az meghívható tetszőleges mennyiségű paraméterrel, C++-ban viszont ez azt jelenti, hogy a függvény nem kér paramétert, azaz visszatérési értéke void. C-ben az alapértelmezett visszatérési érték int, C++-ban ilyen nincs.

2.1.2 A main függvény

C-vel ellentétben a C++-ban a main függvény kaphat argc és argv parancssori argumentumokat, amik az argumentumok számát és magukat az argumentumokat jelentik sorban.

2.1.3 A bool típus

A C++-ban a C-vel ellentétben létezik bool (boolean - logikai érték) típus. Értéke lehet true (1) vagy false (0). C-ben a logikai értéket int vagy enummal reprezentálhatjuk.

2.1.4 A C stílusú több-bájtos sztringek

A több-bájtos, pl. Unicode karakterek reprezentálására a C-ben rendelkezésre áll a w_char típus, amihez inkludálni kell a stddef.h és stdlib.h vagy wchar.h fájlokat. A C++ nyelven a wchar_t egy beépített.

2.1.5 Változódeklaráció mint utasítás

C++-ban minden olyan helyen állhat változódeklaráció, ahol utasítás állhat. Ott hozhatjuk létre a változókat, ahol valóban szükségünk van rájuk. Minden változó onnan használható, ahonnan deklaráljuk

2.2 Függvénynevek túlterhelése

C-ben minden függvényt a neve azonosít csak, C++-ban nemcsak a név, de a függvényekhez rendelt argumentumok száma és típusa is segít az azonosításban, aminek az az eredménye, hogy míg C-ben nem lehet, addig C++-ban lehet túlterhelni függvényt, azaz két különböző függvény rendelkezhet azonos névvel.

2.3 Alapértelmezett függvényargumentumok

A C++-ban lehetőség van a függvények argumentumainak alapértelmezett érték adására. Amikor így hívunk függvényeket, nem adunk meg argumentumokat.

2.4 Paraméteradás referenciatípussal

C-ben csak érték szerinti paraméteradás történik. Tehát egy függvény csak akkor fog módosítani értéket, hogyha egy arra mutató mutatót adunk meg argumentumnak. Ezzel szemben a C++ kínál referencia szerinti paraméteradást is, amelyel már módosítja az értékeket az adott függvény.

3. Objektumok és osztályok

3.1 Objektumorientáltság alapelvei

Az egyik alapelv amivel foglalkoznuk kell az az egységbezárás (encapsulation) alapelve, ahol egy bizonyos adatstruktúrát szeretnénk megjeleníteni a programunkban. Az egységbe záró adatstruktúra neve osztály vagy class. Az osztályok egyedi példányait objektumoknak nevezzük.

3.2 Egységbezárás C++-ban

C-ben az egységbezárást a struct kulcsszóval tesszük, struktúra típusba zárunk. Ugyanazt megtehetjük C++-ban is, de ott már nem csak tagváltozói lehetnek egy struktúrának, hanem tagfüggvényei is. A tagváltozót gyakran attribútumnak, a tagfüggvényt metódusnak nevezzük.

3.3 Adatrejtés

Adatrejtésre szolgál a private kulcsszó, amivel olyan adatokat adhatunk meg, amelyeket nem szeretnénk semmiképpen sem módosítani.

3.4 Konstruktorok és destruktorok

A konstruktor lehetőséget kínál arra, hogy az objektumok létrejüttükkör inicializálják magukat.

A konstruktorral szemben a destruktor az objektumok által birtokolt esetleges erőforrások felszabadítsát végzi el.

3.5 Dinamikus adattagot tartalmazó osztály

Ide tartozik a dinamikus memóriakezelés, a dinamikus adattagok támogatása és a másoló konstruktor.

3.6 Friend függvények

Az egyes osztályok feljogosíthatnak globális függvényeket a védett tagjaikhoz való hozzáférésre. Ezt a friend kulcsszóval tehetjük meg. A konstruktor olyan speciális függvény, amelynek neve megegyezik az osztály nevével és a példányosításakor automatikusan meghívódik.

3.7 Tagváltozók inicializálása

Objektumok állapotát lehet inicializálni konstruktorokban. Az értékadást és inicializálást a C++ nyelvben meg kell különböztetni. Inicializálás a változók létrehozásához kapcsolódik, értékadás a már meglévő változóknak való értékmódosítása, amit az egyenlőségjellel érhetünk el.

3.8 Statikus tagok

Osztályok esetében lehetőség van olyan speciális, úgynevezett statikus tagváltozók definiálására, melyek az adott osztályhoz és nem az osztály objektumaihoz tartoznak.

3.9 Beágyazott definíciók

A C++ nyelvben lehetőség van enumeráció-, osztály-, struktúra- és típusdefiníciók osztálydefiníción belüli megadására. Ezeket beágyazott definícióknak nevezzük.

6. Operátorok és túlterhelésük

6.1 Operátorok általában

C-ben az operátorok az argumentumaikon végeznek műveleteket, adott eredményeket a visszatérési értékeik feldolgozásával használhatjuk. Az operátorok visszatérítési értékét speciális szabályrendszer rögzíti. A C++ a C-hez képest bevezet néhány új operátort, mint a hatókör-operátor (::) vagy pointer-tag operátorok (.* és ->).

6.2 Függvényszintaxis és túlterhelés

Az operátorok speciális függvények, így a C++-ban túlterhelhetők. Ezt az operator kulcsszóval érhetjük el, előtte a visszatérési érték, majd a kulcsszó után maga az operátor jele és utána zárójelekben az operátor argumentumai.

IV. rész

Irodalomjegyzék

11.4. Általános

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

11.5. C

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

11.6. C++

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

11.7. 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.