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	Nagy, Martin	2019. május 10.	

REVISION HISTORY

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

Ajánlás

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

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

Tartalomjegyzék

Előszó

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

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

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

Hogyan forgasd

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

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

Milyen nyelven nyomjuk?

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

Hogyan nyomjuk?

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

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

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

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

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

I. rész

Bevezetés

1. fejezet

Vízió

1.1. Mi a programozás?

1.2. Milyen doksikat olvassak el?

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

1.3. Milyen filmeket nézzek meg?

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

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

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

Ebben a példában egy végtelen ciklust láthatunk aminek a "for"-os sora miatt 100%-os a CPU leterhelés 1 mag esetén mivel szünet nélkül fut. A **top** segítségével tudjuk ellenőrízni, hogy jól csináltuk-e meg a feladatot.

```
int main()
{
for(;;);
sleep(123);
}
```

Hasonlóan az elöző példához egy végtelen ciklus van csak hozzá lett téve egy "sleep" funkció ami szüneteltelteti adott(pl.: 123 másodperc jelen esetben) időre a programot ezáltal 0%-ra csökken a CPU terhelés 1 mag esetén.

```
int main()
#pragma omp parallel
  for (;;);
  }
return 0;
}
```

Itt a feladat hasonló az 1. példához, annyiban tér el, hogy az összes magon 100%-nak kell lennie a CPU terhelésnek. A kódba beírt "#pragma omp parallel" engedélyezi, hogy több szál fusson egyszerre. Ezen kivül még van két dolog egyik a "#include omp.h" a tetején illetve egy egy kapcsolót kell a végére tenni a fordításnál "-fopenmp". Ezeket teljesítve az összes magon 100%-os CPU terhelést fogunk kapni.

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

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

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

  boolean Lefagy(Program P)
  {
     if(P-ben van végtelen ciklus)
```

```
      return true;
    else
      return false;
  }

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

A T100-as program bemenetnek fog kapni egy másik programot aminek elkell döntenie, hogy van-e benne végtelen for ciklus avagy nincs. Ezután a T1000-es program ugyan ezt megcsinálja és megvizsgálja, hogy az adótt kódban van-e végtelen for ciklus vagy nincs. Ha igaz értéket ad megáll ha hamisat akkor végtelen for ciklusba kerül. Ezálltal ilyen programot eddig feltételezhetően lehetetlenség megírni, de egy ilyen program megírása sok gondunkra adna megoldást.

2.3. Változók értékének felcserélése

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

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

Megoldás forrása:

```
#include <iostream>
```

```
int main ()
  {
  int a = 7;
  int b = 1;
  std::cout<<"Jelenleg : a:"<< a << " " "b:" << b << std::endl;
  b = b + a;
  a = b - a;
  b = b - a;
  std::cout<<"Utánna : a:"<< a << " " "b:" << b << std::endl;
  }
```

Tegyük fel, hogy az eredeti a=7, az eredeti b=1. b = 1 + 7 (8) | a = 8 - 7 (1, az eredeti b) | b = 8 - 1 (7, az eredeti a) "cout" sorral pedig kiíratjuk a szöveget illetve az "a" és a "b" értékeket. Ez szimpla logika.

2.4. Labdapattogás

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

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

Megoldás forrása:

```
  int main (void)
  {
    WINDOWS *ablak;
    ablak = initscr ();

    int x = 0;
    int y = 0;
    int xnov = 1;
    int ynov = 1;
    int mx;
    int my;

    for (;;)
    {
      getmaxyz ( ablak,my,mx );
      mvprintw ( y,x,"O" );

      refresh ();
      usleep ( 100000 );
      clear ();

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

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

Az első sorban definiáljuk, hogy az *ablak Window pointer. Ezután egy végtelen ciklust hozunk létre amiben az első sor(getmaxyz) tartalmazza azt a területet ahol a labda pattogni fog. Ezt követően a második sor(mvprintw)-vel íratjuk ki koordinántáként a labdánkat az ablakba. Az "usleep" sor egy kis finomítás, hogy jobban látszódjon az eredmány mivel felfüggeszti a program futását adott időre. Ezek után az if-ek segítségével "irányítjuk" a labdát. Például ha balra felfelé megy a labda és eléri az ablak szélét ezesetben jobbra felfelé fog menni, de ha jobbra lefele megy és eléri a végpontot akkor balra lefele fog.

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

Írj egy programot, ami megnézi, hogy hány bites a szó a gépeden, azaz mekkora az int mérete. Használd ugyanazt a while ciklus fejet, amit Linus Torvalds a BogoMIPS rutinjában!

Megoldás forrása:

```
int main
{
  int szohossz = 0;
  int szo = 1;
  do
  {
    szohossz++;
  }
  while(szo<<=1);
  printf("A szó %d bites\n"; szohossz)
  return 0;
}
```

Bev. progon használt bit shifletésről van ebben a feladatban szó azaz addig shiftelünk míg eltünik az 1-es és csak nullákat fog tartalmazni. Közben számolva, hogy ez milyen hosszú. A fő folyamat a "while"-ban látható mikor is balra haladva shiftelünk egyet.

2.6. Helló, Google!

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

Ennél a feladatnál Dékány Róbert Zsolt tutoriált.

Megoldás videó:

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

void
kiir (double tomb[], int db){

  int i;

  for (i=0; i<db; ++i){
    printf("%f\n",tomb[i]);
  }
}

double
tavolsag (double PR[], double PRv[], int n){

  int i;
  double osszeg=0;

  for (i = 0; i < n; ++i)
    osszeg += (PRv[i] - PR[i]) * (PRv[i] - PR[i]);

  return sqrt(osszeg);
}

void
pagerank(double T[4][4]){
  double PR[4] = { 0.0, 0.0, 0.0, 0.0 };    //ebbe megy az eredmény
  double PRv[4] = { 1.0/4.0, 1.0/4.0, 1.0/4.0, 1.0/4.0};    //ezzel szorzok

  int i, j;

  for(;;){

    // ide jön a mátrix művelet

    for (i=0; i<4; i++){
```

```
      PR[i]=0.0;
      for (j=0; j<4; j++){
        PR[i] += T[i][j] * PRv[j];
      }
    }

      if (tavolsag(PR,PRv,4) < 0.0000000001)
        break;

    // ide meg az átpakolás PR-ből PRv-be

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

  kiir (PR, 4);
}

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 L1[4][4] = {
    {0.0,  0.0,      1.0/3.0,  0.0},
    {1.0,  1.0/2.0,  1.0/3.0,  0.0},
    {0.0,  1.0/2.0,  0.0,      0.0},
    {0.0,  0.0,    1.0/3.0,  0.0}
  };

  double L2[4][4] = {
    {0.0,  0.0,      1.0/3.0,  0.0},
    {1.0,  1.0/2.0,  1.0/3.0,  0.0},
    {0.0,  1.0/2.0,  0.0,      0.0},
    {0.0,  0.0,    1.0/3.0,  1.0}
  };

  printf("\nAz eredeti mátrix értékeivel történő futás:\n");
  pagerank(L);

  printf("\nAmikor az egyik oldal semmire sem mutat:\n");
  pagerank(L1);

  printf("\nAmikor az egyik oldal csak magára mutat:\n");
  pagerank(L2);
```

```
  printf("\n");

  return 0;
}
```

Ennek a feladatnak a kitalálója Larry Pane volt elsősorban ezáltal is lett a feladat lényegének az elnevezése azaz a PageRankolás. Ez egy google álltal kifejlesztett algoritmus ami félig publikus félig nem. Nem olyan bonyolult, de picit összetett és relatív ez az egész. Nevéből kiindulva azt vizsgálja, hogy milyen fontos egy adott oldalt. Józan paraszti ésszel ezt úgy néznénk, hogy ahány oldal mutat a kijelölt oldalunkra annál fontosabb de ez tévhiz. A nagyobb oldalból mutató oldalak nagyobb értékkel bírnak ezálltal lehet, hogy 1 neves oldalról mutató link álltali oldal többet ér mint 10db szenny oldalról mutató oldal.

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

Ennél a feladatnál Dékány Róbert Zsolt tutoriált.

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

```
  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")
```

Ehez a feladathoz szükséges lesz egy kiegészítő mégpedig a matlab.Majd létrehozunk egy függvényt(x).Elsőként egy olyan listát fog adni "x"-ig ami tartalmazza az összes prímet.Ezt követően kivonást végez addig ameddig nem talán olyan párt aminek a különbsége 2(azaz ikerprím).Legvégül pedig visszaadja a reciprokösszegüket. A függvény meghatározása után már nincs más dolgunk minthogy ábrázoljuk ezt a függvényt. x tengelyt felosztjuk 13-tól 1 000 000-ig 100 000 különbséggel. y érték egyenlő lesz a függvény értékével majd kirajzoltatjuk a plot segítségével ezt.

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

Tanulságok, tapasztalatok, magyarázat...

```
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)
```

Legelsőnek megadjuk a kisérletek számát ami 10millió lesz. A sample() függvény egy véletlenséget szimulál így következő lépésben ezt használjuk. "1:3" megadja mettől meddig generáljon random számokat. "kiserletek_szama" azt határozza meg, hogyszor tegye meg és a végén replace=T pedig, hogy ismétlődhessen a szám. A musorvezeto vector pedig az elöző két tömbtől fog függeni ezért csak a hosszúságát adjuk meg. A for ciklus 1-től a kisérletek számáig fog futni. Két eset fog fennállni egyik hogy eltalálta az ajtót a máik pedig, hogy nem. Ha a "kiserlet" és a "jatekos" tömb megegyezik, akkor a "musorvezeto"

nem választhatja azt az ajtót. A setdiff függvény kiveszi a kisérlet tömb első elemével megegyező értéket. Hasonlóan az else-nél is csak ott a kisérlet és a játékos tömb első elemét kikell vonni. Ezt követően pedig már feltudjuk tülteni a musorvezeto vektort amihez a sample() függvényt használjuk.

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás videó:

Megoldás forrása:

```
            #include <iostream>
void tounar(int a){
    for(int i=0; i<a; i++)
        std::cout << "1";

    std::cout << std::endl;
}
int main(){

    int val;
    std::cout << "Type a number in decimal." << std::endl;
    while(std::cin >> val){
        tounar(val);
    }

    return 0;
}
```

Az unáris nem más mint egy számrendszer. Kicsit eltérő mint a rendes mivel itt a legnagyobb szám az egyes. A többi számot úgy kapjuk meg, hogy az egyeseke összeadjuk tehát például a 3-mas számz az 111-ként jön létre. Legelsőnek megvizsgálja azt adott számot a számítógép. Ezt követően elmegy az utolsó szám utánig. Ezután visszalép egyel és elkezd kivonni mindig 1et belőlük. Ez végig fog menni a számsorozatunkon. Ezeket az egyeseket amiket kivont elmenti és mi majd innen fogjuk tudni megnézni az eredményt.

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

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

Megoldás videó:

Megoldás forrása:

```
         S, X, Y változók
a, b, c konstansok
S - abc, S - aXbc, Xb - bX, Xc - Ybcc, bY - Yb, aY - aaX, aY - aa
  S (S - aXbc)
  aXbc (Xb - bX)
  abXc (Xc - Ybcc)
  abYbcc (bY - Yb)
  aYbbcc (aY - aaX)
  aaXbbcc (Xb - bX)
  aabXbcc (Xb - bX)
  aabbXcc (Xc - Ybcc)
  aabbYbccc (bY - Yb)
  aabYbbccc (bY - Yb)
  aaYbbbccc (aY - aa)
  aaabbbccc
```

```
        A, B, C változók
a, b, c konstansok
A - aAB, A - aC, CB - bCc, cB - Bc, C - bc
  A (A - aAB)
  aAB ( A - aAB)
  aaABB ( A - aAB)
  aaaABBB ( A - aC)
  aaaaCBBB (CB - bCc)
  aaaabCcBB (cB - Bc)
  aaaabCBcB (cB - Bc)
  aaaabCBBc (CB - bCc)
  aaaabbCcBc (cB - Bc)
  aaaabbCBcc (CB - bCc)
  aaaabbbCccc (C - bc)
  aaaabbbbcccc
```

Mint ahogy láthatjuk megadtunk két darab szabályt is ebből is látszódik, hogy ez a nyelv nem környezetfüggetlen. Mind a két példánál megvannak adva, hogy mik a szimbólumok és a változok és maga a szabályok is.

3.3. Hivatkozási nyelv

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

Megoldás videó:

Megoldás forrása:

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

        return 0;
    }
```

Az alábbi általunk megírt kód C89 szabványon nem fut le, mivel egy hibaüzenetet kapunk ami szerint a fejlécben nem megengedett a változó deklarációja. C99-es szabványon viszont lefut ez. Természetesen ha a változó deklarálását a kivülre tesszük akkor már lefog futni C89 szabvály szerint. Ennél a feladatnál még szükséges megemlíteni, hogy két kapcsolóra van szükségünk a lefrodításnál, egyik a "-std:c89" másik pedig a "-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!

Megoldás videó:

Megoldás forrása:

```
%{
#include <stdio.h>
int realnum = 0;
%}
digit [0-9]
%%
{digit}*(\.{digit}+)? {++realnum_count;
printf("[realnum: %s - %f]",yytext,atof(yytext));}
%%
```

Legelsőnek is írunk egy C kód csipetet amit látni szeretnénk a generált programba. Egy adott részletet a "%" jellel tudunk lezárni lexerben. Ezután fognak következni a definicók amik jelen esetben számok definiálását fogja jelenteni. Ezután mint az elözőnél lezárjuk a "%" részletet. Ezt követően megadjuk a szabályt ami alapján felismerhetünk bizonyos részleteket. Jelen esetben csak egy szabályunk van ami a valós számokra fog érvényesülni.

```
int main(){
    yylex();
```

```
    printf("The number of real numbers: %d",realnum_count);
    return 0;
}
```

Itt már csak a főprogramunk maradt amiben a "yylex()" függvénny segítségével megívjuk a lexerünket és kiíratjuk az eredményt. Ha ezzel megvagyunk nem maradt más dolgunk mint, hogy beletesszük a flex programba ezt "lex valos.l -o valos.c" paranccsal. Ez a C forráskód jóval hosszabb mint az .l kód ezálltal és látható, hogy megspóroltunk vele némi időt.

3.5. l33t.l

Lexelj össze egy l33t ciphert!

Megoldás videó:

Megoldás forrása:

```
%{
  #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;
}
```

Ebben a feladatban a szleng nyelvvel fogunk foglalkozni. Különböző betűket tudunk használni akár másik számmal vagy másik szimbólikus karakterekkel is. Ilyen például az S betű helyett a $ vagy E betű helyett a 3. Manapság ezek igen elterjedtek a fiatalok körébe, viszont sokan pedig nem értik ezeket a szleng nyelveket.Maga ez egész feladat érthető és könnyen végigvihető. A kódcsípet első részében megvannak adva az adott betűnek a szlengben használt "jelei". Ezután következik, hogy a nagybetűket kisbetűvé alakítja át a kód és átírjam az első részben vett leet karakterekre. A legvégén pedig egyszerűen csak hivatkozunk rá és meghívjuk.

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);
```

Ha a SIGINT nem volt figyelmen kívül hagyva akkor a jelkezelő kezelje. Ha figyelmen kívül volt hagyva továbbra is maradjon úgy.

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

A forciklus fejlécébe hiányzik az i deklaráció.

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

A forciklus fejlécébe hiányzik az i deklaráció.

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

A kód hibamentes ha már létrehoztuk a látható változókat és mutatókat.

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

A kód hibamentes ha már létrehoztuk a látható változókat és mutatókat.

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

Arra kell figyelni hogy ha az f függvény visszatérési értéke nem int akkor a kiírt értékek nem biztos hogy pontosak lesznek.

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

A printf ki fogja írni az f függvény visszatérési értékét a-ra, és a értékét.

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

A kiiratás megtörténik viszont az f függvény most az a változó memória címével fog dolgozni nem az a értékével.

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

1. sor : Minden x esetén létezik olyan y amelyik nagyobb az x-nél és prímszám.

2. sor : Minden x esetén létezik olyan y amelyik nagyobb az x-nél és prímszám illetve y+2 is.

3. sor : Létezik y minden x esetén mikor x az prím és kisebb az y-nál.

4. sor : Létezik y minden x esetén mikor y kisebb mint az x és x az nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

- egész
```
int a;
```
- egészre mutató mutató
```
int *b;
```
- egész referenciája
```
int &c;
```
- egészek tömbje
```
int T[3];
```
- egészek tömbjének referenciája (nem az első elemé)
```
int (&TR)[3] = T;
```
- egészre mutató mutatók tömbje
```
int *T[3];
```
- egészre mutató mutatót visszaadó függvény
```
int *func();
```
- egészre mutató mutatót visszaadó függvényre mutató mutató
```
int *(*func)();
```
- egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény
```
int *(*func)();
```
- függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre
```
int *(*func)();
```

Mit vezetnek be a programba a következő nevek?

-
```
int a;
```

- Egész declaráció.

```
int *b = &a;
```

- Pointer, ami az "a" változóra mutat.

```
int &r = a;
```

- Az ’a’ változó referenciája.

```
int c[5];
```

- 5 elemű tömb deklaráció.

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

- A c tömb referenciája.

```
int *d[5];
```

- Int-re mutató 5 elemű pointer tömb.

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

- Int-re mutató függvény pointer.

```
int *(*l) ();
```

- Int-re mutató függvénypointer pointere.

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

- Egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvény

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

- Függvénymutató egy egészet visszaadó és két egészet kapó függvényre mutató mutatót visszaadó, egészet kapó függvényre

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

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

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

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

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

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

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

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

    }

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

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

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

    tm[3][0] = 42.0;
    (*(tm + 3))[1] = 43.0;  // mi van, ha itt hiányzik a külső ()
    *(tm[3] + 2) = 44.0;
    *(*(tm + 3) + 3) = 45.0;

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

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

    free (tm);

    return 0;
}
```

A feladat főcélja a dinamikus memóriakezelés. Legelején tisztázni kell mi is az az alsó háromszögmátrix. Ez egy olyan mátrix melyben a főátló felett csak is 0 számjegy szerepelhet. Legelső lépés a kódcsípet értelmezésében, hogy deklarálnunk kell az "nr" változót amit jelen esetben 5-re tettünk. Ez egyben egyenlő lesz a mátrix sorainak a számával is. Maga a "malloc" függvény memóriát foglalt le ami visszaad egy pointert. Ez a pointer jelen esetünkben bármi lehet(mi válasszuk meg). A harmadik "for" fogja kiíratni az első kettő "forban" létrejött háromszögmátrixot. Maga a kódcsípetben az "i" a sorok számát míg, a "j" az oszlopok számát mutatja. Azért kell "i+1" mivel így kizárjuk a nullával való osztást. Az utolsó for fogja felszabadítani a helyet a lefoglalt tömbnek.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás videó:

Megoldás forrása:

```
    #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);
    }
}
```

Első két sorban felveszünk 2 darab kontstant változót. Egyik a " MAX_KULCS 100 " lesz másik pedig a "BUFFER_MERET 256". Nevükhez hűen a MAX_KULCS a a kulcs maximális méretét fogja megadni míg a "BUFFER_MERET" a beolvasandó stringek számát maximum. Az intben lévő "strncpy" az egy függvény amivel a kulcsok méretét vizsgáljuk meg. Amennyiben túl nagy a kulcs mérete, ekkor 100 kitöröl belőle 100 karakternyi helyet. A "while" ciklus olvassa be a bájtokat. A kiolvasott bájtokat össze EXOR-ozza a kulcs adott bájtja segítségével. A kulcs indexet növeljük az EXOR után. A titkosított bájtokat egy bufferbe olvassuk.

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás videó:

Megoldás forrása:

```
    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();
      }
    }
```

Az elöző feladat megoldása JAVA környezetben. Viszont van egy fontos eltérés a között ami nem máős minthogy a klcsot a standard inputról kéri be.

4.4. C EXOR törő

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

Megoldás videó:

Megoldás forrása:

```
#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;
        }
```

Karakter sorozatból megadott kulcs alapján próbálja visszafejteni az EXOR törő a szöveget. Egy adott algoritmussal (Bruteforce) töri fel minden kombinációt felhasználva. Használnunk kell majd még egy algoritmust ami a tiszta szöveg előállításában segédkezik. Ez az algoritmust alapján egyes szavak alapján próbálja meg összerakni az értelmes szöveget.

4.5. Neurális OR, AND és EXOR kapu

Ebben a feladatban Dékány Róbert Zsoltot tutoriáltam.

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

```
#   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 háló elméletben könnyen működik, megadjuk, hogy egy adott bemenetre milyen kimenetet adjon vissza. Viszont ez a gyakorlatban egy kicsit nehezebb. Először nézzük a "neurális OR kaput". Ez a logikai vagy. Két db értéket kell megadnunk és a végeredményt majd kijön, hogy hány darab lépésből és mennyi idő alatt sikerült ezt neki elérnie. Másodszor a "neurális AND kaput" nézzük meg. Ez a logikai és. Itt is az elözőhöz mérten megadunk 2db értéket és az eredményt majd megpajuk hány lépésből jött ez ki neki és mennyi idő alatt. A harmadik a "kakuktojás" a három közül azaz az "EXOR", mivel itt is megadjuk a két darab értéket és az eredményt de itt kell hozzá adni egy titkos réteget is. Ez logikában amúgy a kizáró vaggyal felel meg.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás videó:

Megoldás forrása:

```
#include <iostream>
#include "mlp.hpp"
#include "png++/png.hpp"
int main (int argc, char **argv)
{
    png::image <png::rgb_pixel> png_image (argv[1]);
    int size = png_image.get_width()*png_image.get_height();

    Perceptron* p = new Perceptron(3, size, 256, 1);
    double* image = new double[size];

    for(int i {0}; i<png_image.get_width(); ++i)
         for(int j {0}; j<png_image.get_height(); ++j)
            image[i*png_image.get_width()+j] = png_image[i][j].red;
    double value = (*p) (image);
    std::cout << value << std::endl;
    delete p;
    delete [] image;
}
```

A feladat lényege, hogy három rétegű hálót csinálunk majd a 3. rétegben kapni fogunk egy eredményt (számot) amit a három rétegből különböző számításokból kaphatunk meg. Elsősorban behívjuk a képet. Második sorban helyet foglalunk le neki. Későbbiekben megkell adnunk pár paramétert ilyen a hány réteget haználnuk, méretet, 1 számnak kell lenni a végeredménynek. Ezután megint helyet foglalunk le és feltöltjük for ciklussal végül pedig meghívjuk a fgv. operátort. Ellenőrzés képpen pedig kiíratjuk az eredményt.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

Megoldás videó:

Ebben a feladatban Dékány Róbert Zsoltot tutoriáltam.

```
   #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;

}
```

Az egész csokorban hasznos lesz tudni a Mandelbrot "fogalmat", hogy mivel foglalkozik. Alapvetően komplex számokkal dolgozik egyenletkben. A megoldásban szereplő számokat "összekötve" egy képet fogunk kapni. Ebből kilogikázatjuk, hogy kapni fogunk egy kimenő fájlt amibe a megoldás lesz illetve egy képet is külön fájlban. Ezért szükséges a csokor elején telepíteni "png++"-t. Sudo apt segítségével tudjuk ezt megtenni. Miután ezt megtettük és utána néztünk több forrásból, hogy hogyan müködik ez pontosan el is kezdhetjük a feladatokat megcsinálni. Létrefogjuk hozni a képteret (min,max értékekkel). Maga a program további kódcsipetei annyiból állnak, hogy a kijtt számok(halmaz) segítségével kirajzoltatunk egy képet, ezt a képet úgy kapjuk meg, hogy pixelről-pixelre rajzoltatja ki a gép.

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

Megoldás videó:

Megoldás forrása:

```
#include <png++/png.hpp>
#include <complex>

const int N = 500;
const int M = 500;
const double MAXX = 0.7;
const double MINX = -2.0;
const double MAXY = 1.35;
const double MINY = -1.35;

void GeneratePNG(const int tomb[N][M])
{
    png::image< png::rgb_pixel > image(N, M);
    for (int x = 0; x < N; x++)
    {
        for (int y = 0; y < M; y++)
        {
            image[x][y] = png::rgb_pixel(tomb[x][y], tomb[x][y], tomb[x][y ↩
               ]);
        }
    }
    image.write("kimenet.png");
}

int main()
{
    int tomb[N][M];

    double dx = (MAXX - MINX) / N;
    double dy = (MAXY - MINY) / M;

    std::complex<double> C, Z, Zuj;

    int iteracio;

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            real(C) = MINX + j * dx;
            imag(C) = MAXY - i * dy;

            Z = 0;
            iteracio = 0;

            while(abs(Z) < 2 && iteracio++ < 255)
            {
                Zuj = Z*Z+C;
```

```
                Z = Zuj;
            }

            tomb[i][j] = 256 - iteracio;
        }
    }

    GeneratePNG(tomb);

    return 0;
}
```

Mint ahogy láthatjuk sok hasonlóság van ez elöző feladathoz képest viszont van amiben eltér mint például, hogy itt már az " std::complex" osztállyal fogunk dolgozni. Az "std::complex" osztály már tud kezelni komplex számokat ezálltal ez megkönnyíti a munkánkat. Ez annyiba fogja megváltoztatni, hogy nem kell saját magunknak írni egy struktúrát. Ezenfelül szinte ugyan az az elöző feladatban kitagláltakhoz.

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;
    }
```

Clifford Pickover találta fel a bioformokat a Julia halmazokat rajzoló programon dolgozott mikor csodálatos módon egy bug miatt rátalált a bioformokra. Az ilyenfajta szabályrendszer szerint valósulnak meg az egysejtűek. Ez a két halmaz között annyi a különbség, hogy míg a mandel brotban a "c" az egy konstants addig a másiknál egy állandó. Itt is a szélesség és a magasság a felhasználó álltal adott érték lesz mint az elöző két feladatnál.

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás videó:

Megoldás forrása:

```
// Forrás: https://github.com/SullyChen/Mandelbrot-Set-Plotter
#include "SFML/Graphics.hpp"
//resolution of the window
const int width = 1280;
const int height = 720;
//used for complex numbers
struct complex_number
{
    long double real;
    long double imaginary;
};
//mandelbrot komplex alapján legenerál egy mandelbrot halmazt
void generate_mandelbrot_set(sf::VertexArray& vertexarray, int pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            //scale the pixel location to the complex plane for calculations
            long double x = ((long double)j - pixel_shift_x) / zoom;
            long double y = ((long double)i - pixel_shift_y) / zoom;
            complex_number c;
            c.real = x;
            c.imaginary = y;
            complex_number z = c;
            int iterations = 0; //keep track of the number of iterations
            for (int k = 0; k < precision; k++)
            {
                complex_number z2;
                z2.real = z.real * z.real - z.imaginary * z.imaginary;
                z2.imaginary = 2 * z.real * z.imaginary;
                z2.real += c.real;
                z2.imaginary += c.imaginary;
                z = z2;
                iterations++;
                if (z.real * z.real + z.imaginary * z.imaginary > 4)
```

```
                    break;
                }
                //color pixel based on the number of iterations
                if (iterations < precision / 4.0f)
                {
                    vertexarray[i*width + j].position = sf:: ↩
                       Vector2f(j, i);
                    sf::Color color(iterations * 255.0f / ( ↩
                       precision / 4.0f), 0, 0);
                    vertexarray[i*width + j].color = color;
                }
                else if (iterations < precision / 2.0f)
                {
                    vertexarray[i*width + j].position = sf:: ↩
                       Vector2f(j, i);
                    sf::Color color(0, iterations * 255.0f / ( ↩
                       precision / 2.0f), 0);
                    vertexarray[i*width + j].color = color;
                }
                else if (iterations < precision)
                {
                    vertexarray[i*width + j].position = sf:: ↩
                       Vector2f(j, i);
                    sf::Color color(0, 0, iterations * 255.0f /  ↩
                       precision);
                    vertexarray[i*width + j].color = color;
                }
            }
        }
    }
```

Maga a CUDA eléggé elterjed manapság a gamerek és a videósok körében. Mivel nem elég, hogy egy játék elfut már arra is nagy hangsúlyt fektetnek, hogy milyen minőségben fut az el. A CUDA ebben segédkezik leginkább mivel egyszerre több magot futtat a videókártyán ezálltal nagyob teljesítményre képest a számítógép. A kód itt is hasonlít az elözőekhez, a legnagyobb eltérés viszont ott van amikor is itt a CUDA magjai számolják ki a megoldást míg az elözőeknél a CPU számolta. Mint ahogy írtam ezálltal jóval gyorsabb a folyamat.

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:

```
// Forrás: https://github.com/SullyChen/Mandelbrot-Set-Plotter
#include "SFML/Graphics.hpp"
//resolution of the window
const int width = 1280;
const int height = 720;
//used for complex numbers
struct complex_number
{
    long double real;
    long double imaginary;
};
//mandelbrot komplex alapján legenerál egy mandelbrot halmazt
void generate_mandelbrot_set(sf::VertexArray& vertexarray, int pixel_shift_x, int pixel_shift_y, int precision, float zoom)
{
    #pragma omp parallel for
    for(int i = 0; i < height; i++)
    {
        for (int j = 0; j < width; j++)
        {
            //scale the pixel location to the complex plane for calculations
            long double x = ((long double)j - pixel_shift_x) / zoom;
            long double y = ((long double)i - pixel_shift_y) / zoom;
            complex_number c;
            c.real = x;
            c.imaginary = y;
            complex_number z = c;
            int iterations = 0; //keep track of the number of iterations
            for (int k = 0; k < precision; k++)
            {
                complex_number z2;
                z2.real = z.real * z.real - z.imaginary * z.imaginary;
                z2.imaginary = 2 * z.real * z.imaginary;
                z2.real += c.real;
                z2.imaginary += c.imaginary;
                z = z2;
                iterations++;
                if (z.real * z.real + z.imaginary * z.imaginary > 4)
                    break;
            }
            //color pixel based on the number of iterations
            if (iterations < precision / 4.0f)
            {
                vertexarray[i*width + j].position = sf::
```

```
                        Vector2f(j, i);
                    sf::Color color(iterations * 255.0f / ( ↩
                        precision / 4.0f), 0, 0);
                    vertexarray[i*width + j].color = color;
                }
                else if (iterations < precision / 2.0f)
                {
                    vertexarray[i*width + j].position = sf:: ↩
                        Vector2f(j, i);
                    sf::Color color(0, iterations * 255.0f / ( ↩
                        precision / 2.0f), 0);
                    vertexarray[i*width + j].color = color;
                }
                else if (iterations < precision)
                {
                    vertexarray[i*width + j].position = sf:: ↩
                        Vector2f(j, i);
                    sf::Color color(0, 0, iterations * 255.0f /  ↩
                        precision);
                    vertexarray[i*width + j].color = color;
                }
            }
        }
    }
```

Mint szinte az összes többi feladatnál itt is van hasonlóság a mandelbrotos feladatokhoz. Egy függvény (void generate_mandelbrot_set(sf::VertexArray vertexarray..) fogja legenerálni MBH_t-t.

```
    // Forrás: https://github.com/SullyChen/Mandelbrot-Set-Plotter
    #include "SFML/Graphics.hpp"
    //resolution of the window
    const int width = 1280;
    const int height = 720;
    //used for complex numbers
    struct complex_number
    {
        long double real;
        long double imaginary;
    };
    //mandelbrot komplex alapján legenerál egy mandelbrot halmazt
    void generate_mandelbrot_set(sf::VertexArray& vertexarray, int  ↩
        pixel_shift_x, int pixel_shift_y, int precision, float zoom)
    {
        #pragma omp parallel for
        for(int i = 0; i < height; i++)
        {
            for (int j = 0; j < width; j++)
            {
```

```
                    //scale the pixel location to the complex plane for ↩
                        calculations
                    long double x = ((long double)j - pixel_shift_x) /  ↩
                       zoom;
                    long double y = ((long double)i - pixel_shift_y) /  ↩
                       zoom;
                    complex_number c;
                    c.real = x;
                    c.imaginary = y;
                    complex_number z = c;
                    int iterations = 0; //keep track of the number of  ↩
                       iterations
                    for (int k = 0; k < precision; k++)
                    {
                        complex_number z2;
                        z2.real = z.real * z.real - z.imaginary * z. ↩
                           imaginary;
                        z2.imaginary = 2 * z.real * z.imaginary;
                        z2.real += c.real;
                        z2.imaginary += c.imaginary;
                        z = z2;
                        iterations++;
                        if (z.real * z.real + z.imaginary * z.imaginary ↩
                            > 4)
                            break;
                    }
                    //color pixel based on the number of iterations
                    if (iterations < precision / 4.0f)
                    {
                        vertexarray[i*width + j].position = sf:: ↩
                           Vector2f(j, i);
                        sf::Color color(iterations * 255.0f / ( ↩
                           precision / 4.0f), 0, 0);
                        vertexarray[i*width + j].color = color;
                    }
                    else if (iterations < precision / 2.0f)
                    {
                        vertexarray[i*width + j].position = sf:: ↩
                           Vector2f(j, i);
                        sf::Color color(0, iterations * 255.0f / ( ↩
                           precision / 2.0f), 0);
                        vertexarray[i*width + j].color = color;
                    }
                    else if (iterations < precision)
                    {
                        vertexarray[i*width + j].position = sf:: ↩
                           Vector2f(j, i);
                        sf::Color color(0, 0, iterations * 255.0f /  ↩
                           precision);
                        vertexarray[i*width + j].color = color;
```

```
                    }
                }
            }
        }
```

Legelső lépés "sf::" ablaknál kell beállítani tulajdonságokat (pl.: ablakméret). Ezután lekell generálnunk a MBH-t default paraméterek segítségével. Két állapotot fog figyelni : 1 : Szokásosan várja mikor zárjuk be az ablakot. 2 : Belépve az MBH-ba, 2x nagyítással fogja lefuttatni a fájlt.

```
        int main()
        {
            sf::String title_string = "Mandelbrot Set Plotter"; //ablak ↩
                címe
            sf::RenderWindow window(sf::VideoMode(width, height),  ↩
               title_string); //ablak objektum(létrehozza az ablakot a  ↩
               megadott méretekkel és címmel)
            window.setFramerateLimit(30); //frissített ablak/s vagy  ↩
               ilyesmi
            sf::VertexArray pointmap(sf::Points, width * height);

            //értékek inicializálása
            float zoom = 300.0f;
            int precision = 100;
            int x_shift = width / 2;
            int y_shift = height / 2;

            //legenerálja a mbh-t
            generate_mandelbrot_set(pointmap, x_shift, y_shift,  ↩
               precision, zoom);


            /**
            *
            *
            *
            *
            * */
            while (window.isOpen())
            {
                //ciklikusan figyeli az előforduló különböző event-eket ↩
                   , ha egy olyan esemény következik be, hogy  ↩
                   rákattolunk az X gombra, akkor bezárja az ablakot
                sf::Event event;
                while (window.pollEvent(event))
                {
                    if (event.type == sf::Event::Closed)
                        window.close();
                }
```

```
            //ha a bal egérgommbal kattintunk, akkor az egér ↩
               helyére nagyít az alábbi algoritmus segítségével. ↩
               Minden nagyítás után újra legenerálja a mbh-t.
            //zoom into area that is left clicked
            if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
            {
                sf::Vector2i position = sf::Mouse::getPosition( ↩
                   window);
                x_shift -= position.x - x_shift;
                y_shift -= position.y - y_shift;
                zoom *= 2;
                precision += 200;

                #pragma omp parallel for
                for (int i = 0; i < width*height; i++)
                {
                    pointmap[i].color = sf::Color::Black;
                }
                generate_mandelbrot_set(pointmap, x_shift, y_shift, ↩
                    precision, zoom);
            }
            window.clear();
            window.draw(pointmap);
            window.display();
        }

        return 0;
    }
```

5.6. Mandelbrot nagyító és utazó Java nyelven

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

Ez az előző feladat megírása JAVÁ-ban. Itt is megkell adnak a különböző paramétereket. A megoldásnak kapott számok segítségével pixeleket rajzoltatunk ki majd színezünk ki ezálltal kapjuk meg az elvárt képet. Nagyítás során nem történik különösebb változá, ugyan úgy kiszámolja a pixeleket majd kiszinezi őket és így kapjuk meg a képet.

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.

Tanulságok, tapasztalatok, magyarázat... térj ki arra is, hogy a JDK forrásaiban a Sun programozói pont úgy csinálták meg ahogyan te is, azaz az OO nemhogy nem nehéz, hanem éppen természetes neked!

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>
class Random {

    public:

            Random();

            ~Random(){}

            double get();
    private:

            bool exist;
            double value;

};
Random::Random() {
        exist = false;
        std::srand (std::time(NULL));
};
double Random::get() {
    if (!exist){
```

```
            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);

            value = r * v2;
            exist = !exist;

            return r * v1;
    }

    else{
        exist = !exist;
        return value;
    }
};
int main()
{

        Random rnd;

        for (int i = 0; i < 10; ++i) std::cout << rnd.get() << std::endl;

}
```

Az elején szükségünk lesz pár különböző függvénykönyvtárra amik tartalmazzák a kódban használt függvényeket. Miután ezt megtettük létrehozzuk a "Random" osztály amiben megadjuk, hogy a kódunk melyik részei legyenek privátak illetve publikusok. Publikusba tesszük a konstruktórt, destruktort illetve a "get" függvényt. Ezt követően megadjuk a konstuktort ami annyit takar, hogyha új objektumot hozunk létre akkor a a változót hamisra állítjuk. Ezt követően a "get" függvényt írjuk meg amit 2 felé tudunk, az egyik ha nincs legenerált szám tehát hamis a tehát az "exist" hamis, ilyenkor lekell generálnunk 2 számot amiből az egyiket megtartjuk másikat pedig elrakjuk. Másik eset mikor létezik lekért szám ilyenkor az "exist"-et hamisra állítjuk és visszaadjuk a lekért számot. Végül megírjuk a főprogramot amiben az eddig megírt függvényeket felhasználjuk.

JAVA https://sourceforge.net/p/udprog/code/ci/master/tree/source/kezdo/elsojava/PolarGen.java#l10

Ez elöző kód átvitele JAVÁban.

6.2. LZW

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

Megoldás videó:

```
#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);
    }
}
```

Legelőször is muszáj definiálnunk a struktúránkat ami megadja az elem értékét illetve a bal és jobb oldali

mutatónak a mutatóját. Ezt követően létrehozunk egy "uj_elem" függvényt. Amennyiben például nincs elegendő hely vagy bármi féle hibába ütközik ezesetben kidob minket egy hibaüzenettel együtt. Megadjuk a "kiir" és a "szabadiit" függvényeket amelyek elé egy "extern" szót teszünk ami annyit tesz, hogy a programnak ezzel segítünk, hogy csak később definiáljuk teljesen ezeket a függvényeket. Csak ezek a lépések után kezdjók megírni a fő programunkat ami egy karakter típusu változóval és a binfa gyökérelemével kezdünk. Mint ahogy a kódban is láthatjuk az "uj_elem" függvény segítségével létrehozunk a binfához egy új elemet ami jelen esetben egy "/" jel, ezt betesszük a fába ami segítségével látni fogjunk, hogy hol járunk éppen. Ezután írunk egy "while" ciklust. Ezt követően az "if" függvényt használjuk ahol megnézzük, hogy a beolvasott karakter 0 akkor a fa álltal mutatott bal oldali csomópont elemét kell néznük ha nincs ott semmi(NULL) akkor létrehozza a program. Ha 1est kapunk ugyan az a helyzet mint az elözőnél csak jobb oldali ágon. Ezekután ki is íratjuk a fát. A főprogram után megadjuk a "melyseg" és a "max_melyseg" változókat.

6.3. Fabejárás

Járd be az előző (inorder bejárású) fát pre- és posztorder is!

Megoldás videó:

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

        kiir (elem->jobb_egy);
        kiir (elem->bal_nulla);
        --melyseg;
    }
}
```

```
void
kiir (BINFA_PTR elem)
{
    if (elem != NULL)
    {
        ++melyseg;

        if (melyseg > max_melyseg)
            max_melyseg = melyseg;
```

```
        kiir (elem->jobb_egy);
        kiir (elem->bal_nulla);

        for (int i = 0; i < melyseg; ++i)
            printf ("---");
        printf ("%c(%d)\n", elem->ertek < 2 ? '0' + elem->ertek : elem-> ↩
           ertek,
                melyseg);

        --melyseg;
    }
}
```

Egy fának 3 bejárási útvonala lehet. Inorder,preorder illetve postorder. Az elöző kód inorder bejárást alkalmazott most pedig megmutatom preorder és postorder bejárásokban is. Először is kezdjük a preorderrel. Igazából nem sok eltérés van csak annyiban, hogy legelőször a gyökérelemmel kell kezdenünk a vizsgálatot, ezt követi az egyik oldal és a másik oldal zárja be a folyamatot. Következő pedig a postorderes bejárás aminél a lényeg, hogy először az egyik oldalt vizsgáljuk meg majd a másikat és a legvégén a gyökérelemet.

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!

```
// z3a7.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, 2012, 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)
// 0.0.6.3      z3a2.c: Fejlesztgetjük a forrást: http://progpater.blog.hu ↩
   /2011/04/17/a_tizedik_tizenegyedik_labor
// 0.0.6.4      SVN-beli, http://www.inf.unideb.hu/~nbatfai/p1/forrasok-SVN ↩
   /bevezetes/vedes/
// 0.0.6.5      2012.03.20, z3a4.cpp: N betűk (hiányok), sorvégek, vezető  ↩
   komment figyelmen kívül: http://progpater.blog.hu/2012/03/20/ ↩
   a_vedes_elokeszitese
// 0.0.6.6      z3a5.cpp: mamenyaka kolléga észrevételére a több komment  ↩
   sor figyelmen kívül hagyása
//    http://progpater.blog.hu/2012/03/20/a_vedes_elokeszitese/ ↩
   fullcommentlist/1#c16150365
// 0.0.6.7  Javaslom ezt a verziót választani védendő programnak
// 0.0.6.8  z3a7.cpp: pár kisebb javítás, illetve a védések támogatásához  ↩
   további komment a <<
```

```
//    eltoló operátort tagfüggvényként, illetve globális függvényként ↩
   túlterhelő részekhez.
//    http://progpater.blog.hu/2012/04/10/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_4/fullcommentlist/1# ↩
   c16341099
//

#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
     laboron, 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)
  {
  }
   ~LZWBinFa ()
  {
    szabadit (gyoker.egyesGyermek ());
    szabadit (gyoker.nullasGyermek ());
  }
```

```
/* 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.

   a binFa << b (ahol a b majd a végén látszik, hogy már az '1' vagy a
      '0') azt jelenti
   tagfüggvényként, hogy binFa.operator<<(b) (globálisként így festene:
      operator<<(binFa, b) )

   */
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 megfelelő 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);
}
/* már nem használjuk, tartalmát a dtor hívja
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,

     a kiFile << binFa azt jelenti, hogy

      - tagfüggvényként: kiFile.operator<<(binFa) de ehhez a kiFile
         valamilyen
      std::ostream stream osztály forrásába kellene beleírni ezt a
         tagfüggvényt,
      amely ismeri a mi LZW binfánkat...

      - globális függvényként: operator<<(kiFile, binFa) és pont ez látszik
          a következő sorban:

     */
  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-bet
       űvel" 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 ↩
         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);

// fejlesztgetjük a forrást: http://progpater.blog.hu/2011/04/17/a_tizedik_tizenegyedik_labor
if (!beFile)
  {
    std::cout << inFile << " nem letezik..." << std::endl;
    usage ();
    return -3;
  }

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... :) 1. az említett 5. ea. C -> C++ gyökkettes átírási példáit

while (beFile.read ((char *) &b, sizeof (unsigned char)))
  if (b == 0x0a)
    break;

bool kommentben = false;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
  {

    if (b == 0x3e)
{     // > karakter
  kommentben = true;
  continue;
}

    if (b == 0x0a)
{     // újsor
  kommentben = false;
  continue;
```

```
  }

      if (kommentben)
  continue;

      if (b == 0x4e)    // N betű
  continue;

      // 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 eddig..., most már simán írjuk az if fejébe a legmagasabb ↩
        helyiértékű bit vizsgálatát
    // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if  ↩
       megmondja melyik:
    if (b & 0x80)
      // 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;

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

  return 0;
}
```

Bevezetés a programozásba című tantárgyon már taglaltuk ezt a programot és csipekedtünk benne. Itt már

alapból a gyökér része a binfának.

6.5. Mutató a gyökér

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

Megoldás videó:

A teljes forráskód:

```
// z3a7.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, 2012, 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)
// 0.0.6.3      z3a2.c: Fejlesztgetjük a forrást: http://progpater.blog.hu ↩
   /2011/04/17/a_tizedik_tizenegyedik_labor
// 0.0.6.4      SVN-beli, http://www.inf.unideb.hu/~nbatfai/p1/forrasok-SVN ↩
   /bevezetes/vedes/
// 0.0.6.5      2012.03.20, z3a4.cpp: N betűk (hiányok), sorvégek, vezető  ↩
   komment figyelmen kívül: http://progpater.blog.hu/2012/03/20/ ↩
   a_vedes_elokeszitese
// 0.0.6.6      z3a5.cpp: mamenyaka kolléga észrevételére a több komment  ↩
   sor figyelmen kívül hagyása
//    http://progpater.blog.hu/2012/03/20/a_vedes_elokeszitese/ ↩
   fullcommentlist/1#c16150365
// 0.0.6.7  Javaslom ezt a verziót választani védendő programnak
// 0.0.6.8  z3a7.cpp: pár kisebb javítás, illetve a védések támogatásához  ↩
   további komment a <<
//    eltoló operátort tagfüggvényként, illetve globális függvényként  ↩
   túlterhelő részekhez.
//    http://progpater.blog.hu/2012/04/10/ ↩
   imadni_fogjak_a_c_t_egy_emberkent_tiszta_szivbol_4/fullcommentlist/1# ↩
   c16341099
//

#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
     laboron, 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 ()
    {
        gyoker = new Csomopont ('/');
        fa = gyoker;
    }
    ~LZWBinFa ()
    {
        szabadit (gyoker->egyesGyermek ());
        szabadit (gyoker->nullasGyermek ());
        delete(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.

 a binFa << b (ahol a b majd a végén látszik, hogy már az '1' vagy a '0') azt jelenti
 tagfüggvényként, hogy binFa.operator<<(b) (globálisként így festene: operator<<(binFa, b) )

 */
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);
    }
    /* már nem használjuk, tartalmát a dtor hívja
  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,

     a kiFile << binFa azt jelenti, hogy

      - tagfüggvényként: kiFile.operator<<(binFa) de ehhez a kiFile valamilyen
      std::ostream stream osztály forrásába kellene beleírni ezt a tagfüggvényt,
      amely ismeri a mi LZW binfánkat...

      - globális függvényként: operator<<(kiFile, binFa) és pont ez látszik a következő sorban:

     */
    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 működik 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 &); //másoló konstruktor
        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);

// fejlesztgetjük a forrást: http://progpater.blog.hu/2011/04/17/ ↩
   a_tizedik_tizenegyedik_labor
if (!beFile)
{
    std::cout << inFile << " nem letezik..." << std::endl;
    usage ();
    return -3;
}

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)))
    if (b == 0x0a)
        break;

bool kommentben = false;

while (beFile.read ((char *) &b, sizeof (unsigned char)))
{

    if (b == 0x3e)
    {     // > karakter
        kommentben = true;
        continue;
    }

    if (b == 0x0a)
    {     // újsor
        kommentben = false;
        continue;
    }

    if (kommentben)
        continue;

    if (b == 0x4e)    // N betű
        continue;
```

```
        // 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 eddig..., most már simán írjuk az if fejébe a legmagasabb helyiértékű bit vizsgálatát
            // csupa 0 lesz benne a végén pedig a vizsgált 0 vagy 1, az if megmondja melyik:
            if (b & 0x80)
                // 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;

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

    return 0;
}
```

Maga az "aggreráció" szó jelentése annyit tesz, hogy különböző elemeket kapcsolnak össze. Egy könnyebb példa erre az összeadás művelet. Megkell változtatnunk azokat a kód csípeteket ahol a gyökér változóira hivatkozunk. A gyökér típusát könnyen megtudjuk változtatni csomópontra mutató pointerré ami ehez a feladathoz fog kellenni (Csomopont *gyoker;). A gyökér elől törölni kell a különböző operátorokat.

6.6. Mozgató szemantika

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

Megoldás forrása:

Ez egy egyszerű feladat azoknak akik átlátják feladatot.

```
    LZWBinFa ( LZWBinFa && regi ) {
        std::cout << "LZWBinFa move ctor" << std::endl;

        gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
        gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

        regi.gyoker.ujEgyesGyermek ( nullptr );
        regi.gyoker.ujNullasGyermek ( nullptr );

    }
    LZWBinFa& operator = (LZWBinFa && regi)
    {
        if (this == &regi)
            return *this;

        gyoker.ujEgyesGyermek ( regi.gyoker.egyesGyermek() );
        gyoker.ujNullasGyermek ( regi.gyoker.nullasGyermek() );

        regi.gyoker.ujEgyesGyermek ( nullptr );
        regi.gyoker.ujNullasGyermek ( nullptr );

        return *this;
    }
```

Legelőször a feladatban szereplő két szempontot kell véghez vinnünk azaz a mozgato konstruktort illetve az értékadást. Egyenlővé kell tennük a régi fa illetve az új fa tartalmát, ezt úgy tehetjük meg, ha a régi fát átmásoljuk az új fába. Maga a a kódcsipetben szereplő konstruktor illetve érték adásnak a mintája szinte megegyező.

```
LZWBinFa binFa2 = std::move(binFa);

    kiFile << binFa2;
    kiFile << "depth = " << binFa2.getMelyseg () << std::endl;
    kiFile << "mean = " << binFa2.getAtlag () << std::endl;
    kiFile << "var = " << binFa2.getSzoras () << std::endl;
```

Miután ezek ezek megtörténtek a "move" fügvénnyel átvisszük az új fába a régi fát. Ezek után nincs más dolgunk mint már csak kiíratni az új fa tartalmát amit a szokott módon tesszük meg a szükséges paraméterekkel.

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

1. passzolási lehetősségem (labor)

7.2. Java életjáték

Írd meg Java-ban a John Horton Conway-féle életjátékot, valósítsa meg a sikló-kilövőt!

```
/*
 * Sejtautomata.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * Sejtautomata osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class Sejtautomata extends java.awt.Frame implements Runnable {
    /** Egy sejt lehet élő */
    public static final boolean ÉLŐ = true;
    /** vagy halott */
    public static final boolean HALOTT = false;
    /** Két rácsot használunk majd, az egyik a sejttér állapotát
     * a t_n, a másik a t_n+1 időpillanatban jellemzi. */
    protected boolean [][][] rácsok = new boolean [2][][];
```

```
    /** Valamelyik rácsra mutat, technikai jellegű, hogy ne kelljen a
     * [2][][]-ból az első dimenziót használni, mert vagy az egyikre
     * állítjuk, vagy a másikra. */
    protected boolean [][] rács;
    /** Megmutatja melyik rács az aktuális: [rácsIndex][][] */
    protected int rácsIndex = 0;
    /** Pixelben egy cella adatai. */
    protected int cellaSzélesség = 20;
    protected int cellaMagasság = 20;
    /** A sejttér nagysága, azaz hányszor hány cella van? */
    protected int szélesség = 20;
    protected int magasság = 10;
    /** A sejttér két egymást követő t_n és t_n+1 diszkrét időpillanata
     közötti valós idő. */
    protected int várakozás = 1000;
    // Pillanatfelvétel készítéséhez
    private java.awt.Robot robot;
    /** Készítsünk pillanatfelvételt? */
    private boolean pillanatfelvétel = false;
    /** A pillanatfelvételek számozásához. */
    private static int pillanatfelvételSzámláló = 0;
    /**
     * Létrehoz egy <code>Sejtautomata</code> objektumot.
     *
     * @param      szélesség    a sejttér szélessége.
     * @param      magasság     a sejttér szélessége.
     */
    public Sejtautomata(int szélesség, int magasság) {
        this.szélesség = szélesség;
        this.magasság = magasság;
        // A két rács elkészítése
        rácsok[0] = new boolean[magasság][szélesség];
        rácsok[1] = new boolean[magasság][szélesség];
        rácsIndex = 0;
        rács = rácsok[rácsIndex];
        // A kiinduló rács minden cellája HALOTT
        for(int i=0; i<rács.length; ++i)
            for(int j=0; j<rács[0].length; ++j)
                rács[i][j] = HALOTT;
        // A kiinduló rácsra "élőlényeket" helyezünk
        //sikló(rács, 2, 2);
        siklóKilövő(rács, 5, 60);
        // Az ablak bezárásakor kilépünk a programból.
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent e) {
                setVisible(false);
                System.exit(0);
            }
        });
        // A billentyűzetről érkező események feldolgozása
```

```
        addKeyListener(new java.awt.event.KeyAdapter() {
            // Az 'k', 'n', 'l', 'g' és 's' gombok lenyomását figyeljük
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_K) {
                    // Felezük a cella méreteit:
                    cellaSzélesség /= 2;
                    cellaMagasság /= 2;
                    setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                            Sejtautomata.this.magasság*cellaMagasság);
                    validate();
                } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    // Duplázzuk a cella méreteit:
                    cellaSzélesség *= 2;
                    cellaMagasság *= 2;
                    setSize(Sejtautomata.this.szélesség*cellaSzélesség,
                            Sejtautomata.this.magasság*cellaMagasság);
                    validate();
                } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
                    pillanatfelvétel = !pillanatfelvétel;
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_G)
                    várakozás /= 2;
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_L)
                    várakozás *= 2;
                repaint();
            }
        });
        // Egér kattintó események feldolgozása:
        addMouseListener(new java.awt.event.MouseAdapter() {
            // Egér kattintással jelöljük ki a nagyítandó területet
            // bal felső sarkát vagy ugyancsak egér kattintással
            // vizsgáljuk egy adott pont iterációit:
            public void mousePressed(java.awt.event.MouseEvent m) {
                // Az egérmutató pozíciója
                int x = m.getX()/cellaSzélesség;
                int y = m.getY()/cellaMagasság;
                rácsok[rácsIndex][y][x] = !rácsok[rácsIndex][y][x];
                repaint();
            }
        });
        // Egér mozgás események feldolgozása:
        addMouseMotionListener(new java.awt.event.MouseMotionAdapter() {
            // Vonszolással jelöljük ki a négyzetet:
            public void mouseDragged(java.awt.event.MouseEvent m) {
                int x = m.getX()/cellaSzélesség;
                int y = m.getY()/cellaMagasság;
                rácsok[rácsIndex][y][x] = ÉLŐ;
                repaint();
            }
        });
        // Cellaméretek kezdetben
```

```
        cellaSzélesség = 10;
        cellaMagasság = 10;
        // Pillanatfelvétel készítéséhez:
        try {
            robot = new java.awt.Robot(
                    java.awt.GraphicsEnvironment.
                    getLocalGraphicsEnvironment().
                    getDefaultScreenDevice());
        } catch(java.awt.AWTException e) {
            e.printStackTrace();
        }
        // A program ablakának adatai:
        setTitle("Sejtautomata");
        setResizable(false);
        setSize(szélesség*cellaSzélesség,
                magasság*cellaMagasság);
        setVisible(true);
        // A sejttér életrekeltése:
        new Thread(this).start();
    }
    /** A sejttér kirajzolása. */
    public void paint(java.awt.Graphics g) {
        // Az aktuális
        boolean [][] rács = rácsok[rácsIndex];
        // rácsot rajzoljuk ki:
        for(int i=0; i<rács.length; ++i) { // végig lépked a sorokon
            for(int j=0; j<rács[0].length; ++j) { // s az oszlopok
                // Sejt cella kirajzolása
                if(rács[i][j] == ÉLŐ)
                    g.setColor(java.awt.Color.BLACK);
                else
                    g.setColor(java.awt.Color.WHITE);
                g.fillRect(j*cellaSzélesség, i*cellaMagasság,
                        cellaSzélesség, cellaMagasság);
                // Rács kirajzolása
                g.setColor(java.awt.Color.LIGHT_GRAY);
                g.drawRect(j*cellaSzélesség, i*cellaMagasság,
                        cellaSzélesség, cellaMagasság);
            }
        }
        // Készítünk pillanatfelvételt?
        if(pillanatfelvétel) {
            // a biztonság kedvéért egy kép készítése után
            // kikapcsoljuk a pillanatfelvételt, hogy a
            // programmal ismerkedő Olvasó ne írja tele a
            // fájlrendszerét a pillanatfelvételekkel
            pillanatfelvétel = false;
            pillanatfelvétel(robot.createScreenCapture
                    (new java.awt.Rectangle
                    (getLocation().x, getLocation().y,
```

```
                szélesség*cellaSzélesség,
                magasság*cellaMagasság)));
        }
    }
    /**
     * Az kérdezett állapotban lévő nyolcszomszédok száma.
     *
     * @param   rács    a sejttér rács
     * @param   sor     a rács vizsgált sora
     * @param   oszlop  a rács vizsgált oszlopa
     * @param   állapor a nyolcszomszédok vizsgált állapota
     * @return int a kérdezett állapotbeli nyolcszomszédok száma.
     */
    public int szomszédokSzáma(boolean [][] rács,
            int sor, int oszlop, boolean állapot) {
        int állapotúSzomszéd = 0;
        // A nyolcszomszédok végigzongorázása:
        for(int i=-1; i<2; ++i)
            for(int j=-1; j<2; ++j)
                // A vizsgált sejtet magát kihagyva:
                if(!((i==0) && (j==0))) {
            // A sejttérből szélének szomszédai
            // a szembe oldalakon ("periódikus határfeltétel")
            int o = oszlop + j;
            if(o < 0)
                o = szélesség-1;
            else if(o >= szélesség)
                o = 0;

            int s = sor + i;
            if(s < 0)
                s = magasság-1;
            else if(s >= magasság)
                s = 0;

            if(rács[s][o] == állapot)
                ++állapotúSzomszéd;
                }

        return állapotúSzomszéd;
    }
    /**
     * A sejttér időbeli fejlődése a John H. Conway féle
     * életjáték sejtautomata szabályai alapján történik.
     * A szabályok részletes ismertetését lásd például a
     * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
     * matematikai játékok. Polygon, Szeged 1998. 171. oldal.)
     */
    public void időFejlődés() {
```

```
        boolean [][] rácsElőtte = rácsok[rácsIndex];
        boolean [][] rácsUtána = rácsok[(rácsIndex+1)%2];

        for(int i=0; i<rácsElőtte.length; ++i) { // sorok
            for(int j=0; j<rácsElőtte[0].length; ++j) { // oszlopok

                int élők = szomszédokSzáma(rácsElőtte, i, j, ÉLŐ);

                if(rácsElőtte[i][j] == ÉLŐ) {
                /* Élő élő marad, ha kettő vagy három élő
                 szomszedja van, különben halott lesz. */
                   if(élők==2 || élők==3)
                       rácsUtána[i][j] = ÉLŐ;
                   else
                       rácsUtána[i][j] = HALOTT;
                }  else {
                /* Halott halott marad, ha három élő
                 szomszedja van, különben élő lesz. */
                   if(élők==3)
                       rácsUtána[i][j] = ÉLŐ;
                   else
                       rácsUtána[i][j] = HALOTT;
                }
            }
        }
        rácsIndex = (rácsIndex+1)%2;
    }
    /** A sejttér időbeli fejlődése. */
    public void run() {

        while(true) {
            try {
                Thread.sleep(várakozás);
            } catch (InterruptedException e) {}

            időFejlődés();
            repaint();
        }
    }
    /**
     * A sejttérbe "élőlényeket" helyezünk, ez a "sikló".
     * Adott irányban halad, másolja magát a sejttérben.
     * Az élőlény ismertetését lásd például a
     * [MATEK JÁTÉK] hivatkozásban (Csákány Béla: Diszkrét
     * matematikai játékok. Polygon, Szeged 1998. 172. oldal.)
     *
     * @param   rács    a sejttér ahová ezt az állatkát helyezzük
     * @param   x       a befoglaló tégla bal felső sarkának oszlopa
     * @param   y       a befoglaló tégla bal felső sarkának sora
     */
```

```
    public void sikló(boolean [][] rács, int x, int y) {

        rács[y+ 0][x+ 2] = ÉLŐ;
        rács[y+ 1][x+ 1] = ÉLŐ;
        rács[y+ 2][x+ 1] = ÉLŐ;
        rács[y+ 2][x+ 2] = ÉLŐ;
        rács[y+ 2][x+ 3] = ÉLŐ;

    }
    /**
     * A sejttérbe "élőlényeket" helyezünk, ez a "sikló ágyú".
     * Adott irányban siklókat lő ki.
     * Az élőlény ismertetését lásd például a
     * [MATEK JÁTÉK] hivatkozásban /Csákány Béla: Diszkrét
     * matematikai játékok. Polygon, Szeged 1998. 173. oldal./,
     * de itt az ábra hibás, egy oszloppal told még balra a
     * bal oldali 4 sejtes négyzetet. A helyes ágyú rajzát
     * lásd pl. az [ÉLET CIKK] hivatkozásban /Robert T.
     * Wainwright: Life is Universal./ (Megemlíthetjük, hogy
     * mindkettő tartalmaz két felesleges sejtet is.)
     *
     * @param   rács    a sejttér ahová ezt az állatkát helyezzük
     * @param   x       a befoglaló tégla bal felső sarkának oszlopa
     * @param   y       a befoglaló tégla bal felső sarkának sora
     */
    public void siklóKilövő(boolean [][] rács, int x, int y) {

        rács[y+ 6][x+ 0] = ÉLŐ;
        rács[y+ 6][x+ 1] = ÉLŐ;
        rács[y+ 7][x+ 0] = ÉLŐ;
        rács[y+ 7][x+ 1] = ÉLŐ;

        rács[y+ 3][x+ 13] = ÉLŐ;

        rács[y+ 4][x+ 12] = ÉLŐ;
        rács[y+ 4][x+ 14] = ÉLŐ;

        rács[y+ 5][x+ 11] = ÉLŐ;
        rács[y+ 5][x+ 15] = ÉLŐ;
        rács[y+ 5][x+ 16] = ÉLŐ;
        rács[y+ 5][x+ 25] = ÉLŐ;

        rács[y+ 6][x+ 11] = ÉLŐ;
        rács[y+ 6][x+ 15] = ÉLŐ;
        rács[y+ 6][x+ 16] = ÉLŐ;
        rács[y+ 6][x+ 22] = ÉLŐ;
        rács[y+ 6][x+ 23] = ÉLŐ;
        rács[y+ 6][x+ 24] = ÉLŐ;
        rács[y+ 6][x+ 25] = ÉLŐ;
```

```
        rács[y+ 7][x+ 11] = ÉLŐ;
        rács[y+ 7][x+ 15] = ÉLŐ;
        rács[y+ 7][x+ 16] = ÉLŐ;
        rács[y+ 7][x+ 21] = ÉLŐ;
        rács[y+ 7][x+ 22] = ÉLŐ;
        rács[y+ 7][x+ 23] = ÉLŐ;
        rács[y+ 7][x+ 24] = ÉLŐ;

        rács[y+ 8][x+ 12] = ÉLŐ;
        rács[y+ 8][x+ 14] = ÉLŐ;
        rács[y+ 8][x+ 21] = ÉLŐ;
        rács[y+ 8][x+ 24] = ÉLŐ;
        rács[y+ 8][x+ 34] = ÉLŐ;
        rács[y+ 8][x+ 35] = ÉLŐ;

        rács[y+ 9][x+ 13] = ÉLŐ;
        rács[y+ 9][x+ 21] = ÉLŐ;
        rács[y+ 9][x+ 22] = ÉLŐ;
        rács[y+ 9][x+ 23] = ÉLŐ;
        rács[y+ 9][x+ 24] = ÉLŐ;
        rács[y+ 9][x+ 34] = ÉLŐ;
        rács[y+ 9][x+ 35] = ÉLŐ;

        rács[y+ 10][x+ 22] = ÉLŐ;
        rács[y+ 10][x+ 23] = ÉLŐ;
        rács[y+ 10][x+ 24] = ÉLŐ;
        rács[y+ 10][x+ 25] = ÉLŐ;

        rács[y+ 11][x+ 25] = ÉLŐ;

    }
    /** Pillanatfelvételek készítése. */
    public void pillanatfelvétel(java.awt.image.BufferedImage felvetel) {
        // A pillanatfelvétel kép fájlneve
        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("sejtautomata");
        sb.append(++pillanatfelvételSzámláló);
        sb.append(".png");
        // png formátumú képet mentünk
        try {
            javax.imageio.ImageIO.write(felvetel, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
    // Ne villogjon a felület (mert a "gyári" update()
    // lemeszelné a vászon felületét).
    public void update(java.awt.Graphics g) {
```

```
        paint(g);
    }
    /**
     * Példányosít egy Conway-féle életjáték szabályos
     * sejttér obektumot.
     */
    public static void main(String[] args) {
        // 100 oszlop, 75 sor mérettel:
        new Sejtautomata(100, 75);
    }
}
```

A címből kiderült, hogy John Horton Conway alkotta meg ezt a játékot 1970-ben. Az addigi játékokhoz képest ez eltérő volt mivel ehez a játékhoz nem kellett felhasználó hanem akár elég volt egy számítógép is. Ez nem teljesen igaz mert a felhasználónak volt egy dolga mégpedig beállítani a kezdő pozíciót ahonnan indul maga a játék. Maga a játéktér egy saktáblához hasonlít ami bármekkora méretű lehet. Egy cellának 8 szomszédos cellája van. A szabályok : Ha a cella üres ezesetben a mellette lévő cella is üres lesz kivéve egy esetben amikor a 8-ból pontosan 3db cella foglalt. Ezesetben a következő cella is foglalt lesz. Következő szabály mikor egy cella foglalt, ezesetben a következő cella is foglalt lesz kivéve ha 2 vagy 3db cella foglalt, ekkor üresek lesznek a következők. Maga a játék kitalálója ezeket a cellákat élő illetve holt állípótúaknak títulálta a foglalt és üres helyett.

7.3. Qt C++ életjáték

Most Qt C++-ban!

A megoldás forrása https://sourceforge.net/p/udprog/code/ci/master/tree/source/labor/Qt/Sejtauto/

Ennél a feladatnál ugyan az a lényeg mint az elözőnél csak míg annál JAVÁ-ban írtuk meg itt C++-ban kell megírni ezt. Maga a játék értelmezése és a szabályzatok detto ugyan azok. Annyi változás van még, hogy itt a QT-t kell segítségül hívni. Maga a QT egy keretrendszeres alkalmas amit GUI-s alkalmazások és nem GUI-s programoknál használnak.

7.4. BrainB Benchmark

Megoldás forrása: https://github.com/nbatfai/esport-talent-search

A nevébúl adódoan az agyat veszi igénybe ez a "játék". Az e-sport területén használandó ez a szoftver mivel az adott játékos reakcióképességét képes felismeri és visszaadni. A játék lényege, hogy "Samu Entropy" karakterünket kell figyelnünk és, hogy a kék körben kell tartanunk a kurzorunkat. Ahogyan sikerül "level upolni" úgy egyre több doboz jelenik meg a képernyőn illetve ha ez nem lenne elég egyre gyorsabban is fognak mozogni. A játék során tudunk előre lépni illetve akár hátra is ha nem sikerült teljesíteni az adott "szintet". Elég egyszerű és könnyen használható kis program ami ad egy virtuális képet a reakcióképességedről ami az e-sportolók körében elég izgalmas téma.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

A megoldás forrása Bátfai Norbert tulajdona.

A szoftver lényege, hogy kézzel írott számjegyeket ismerjen fel a számítógép. Mesterséges inteligencia. Ebben azt esetben MNIST adatbázisból vett számjegyeket fogunk felismertetni vele. Mivel most használjuk először a "Python-t" és a "TensorFlow-t" ezért ezt most telepítenünk kell. Maga "TensorFlow" nagyon hasznos program, sok nagyobb cég is nem hiába használja. Maga a gépi tanulásban játszik fontos szerepet. Jelen esetben ezt a kód csípetet nevezhetjük a "TensorFlow" Hello Worldjének. Ez az első programunk célja, hogy összeszorozzon két számot neurális hálokk felhasználásával. Mivel most kezdjük a Python használatát érdemes tudni, hogy nem "{" és "}" adják meg a blokkokat hanem tabulárotoros behúzások.

8.2. Mély MNIST

Python

Megoldás videó:

1. passzolási lehetősségem (előadás)

8.3. Minecraft-MALMÖ

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

Megoldás forrása: https://bhaxor.blog.hu/2018/10/28/minecraft_steve_szemuvege

2. passzolási lehetősségem (előadás)

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

3. passzolási lehetősségem (előadás)

9.2. Gimp Scheme Script-fu: króm effekt

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely megvalósítja a króm effektet egy bemenő szövegre!

Megoldás videó: https://youtu.be/OKdAkI_c7Sc

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

4. passzolási lehetősségem (előadás)

9.3. Gimp Scheme Script-fu: név mandala

Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

Megoldás videó: https://bhaxor.blog.hu/2019/01/10/a_gimp_lisp_hackelese_a_scheme_programozasi_nyelv

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

Mint ahogy a feladat nevéből is kiderül a feladatunk nem más lesz mint, egy mandala kirajzolása a GIMP programban. Annyi a kikötés, hogy nem lehet grafikus felületet használni hanem kóddal kell dolgoznunk. Mint ahogy láthatjuk a mandala egy szimetrikus kör alapú kép. Először a szöveg hosszás és méretét határozzuk meg. Csak ezek után jön maga a mandala kirajzolása. Egy rétegben létrejön a szöveg és a szöveg betűtípusa is amit maga a felhasználó állított be. Amiért szimetrikus kör alapú képnek nevezzük az azért van mert mikor létrejött a réteg tükrözi a program majd elfordítja és úgy is tükrözi. Így jön létre a szimetrikus kör alapú mandala. A legvégén mint eddig is már csak a kíratás / kirajzolás jelen esetben a kirajzolás marad.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

5. passzolási lehetőségem (előadás)

10.2. Programozás bevezetés

A faceebookon kiírt kutatásban való résztével 1. passzolási lehetőségem

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

10.3. Programozás

A faceebookon kiírt kutatásban való résztével 2. passzolási lehetőségem

III. rész

Második felvonás

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

11. fejezet

Helló, Arroway!

11.1. A BPP algoritmus Java megvalósítása

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

11.2. Java osztályok a Pi-ben

Az előző feladat kódját fejleszd tovább: vizsgáld, hogy Vannak-e Java osztályok a Pi hexadecimális kifejtésében!

Megoldás videó:

Megoldás forrása:

Tanulságok, tapasztalatok, magyarázat...

IV. rész

Irodalomjegyzék

11.3. Általános

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

11.4. C

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

11.5. C++

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

11.6. Lisp

[METAMATH] Chaitin, Gregory, *META MATH! The Quest for Omega*, http://arxiv.org/PS_cache/math/-pdf/0404/0404335v7.pdf , 2004.

Köszönet illeti a NEMESPOR, https://groups.google.com/forum/#!forum/nemespor, az UDPROG tanulószoba, https://www.facebook.com/groups/udprog, a DEAC-Hackers előszoba, https://www.facebook.com/-groups/DEACHackers (illetve egyéb alkalmi szerveződésű szakmai csoportok) tagjait inspiráló érdeklődésükért és hasznos észrevételeikért.

Ezen túl kiemelt köszönet illeti az említett UDPROG közösséget, mely a Debreceni Egyetem reguláris programozás oktatása tartalmi szervezését támogatja. Sok példa eleve ebben a közösségben született, vagy itt került említésre és adott esetekben szerepet kapott, mint oktatási példa.