Hoppá, Error

Z3r0 t0 h3r0 edition

Ed. BHAX, DEBRECEN,
2019. május 9., v. 0.2.2

COLLABORATORS

	TITLE : Hoppá, Error	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert Ã©s Ratku, Dániel	2019. május 9.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	Aktualizálás, javítások.	nbatfai
0.0.5	2019-03-01	A Helló, Turing csokor megkezdése és a dokumentum formájának a személyre szabása és a saját repository-ba való feltöltés.	ratku.dani
0.0.6	2019-03-04	A Helló, Turing csokor nyolc feladatából az első öt feladatának befejezése.	ratku.dani

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
0.0.7	2019-03-05	A Helló, Turing csokor befejezése és a Helló, Chomsky feladatcsokor átfutása.	ratku.dani
0.0.8	2019-03-08	A Helló, Chomsky feladat csokor megkezdése, és a github frissítése.	ratku.dani
0.0.9	2019-03-11	A Helló, Chomsky csokor befejezése, és a teljes feladatsor feltöltése githubra.	ratku.dani
0.1.0	2019-03-15	A Helló, Caesar feladatok elkezdése.	ratku.dani
0.1.1	2019-03-17	A Helló, Caesar csokor felének a kidolgozása.	ratku.dani
0.1.2	2019-03-18	A Helló, Caesar feladatcsokor teljes elkészítése.	ratku.dani
0.1.3	2019-03-25	A Helló, Mandelbrot csokor elkészítése és a Gutenberg elkezdése.	ratku.dani
0.1.4	2019-03-30	A Helló, Welch elkészítése.	ratku.dani
0.1.5	2019-04-04	A Helló, Mandelbrot csokor majdnem teljesen kész.	ratku.dani
0.1.6	2019-04-06	A Helló, Mandelbrot csokor képekkel is kidolgozva kész.	ratku.dani
0.1.7	2019-04-10	A Helló, Schwarzenegger csokor elkészítése.	ratku.dani

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.1.8	2019-04-20	A Helló, Chaitin csokor még nem teljes kidolgozása, és az eddig leírt könyv struktúrális ellenőrzése.	ratku.dani
0.1.9	2019-04-25	A Helló, Chaitin csokor befejezése. Nyelvi helyességi vizsgálat a könyvben.	ratku.dani
0.2.0	2019-05-01	Eddigi munka teljes leellenőrzése a licence-s linkelésekkel és helyesírás javítással.	ratku.dani
0.2.1	2019-05-05	Összes feladatcsokor véglegesítése és ellenőrzése és a tutoráljaim és tutorálóim hivatalos rögzítése.	ratku.dani
0.2.2	2019-05-09	Helló, Gutenberg csokor befejezése. A dokumentum teljes körű végső finomítása és átadása.	ratku.dani

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?

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

1.2. Milyen doksikat olvassak el?

- Kezd ezzel: http://esr.fsf.hu/hacker-howto.html!
- Olvasgasd aztán a kézikönyv lapjait, kezd a **man man** parancs kiadásával. A C programozásban a 3-as szintű lapokat fogod nézegetni, például az első feladat kapcsán ezt a **man 3 sleep** lapot
- C kapcsán a [KERNIGHANRITCHIE] könyv adott részei.
- C++ kapcsán a [BMECPP] könyv adott részei.
- Az igazi kockák persze csemegéznek a C nyelvi szabvány ISO/IEC 9899:2017 kódcsipeteiből is.
- Amiből viszont a legeslegjobban lehet tanulni, az a The GNU C Reference Manual, mert gcc specifikus és programozókra van hangolva: szinte csak 1-2 lényegi mondat és apró, lényegi kódcsipetek! Aki pdf-ben jobban szereti olvasni: `https://www.gnu.org/software/gnu-c-manual/gnu-c-manual.pdf`
- Az R kódok olvasása kis általános tapasztalat után automatikusan, erőfeszítés nélkül menni fog. A Python nincs ennyire a spektrum magától értetődő végén, ezért ahhoz olvasd el a [BMECPP] könyv - 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket nézzek meg?

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

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

1.4. Tutorálóim:

- Pankotai Kristóf

  A feladatok a következőek voltak: 2.2; 2.3; 2.4; 2.6; 3.1; 3.3; 3.5; 3.6; 9.1; 9.3;
- Pataki Donát

  A feladatok a következőek voltak: 3.3; 7.2; 7.4; 8.1;

1.5. Tutoráltjaim:

- Pankotai Kristóf

  A feladatok a következőek voltak: 2.1; 2.5; 2.7; 2.8; 3.4; 6.1; 6.3; 7.2; 8.2;
- Pataki Donát

  A feladatok a következőek voltak: 2.8;

II. rész

Tematikus feladatok

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

2. fejezet

Helló, Turing!

2.1. Végtelen ciklus

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

Megoldás forrása:

Egy magot altató program.

Egy magot 100%-ban kihasználó program..

Minden magot 100%-ban kihasználó program.

A feladat során, az egy magot 100%-osan kihasználó programban, egy olyan do ciklust használtam, amely sosem éri el az abban megadott értéket, ezáltal a végtelenségig futva terheli a processzort. Az összes magot 100%-ban leterhelő program során, meg kellett hívnunk az omp könyvtárat, amellyel a "#pragma omp parallel"-t, amivel elérjük, hogy párhuzamosan terheljük a processzor összes szálát. Emellett egy olyan paraméterek nélküli for ciklust, amellyel megkapjuk a kívánt kihasználtsági szintet. A magot altató programhoz meg kellett hívni az unistd.h nevezetű könyvtárat, amivel tudjuk már használni a nem standard sleep függvényt, ami úgy működik, hogy a bele implantált érték ideéig leállítja a processzor egy szálát. Az egész feladatot végtelen ciklusokkal oldottam meg.

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

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

Megoldás videó:

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

```
Program T100
{

  boolean Lefagy(Program P)
```

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }
```

```
}
```

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

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

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

(A feladat megoldásában Pankotai Kristóf segített) Egy olyan szoftvert kell megírni, ami el tudja dönteni egy adott különböző programról, hogy az lefagy-e, azaz a végtelenségig tud-e futni. Ezt nem lehet elkészíteni, mivel olyan programot nem lehet írni, ami egy másik program végtelenségét vizsgálva megelőzné azt, és annak a végére érkezve eldönthetné, hogy ez egy befagyhatatlan program-e a végtelensége miatt. Ezen kívül megemlítendő, hogy olyan kódhalmazt nem lehet írni, ami magát, a másik program kódját elemezve tudná megállapítani a feladatban felvetett probléma megoldását.

2.3. Változók értékének felcserélése

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

Megoldás forrása:

Változókat felcserélő program.

(A feladat megoldásában Pankotai Kristóf segített)Ennek a feladatnak a megoldásához be kell kérni két számot, amelyet a megadott számokkal különböző műveleteket elvégezve megkapjuk ezeknek a sorrendben felcserélt változatát. Megemlítés képpen szeretném felhozni, hogy érdemes kisebb számokkal dolgozni az esetleges lehetséges legkisebb hibahatér megközelítése végett. A feladatban három művelet szerepel, amelyben a változókat adjuk és vonjuk ki egymásból úgy, hogy a kapott eredmény a feladatnak megfelelő módon jelenjen meg. Ez a feladat már szerepelt az első egyetemi féléves leckék között.

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.)

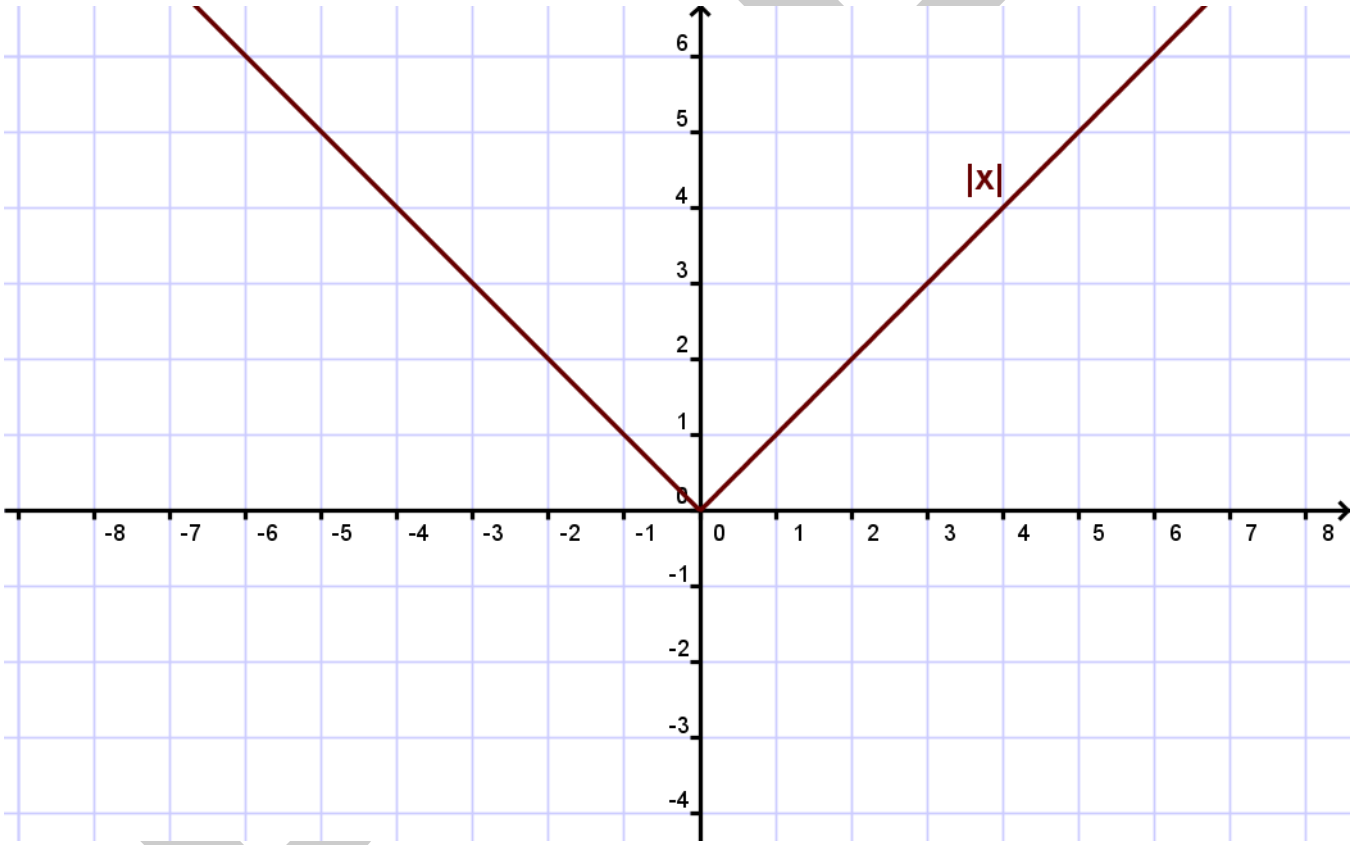
Megoldás forrása:

Labdapattogtató feladat megoldása.

Labdapattogtató feladat if nélkül.

Ennek a feladatnak a megoldásához meg kell hívnunk az unistd.h és a curses.h könyvtárat. A void azért kell bele, mert nem kér be semmit a program. Először is inícializálnunk kell az ablakot, amelynél figyelembe vesszük a felhasználó által kihúzott méretet. Deklaráljuk a szükséges változókat, például a kiinduló értéket és a haladás irányát. Majd egy végtelen ciklusban az ablak méretének a változását is figyelembe véve vetítjük ki a labda pattogását. Ezt folyamatosan frissítjük és a usleep-et használva megadva a sebességét halad a "labda". A pattogás menetét, azaz irányát if elágazásokkal határozzuk meg, szám szerint néggyel, amely tartalmazza a fel, le, jobbra, és a balra való irányokat.

Az if elágazások nélküli feladatnál általunk előre megadott ablak, azaz pályaméret generálunk. Ebben for ciklusokkal haladunk az előbb elmített tömbben, ezzel vizsgáljuk hol van a labda helye. A ciklus futásával halad a labda, ami ha eléri valamelyik tengely falát, akkor az adott iránnyal ellentétes irányba kezdi el a mozgást. Ekkor a folyamat előjele is változik. Ezáltal láthatjuk, hogy if-eket használva elég egyszerűen kivitelezhető a megoldás. Ezek nélkül viszont már eléggé komplikálttá változott a helyzetünk. Megoldás gyanánt a matematika irányába venném az irányt, ahol egy bizonyos függvény megoldást mutathat a munkákban, még pedig az abszolútérték lesz ez. Egy képet csatolva a feladathoz láthatjuk, hogy a menetét alapul véve, a saját koordinátáink tengelyére módosítva elérhetjük a kívánt hatást.



A kép, egy általános esetű abszolútérték függvényt ábrázol.

A kép forrása : https://archive.geogebra.org/en/upload/files/magyar/rezita/absx.html

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:

Szóhossz feladat megoldása.

Ebben a feladatban meg kellett vizsgálnunk az int típusnak az értékét a gépünkön. Ebben a programban két változóval kell dolgoznunk. Az egyik int értéke nulla, a másik pedig a 0x01, ami a következő sort jelöli. Ezután egy do while ciklussal növeljük a h értékét folyamatosan, egészen addig bitshift/bitwise-olnunk eggyel, amíg el nem jutunk az n által jelölt értékig. Ekkor a program megáll, és megkapjuk a bit értékét. Ebben az esetben balra bitwise-olásról van szó.

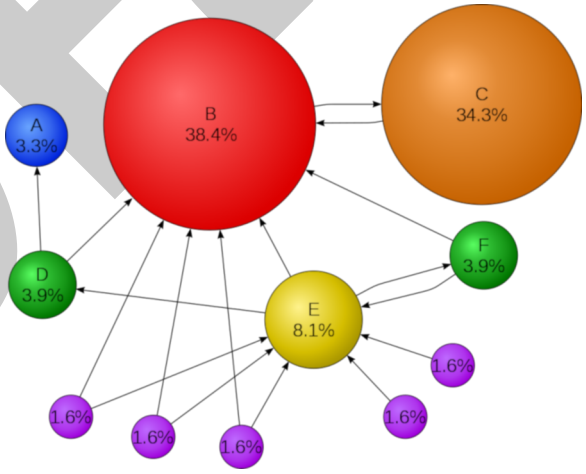
2.6. Helló, Google!

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

Megoldás forrása:

PageRank feladat.

Először is a cégről pár szót, ha már a mai tech világ egyik, ha nem a leghatalmasabb birodalmáról beszélünk. Az alapítóik, Larry Page és Sergey Brin még az egyetemen ismerkedtek meg, ahol megfogalmazódott bennük, hogy létrehozzák a világ legjobb keresőmotorját/algoritmusát, amelyből 1998-ra egy vállalat nőtte ki magát. Ennek az algoritmusnak a céja az oldalak "jó"-ságának bemutatása, ezalatt értve a minőségét, amit az adott oldalra mutató linkek száma határozza meg. Ezáltal a ranglistán egyre előrébb kerülve, a keresőben is elsőbbséget élvezhet az adott weboldal. Összefoglalva az elméletét a lényege, hogy egy hierarchikus rendszert építsen ki a weboldalak között, melyet az oldalak linkek általi mutatása, azaz szavazata határoz meg. A hozzá tartozó kódsorban egy tömbben fogjuk eltárolni a mutatásokat, azaz a szavazatokat az egyik oldalról a másikra. Ezen felül egy for ciklussal számoltatjuk ki ezeket az értékeket, ahol a "távolságot" is figyelembe véve próbálunk egyre pontosabb eredményt kapni, amelynek a végén a program futása után a végcélt láthatjuk. A futtatása során fontos kiemelni, hogy nem szabad elfeledkezni a végére illesztett -lm kapcsolóról, ami által feltudja ismerni a C nyelvű fordítónk, hogy hogyan is kéne működnie.



A képen a PageRank mögötti matematikai ábra.
https://hu.wikipedia.org/wiki/PageRank#/media/File:PageRanks-Example.svg

2.7. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

Megoldás forrása:

Ikerprím feladat.

Maga a tétel az ikerprímekről szól, amelyek definiálva úgy tekinthetőek, mint az egymást követő prímek amik között a különbség kettő. A feladat megoldása során deklaráljuk a prímeket egy adott számig, majd a differenciáljukat nézve, a nagyobból kivonjuk a kisebbet. Egy feltétellel a diferenciált keresünk a kettesre. Páronként deklaráljuk a kapott prímeket és a tételnek megfelelően reciprokkal szorozva adjuk össze őket egymással. Legvégül pedig a programon belül egy összesített értéket adunk vissza. Bezárólag pedig, matlab könyvtár által a megadott módon tudjuk ábrázolni a kapott eredményt. Futtatni a következő módon lehet: r -f [fájl neve].

2.8. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

Megoldás forrása:

A Monty Hall probléma megoldása.

Maga a Monty Hall probléma egy paradoxon, amelyben feltétel szerint van egy játékos, egy műsorvezető és a játékosnak három ajtó közül kell választania, amelyekből csak az egyik rejti a nyereményt. Miután a játékos választott egy ajtót, a műsorvezető kinyit egy olyan ajtót ami mögött nincs nyeremény, mivel hogy ő tudja melyik ajtó mit rejt. Ezek után megkérdezi a játékost, hogy szeretne-e a választásán módosítani, és ekkor jön a lényeg. A probléma azt tartalmazza, hogy a változtatással megnő a győzelem esélye 1/3-ról 2/3-ra. Ez azért van, mert azt feltételezzük, hogy a műsorvezető az általunk nem választott két ajtó közül, azért azt nyitotta ki, amelyiket, mert a másik mögött van a jutalom. A programunk úgy épül fel, hogy először is meghatározzuk a kísérletek számát, ami bármennyi lehet, majd létrehozzuk a kísérletek és a játékosok nevű változót, ami tartalmazza a választások számát egytől háromig, és a replace=T-vel pedig engedélyezzük, hogy a tippek száma többször is előfordulhasson a kísérletek során. A műsorvezető pedig a kísérletek számával lesz megadva. Egy for ciklussal megyünk végig a kísérleteken, ebbe viszont implementálnunk kell egy if elágazásokkal vizsgáljuk ki a választásokal. Az első if-ben a nyertes lehetőség értéke megegyezik a játékos tippjével, ekkor a műsorvezető a három lehetőségből kivonja a kísérlet értékét.Más esetben pedig, amikor nem találja el egyből a győztes ajtót, akkor a három lehetőségből kivonja a tippet és ami mögött a nyeremény van. Végül pedig kiírja az így kapott értéket. Következőnek pedig meghatározzuk, hogy változtat-e a játékos vagy nem. A nem esetén megegyezik a kísérlet a játékos tippjével. Ha pedig változtat, akkor egy for ciklussal kiválasztjuk azt, ami nem az eredeti választás, és nem is a műsorvezető által nyitott ajtó. A program pedig a kísérletek számának kiíratásával, a nem vátoztatással való nyereség és a változtatással való nyereség hányadosát, és a kísérletek számához való összes nyereség számát.

3. fejezet

Helló, Chomsky!

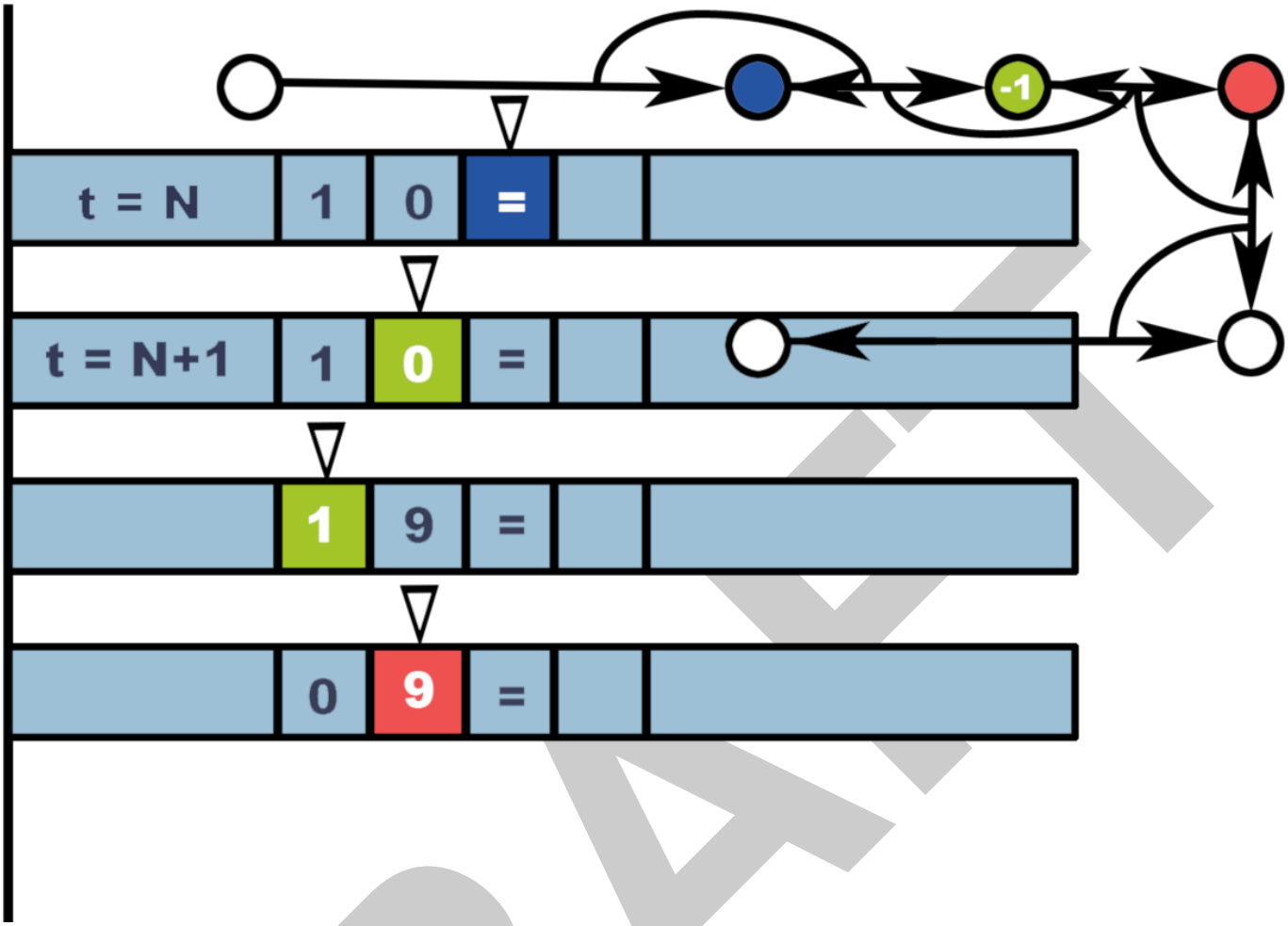
3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása:

Unárisba váltó Turing gép

A feladat lényege, hogy egy olyan programot hozzunk létre, ami a tízes számrendszerből, azaz decimálisból váltsunk át egyes számrendszerre, azaz unárisba pozitív egész számokat. A program, és elméletünk hátteréül, az előadáson áttárgyalt Turing gépet kell alapul vennünk, mint az ugyebár a lecke címében is szerepel. A folyamat során végülis egy for ciklussal megyünk végig a számon, ahol egymás után sorban megjelenített "I" vonalak fogják jelezni a számot, amit eredményül kaptunk. Először is meghívjuk az alap könvtárakat, majd deklarálunk két integer változót. Bekérünk egy egész számot, amit egy for ciklussal alakítunk át. Ami megy sorra, és húzza a vonalakat, majd ha a szám öttel osztható lesz maradék nélkül, akkor egy szóközt helyez le, hogy ezáltal is könnyebbé legyen téve az értelmezése.



A kép a turing gépet illusztrálja.

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:

(A feladat megoldása során a Dicső mentorom, Pataki Donát volt)

Első:

S (S $\rightarrow$ aXbc)

aXbc (Xb $\rightarrow$ bX)

abXc (Xc $\rightarrow$ Ybcc)

abYbcc (bY $\rightarrow$ Yb)

aYbbcc (aY $\rightarrow$aa)

aabbcc

Második:

S (S $\rightarrow$ aXbc)

aXbc (Xb $\rightarrow$ bX)

abXc (Xc $\rightarrow$ Ybcc)

abYbcc (bY $\rightarrow$ Yb)

aYbbcc (aY $\rightarrow$ aaX)

aaXbbcc (Xb $\rightarrow$ bX)

aabXbcc (Xb $\rightarrow$ bX)

aabbXcc (Xc $\rightarrow$ Ybcc)

aabbYbccc (bY $\rightarrow$ Yb)

aabYbbccc (bY $\rightarrow$ Yb)

aaYbbbccc (aY $\rightarrow$ aa)

aaabbbccc

A feladat egyik érdekessége mielőtt jobban belemennénk az, hogy ez a lecke prezenteltálja a leginkább a feladatcsokor címeként szolgáló személyt. Noam Chomsky, az MIT egyik professzora, aki egy hihetetlen fejlesztője volt a formális nyelvtan történetében. Jelen esetben nekünk a formális nyelvek elsődleges főkategóriáját kell használnunk, a generatív nyelvtant. Ez egy olyan szabályrendszer, amellyel jelsorozatokat lehet létrehozni, azaz lekreálni olyan céllal, hogy megadja a módját az átírási eljárásnak. A feladat megoldása során az előadás diasorán szereplő példázatot használtam fel, mivel tudtam, hogy ez tökéletesen működő modja a feladat megoldásához.

3.3. Hivatkozási nyelv

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

Megoldás forrása:

Nyelvi szabvány

(A feladat megoldásában Pankotai Kristóf.) A program a két nyelvi keret változását mutatja be. A C99 szabvány egyik legjelentősebb újítása az volt, hogy a felhasználó immáron képes a C++ nyelv típusú kommentelését használni. Egyéb újításnak vehető a for ciklus szabadabb használata.

3.4. Saját lexikális elemző

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

Megoldás forrása:

Saját lexikális elemző

Ez a program a lefuttatása során egy másik programot állít elő. A lefuttatás menete: lex -o program.c program.l . A C-s programuk fordításánál pedig szükség van a végére illeszteni egy -lfl tag-et. A működési elve az, hogy az L nyelv lexelve létrehozza azt a C programot, amely egy komplex karaktersorból kitudja szűrni a valós számokat. A programot a két dupla % (százalékjel) szedi három részre. Az első harmada tartalmazza a C programba kerülő részt. A középső része tartalmazza a szabályrendszert, és a C-s ciklust. A végső harmadban pedig a komplett main rész van implementálva.

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás forrása:

Saját l33t chiper

A programunk az L felépítés során, egy másik C forrást hoz létre, melyet a -lfl parancssal kell lefordítani. A kódsor lényege, hogy a betű- és számkészletünkhoz rendel hasonló karaktereket, melyet a futás során kicserél. Ezt úgy éri el, hogy minden betűhöz és számhoz hozzárendel egy négy tagú mátrixot, amelyben jelöljük az általunk vélt megfelelő hasonmás karaktereket. Ennek következményeként annak alapjául, ahány karaktert adunk meg a futtatás során, annak rengeteg számú változatát kaphatjuk vissza, mivel a mátrixok és a véletlenszerűen történő ebből való változás során rengeteg verzióban kaphatunk meg egyetlen egy mondatot.

3.6. A források olvasása

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

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

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

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

i.

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

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

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

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

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

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

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

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

i. -> Azt jelenti, ha eddig nem volt figyelmen kívül helyezve a SIGINT jel, akkor a jelkezelő függvény kezelje majd ezt a valamit le.

ii. -> Adott egy for ciklus, amely lényegi része, hogy nullától kezdve folyamatosan növelje az értékét eggyel, amíg négy nem lesz.

iii. -> Ugyan az mint az előző, szóval a for ciklus addig növeli a változó értékét, amíg az nulláról el nem éri a négyet.

iv. -> A célja az lenne, hogy nullától négyig a tömb valahanyadik (jelen esetben i-edik) elemének érékét növelje, viszont ez egy hibás kód, mert egyszerre van deklarálva minden, így nem tudjuk fordítani.

v. -> Ez a for ciklus addig megy, amíg az i kisebb mint az n. Majd hozzácsatolunk két mutatót, amihez szükség van a két további változóra.

vi. -> Kiíratjuk a printf paranccsal a függvényünk visszatérített értékét. A kiértékelés sorrendhibája miatt hibás a program.

vii. -> Két decimálist íratunk ki a printf-fel.

viii. -> Majdnem ugyan az mint az előbbi társa. Itt az "a" is változó lesz, amit az "f" fog felhasználni.

A feladat teljes kidolgozása során a kötelező olvasmányok függvényelemzési részletét alkalmaztam. Segítségül hívtam még Racs Tamás munkáját a megfogalmazásom tökéletesítésében.

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

Matlag

A feladat megértéséhez először is szükségünk van az elsőrendű logika ismeretére, ugyanis az adott példa alapján, ez négy logikai állításból áll. A legfontosabb megérteni az alap kifejezéseket. A foreall jelzi az univerzális kvantort, azaz a bármely-bármelyik, az exist, a létezik kifejezést jelöli. A wedge az implikáció és a supset pedig a konjukció és a neg pedig a negált. A program pedig AR nyelven íródott. Ezen alapvető logikai ismeretekkel már megoldható lesz a négy logikai állítás.

I. -> Bármely x esetén létezik olyan y, amelynél ha x kisebb, akkor y prím szám lesz.

II. -> Bármely x esetén létezik olyan y, amelynél ha x kisebb, akkor y prím szám lesz, és ha y prím szám, akkor annak második utánai is prím szám lesz.

III. -> Létezik olyan y, amelynél bármely x esetén az x prím ,és x kisebb, mint y.

IV. -> Létezik olyan y, amelynél bármely y kisebb x esetén az x nagyobb, és x nem prím.

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

-
```
int a;
```

Egy "a" elnevezésű int típusú változó.

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

  Egy, a "b" egészre mutató mutató ami az "a" változóra mutat.

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

  Egy memóriacímet tartalmazó r integer mutató mutató.

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

  Egy öt elemű integer típusú tömböt.

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

  Egy öt elemű integer típusú tömbre mutató mutató, ami a c-re mutat.

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

  Egy öt elemű integerekre mutató mutatóiból álló tömb.

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

  Egy integerrel visszatérő paraméterek nélkül álló függvényre mutató mutató.

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

  Egy integerre mutató mutatóval visszetérő, és paraméterek nélkül álló függvényre mutató mutató.

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

  Egy integerrel visszatérő, két integert váró függvényre mutató mutatóval visszatérő, két integert visszaváró függvény.

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

  Egy fuggvénymutató, ami egy integert visszaadó és két integert kapó függvényre mutató mutatót visszaadó, integert eredményül kapó függvényre.

Megoldás forrása:

A feladatok deklarálva

Tutorálóm: Pataki Donát, bár segítségül hívtam Nagy Lajost is.
  ```

4. fejezet

Helló, Caesar!

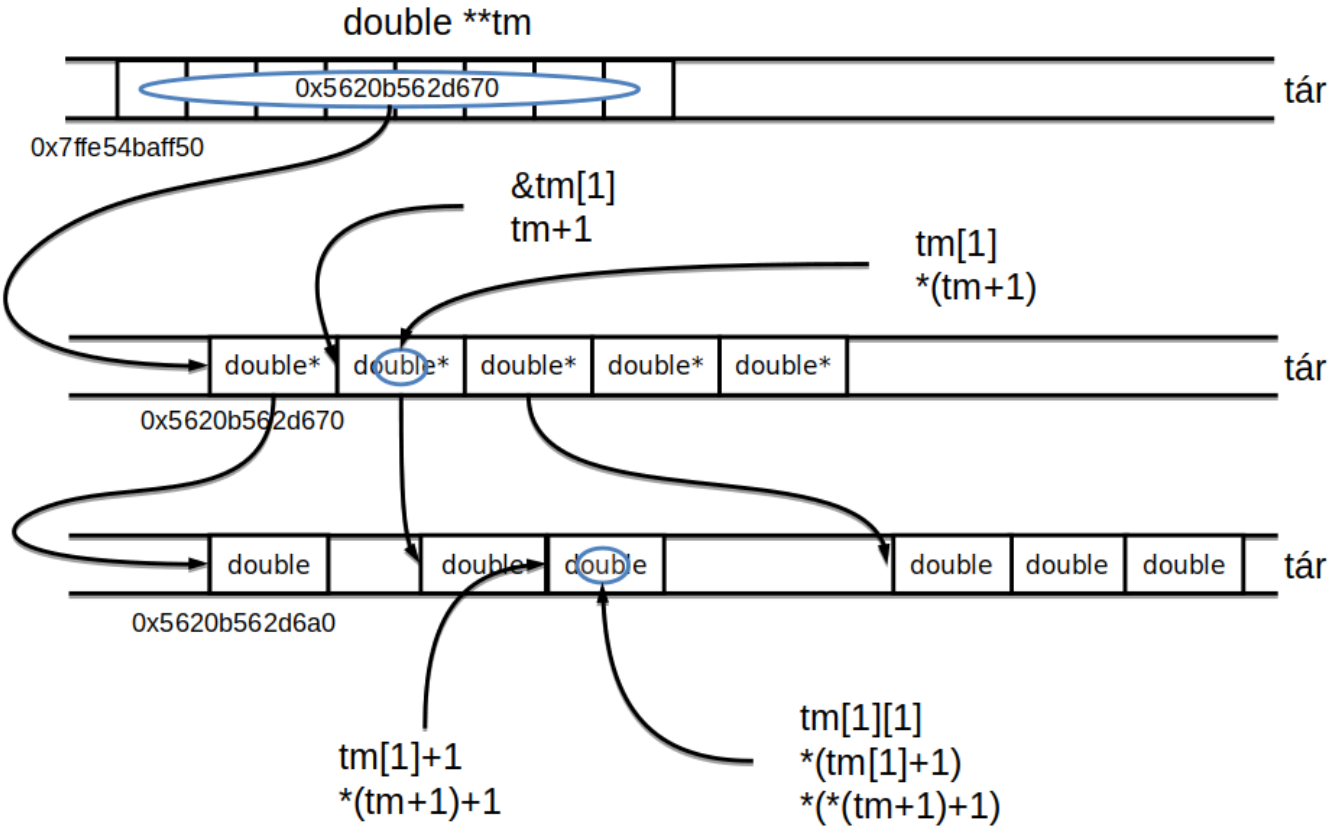
4.1. double ** háromszögmátrix

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

Megoldás forrása:

Mátrix megoldása

A programunk egy négyzetes mátrixról szó, amelynek a fő tulajdonsága az, hogy oszlopainak és sorainak száma megegyezik, és a főátlója alatt csupa nullákat kapunk, bár a kódunkban majd az átló alatti rész lesz csupa nulla. Először is deklaráljuk a két fő válrozónkat, ami a sorok száma lesz, és a tm mutatót, ami double típusú. Az if-ben szereplő részt megírva, a tm változó eredménye a malloc paranccsal megkapott tárterületet, ami a sor, szorozva 8 bájtnyi mérettel rendelkezik. A malloc-ot rá lehet kényszeríteni, hogy bármit visszaadjon, mert alapesetben void*-ot adna vissza. Ezután vizsgáljuk, hogy a malloc tud-e egyáltalán területet foglalni, mert ha nem, akkor a return -1-e visszaugrunk a program elejére. A lényegi rész ezután következik, ugyanis önmagával a malloc-cal a sikeres memóriafoglalás után megkapjuk a méretet, és a hozzájuk kapcsolódó mutató mutatót. Ehhez kell hasznosítanunk a feladat címében is szereplő pointer, azaz mutató egységet, a double**-ot. A program végén lévő for ciklus szegmens pedig feltölti a mátrixunk alsó felét.



*A kép, a double ** háromszögmátrixot ábrázolja.*
Forrás: Bátfai Norbert prezentációjából.

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása:

C Exor feladat.

C Exor feladat github-on.

Ez a feladat a titkosításról szól, melynek a kulcsa az EXOR, azaz a kizáró vagy logikai elem. A program működéséhez szükségünk van, egy bárki által elkészített szöveget tartalmozó fájlra, mondjuk egy txt-re, amit majd a program átkódol az általunk meghatározott kulccsal, ami egy nyolc karakteres sztringként lesz kezelve, ami azért fontos, mert a vissza kódoláshoz is ekkora hosszúságú kulcs kell, amivel majd a későbbiekben dekódolni is tudjuk. Maga a forrásnak a megírását a kulcs méretének és a buffer méretének meghatározásával kezdjük, amelyek konstansok lesznek. A main-be deklarálnunk kell a kulcs és a buffer tömböket, amik a szükséges kulcsot és az eredményt tárolják. A deklarált két int, a a kulcsunk aktuális részét/elemét mutatja, amivel végbe megy majd a művelet, és a második pedig, a beolvasott bájtoknak az összegét mondja meg. A harmadik int, azaz a kulcs_meret során használjuk strlen és a strncpy függvényekre, amelyeknek a lényege, hogy lerögzíti a hosszát stringben, amíg a ncpy pedig a végső másolatát rögzíti szintén stringben. Ezután egy végső while és for ciklussal folyamatosan olvasunk a bemenetről, és tároljuk a beolvasottakat a bufferben, amíg már nem tudunk több mindent beolvasni. Ekkor a ciklusban az olvasó 0

értéket ad vissza, amivel a program véget ér. A program megírása után a gcc-vel lefordítom, majd a futtatáshoz a ./programneve kulcs <a szöveges fájl, amiből kódolunk> "a fájl, amibe a titkosított szövegrészlet kerül bele".

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

Megoldás forrása:

Ezen az oldalon található meg a megoldás.

Ebben az esetben ugyan azt akarjuk elérni, mint a C nyelvű titkosító helyzetében. Azaz hogy az általunk, vagy a felhasználó által meghatározott kulccsal egy szintén saját választott szöveges dokumentumot kódolunk le. A szerkezete és működése ezáltal azonos lesz, az előbb említett nyelvben megírt társával. Fordítani a javac programneve.java . Majd a program futtatásához a köverkező kell: java programneve kulcs > titkosított.szöveg . Egyéb nyelvi különbségek között kiemelendő, hogy itt a titkosítás egy külön erre a célra létrehozott osztályban lelhető meg, amit külön kell meghívni.

4.4. C EXOR törő

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

Megoldás forrása:

C Exor törés.

C Exor törés github-on.

Ez a program, az előbb megcsinált kódoló programunk által legenerált szöveges fájlt tudja dekódolni az általunk előzőleg megadott kulccsal, amely nyolc karakterből álló szting. A program alapvető definiálásból indul, ami a maximális szöveg terjedelmet, a bájtokban mért memória tárolást, a kulcs méretét, ami megint csak az említett nyolc karakterből álló sztring lehet, és végűl a szabadszoftverű operációs rendszerforrást. Ezek után az első változónk, az átlagos szóhossz, amit a beleimplementált for ciklussal a szóközök segítségével számítunk ki. Ebben az esetben az "sz" változó, a szóközök számát jelenti. A számítás végén a return utasítással úgy adjuk vissza az értéket, hogy elosztjuk a hosszúságot a szóközök megszámlált értékével. A következő szegmens a szöveg tisztaságának vizsgálata, amivel csökkentjük a törések potenciálját. A középső komponens maga az exor része lesz, amiben a for ciklussal bájtonként hajtkuk végre a műveletet. A benne levő "%" jel által lesz a kulcs mindig aktuális. Az elkövetkezendő részlet pedig a törés végrehajtása lesz. Ehhez be kell olvasni a titkos fájlt, amit egy while ciklussal érünk el addig, amíg csak van adat, ha már nincs, akkor a read 0 értékkel tér vissza, és leáll az utasítás. A rengeteg egybeágyazott for ciklussal elérjük, hogy az összes lehetséges kulcs álljon elő. Végül, ha sikerül az exortörés, kiírjuk a kulccsal a tiszta szöveget. A lefordítása alapesetű, és a futtatása megegyezik a titkosító programéval.

4.5. Neurális OR, AND és EXOR kapu

R

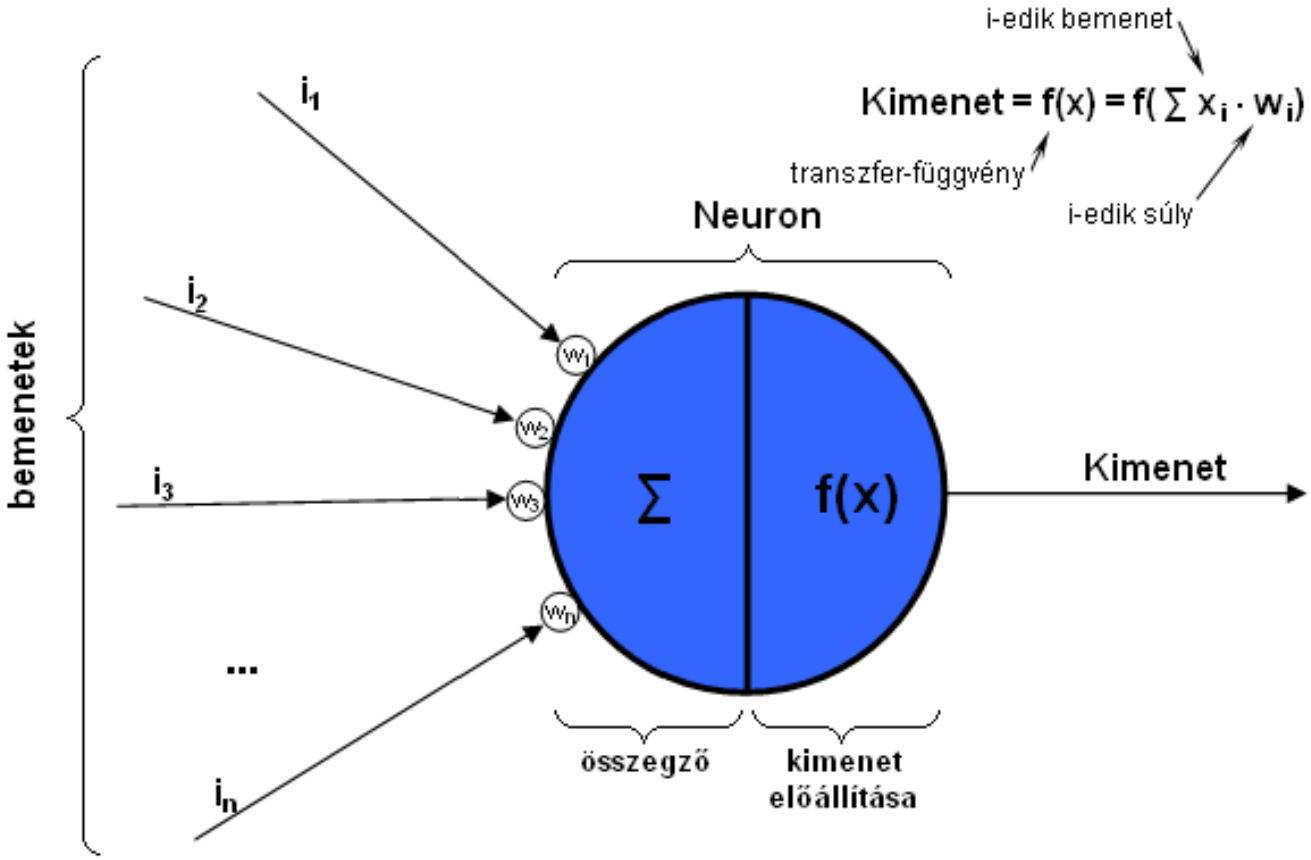
Megoldás forrása: https://gitlab.com/nbatfai/bhax/tree/master/attention_raising/NN_R

Neurális

A feladat megoldása során Pataki Donát segítségét kértem.

Maga a neurális hálózat, röviden definiálva a neuronok egymásba ágyazódott csoportja. A neuron leírása pedig az emberi idegrendszer legkisebb része. Ez egy olyan mesterségesen előállított neurális szerkezet, ahol a feladat címében is említett főleg logikai elemeket használjuk. Kezdetlegesen kiemelendő, hogy az OR és az AND abszolút logikai összekötőkként szolgálnak, azaz az OR lesz a VAGY tényező, amíg az AND a logikai ÉS-t jelenti. Itt és most, azt kell leszögezni, hogy a kapott értékek, azaz eredményvizsgálaté a fő szerep. Amennyiben az OR esetét tekintjük, akkor az eredményünk akkor lesz 1, ha mindkettő szegmens értéke 1. Ezzel szemben az AND logikai esetben, elég egy kitételnek, azaz csak az egyik ágazatnak értékének 1-nek lennie, hogy az eredmény is ennyi legyen. A végső eset, amiről az eddigi feladatok is szóltak, az EXOR-ról szól. Ebben az esetben pedig a kiértékelendő értékeknek különbözniük kell, (amely 0 és 1 lehet) hogy megkapjuk a célnak kitűzött egyest.

A típusait, azaz a rétegeit tekintve három különböző réteget tudunk megnevezni. Az első az input, azaz a bemeneti layer, ahol magába a neutrális hálózatunkba visszük be az adatot. Második layeré a kimeneti réteg, ami a környezetbe viszi át az adatot. Utolsó sorban, a harmadik layer, ami a rejtett szegmens, ahol végbe megy a kapcsolódás, tanulás és a folyamatok tömkelege.



A kép, a mesterséges neuron vázlatát kívánja bemutatni.
Forrás:
https://hu.wikipedia.org/wiki/F%C3%A1jl:Mesters%C3%A9ges_neuron_v%C3%A1zlatos_rajza.png.

4.6. Hiba-visszaterjesztéses perceptron

C++

Megoldás forrása:

Visszaterjesztés.

A feladat megoldásában Pataki Donát segített.

A hiba-visszaterjesztés nélkül perceptronról szeretnék bevezetésként regélni. Önmagában ez egy, az emberi elme, annak is a központjának, azaz az agy működésének elvét szeretné lemásolni. Maga, mint algoritmus, futása során eléggé hasonlatos az előbbi feladatban szereplő társához, a Neutrális hálozathoz. Ennek viszont a menetében eltérések szerepelnek, ugyanis itt súlyokat vezetünk be a mechanikába. A súlyok lényege, hogy az első lefutás utáni kapott értéket ezek által tudjuk módosítani azáltal, hogy az említett eset értékét növeljük, vagy csökkentjük. De hogy mikor és hogyan, mennyit kéne modifikálnunk rajta? Ekkor térünk ki a teljes feladat címre, a hiba-visszaterjesztése, ugyanis ezáltal lesz megvalósítva a feladat az általunk korábban említett harmadik layerben. Ez egy lineáris módon történő folyamat során addig fogja módosítgatni magának a súlyok értékét, amíg a hibát a legkisebb mértékig nem tudjuk leredukálni, azaz az elején bevitt értékig. A futtatásunk során a szokásos C++-os konzolos verziót használjuk: g++ mlp.hpp main.cpp -o "aminek elakarjuk nevezni" -lpng. A nem lefordulás esetén odailleszthetjük a végére, hogy: -std=c++11.

5. fejezet

Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

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

Megoldás forrása:

Mandelbrot halmaz kiszámolása.

Mielőtt a feladatban szereplő halmazzal dolgoznánk, tisztáznunk kell, hogy mi az a fraktál, és milyen kapcsolatban állnak a Mandelbrot-halmazzal. A fraktálok lényegében olyan alakzatok, melyek végtelenül komplexek. Két fő tulajdonságuk van, az egyik, hogy a legtöbb geometria alakzattal ellentétben a fraktálok szélei "szakadozottak", nem egyenletesek. A másik tulajdonságuk pedig, hogy nagyon hasonlítanak egymásra. Ha egy kör határfelületét folyamatosan nagyítjuk, egy idő után kisimul(a csúcsokat leszámítva), megkülönböztethetetlenné válik egy egyenestől. Ezzel szemben a fraktálok első tulajdonsága, mi szerint határfelöletük szakadozott, megmarad, függetlenül a nagyítás mértékétől. A Mandelbrot halmaz is a fraktálok közé tartozik. Ezt és a hozzá tartozó szabályt Benoit Mandelbrot fedezte fel 1979-ben. A halmaz komplex számokból áll, és az ezekből álló sorozat konvergens, azaz korlátos. Ezeket a számokat ábrázolva a komplex számsíkon kapjuk meg a Mandelbrot-halmaz híres farktálját.

Feladatbevezetésnek elsődlegesen is be kell vezetnem, hogy itt a komplex számokra, és fraktálokra lesz szükségünk. Miért is? Mert ahhoz, hogy a feladat során említett Mandelbrot halmazt megkaphassuk és láthassuk, deklarálnunk kell azokat a komplex számokat, amiket felakarunk használni azzal a kitétellel, hogy ezeknek a számoknak a hosszértéke egy általunk meghatározott számnak kell lennie. Ezáltal lesz a fraktálok soraiban is maga, az említett halmaz. Ehhez szükségünk van egy olyan vázolt részelemre, amin megfelelően tudjuk ábrázolni/kimutatni a kapott képet. A mögöttes komplexebb matematikai elméleti és gyakorlati hátteret nem tudom felvázolni a kellő tudás ismerete hiányában, szóval a bevezetés és általános tudnivalók tisztázása ezzel le is zárul.

```
            // mandelpngt.c++
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
```

```
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
//  Mandelbrot png
//  Programozó Páternoszter/PARP
//   https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↩
   _01_parhuzamos_prog_linux
//
//  https://youtu.be/gvaqijHlRUs
//
#include <iostream>
#include "png++/png.hpp"
#include <sys/times.h>

#define MERET 600
#define ITER_HAT 32000

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

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

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

    // a számítás
    float dx = (b - a) / szelesseg;
    float dy = (d - c) / magassag;
    float reC, imC, reZ, imZ, ujreZ, ujimZ;
    // Hány iterációt csináltunk?
    int iteracio = 0;
    // Végigzongorázzuk a szélesség x magasság rácsot:
    for (int j = 0; j < magassag; ++j)
    {
        //sor = j;
        for (int k = 0; k < szelesseg; ++k)
        {
```

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

                ++iteracio;

            }

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

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

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

}

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

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

```
    int kepadat[MERET][MERET];

    mandel(kepadat);

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

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

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

}
```

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

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

Megoldás forrása:

Mandelbrot halmaz std::complex osztállyal.

Itt, az előző feladattal ellentétben, hála a címben is szereplő `std::complex` osztályának hála a ciklizálásunk sokkalta tisztább, és egy jóval egyszerűsítettebb szegmenst kapunk eredményül. A futtatás képletét és menetét az alább beillesztett kódsor is tartalmazza. A kódok beillesztése azért történt most az eddigiektől különböző módon, hogy szemléletesebb legyen a két hasonló tartalmú program gyakorlati leírásának az ellentéte. Ez is megmutatja, hogy akár egyetlen egy új header fájl beimplementálásával mennyivel megtudjuk könnyíteni a dolgokunkat a kódolási részben. Ezáltal a kódunk kifinomultabb, tisztább és ezáltal átláthatóbb is.

```
            // Verzio: 3.1.2.cpp
// Forditas:
```

```
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040  ↩
   -0.019473810573093663922605855987058021128 18  ↩
   -0.019473810572541341845642648422654019668 7  ↩
   0.798505756933826860155534177465597167611 1  ↩
   0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913 1  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327099751215428 1  ↩
   0.213538705180497528912653137922461610287 4
// Nyomtatas:
// a2ps 3.1.2.cpp -o 3.1.2.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// ps2pdf 3.1.2.cpp.pdf 3.1.2.cpp.pdf.pdf
//
//
// Copyright (C) 2019
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.


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

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

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

```
double d = 1.3;

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

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

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

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

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

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

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

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

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

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

```
            ++iteracio;
          }

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

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

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

}
```

5.3. Biomorfok

Megoldás forrása:

Biomorfok

Az internetes kutatások során kiderül,t hogy ez végülis majdnem egyenlő a Mandelbrot halmazzal, mivel hogy az utóbb említett halmaz tartalmazza az összes először említett halmazt. Mint fraktál, erre is érvényes az a szinte hihetetlen, sőt, nem csak szinte, hanem teljesen hihetetlen állítás, hogy akármennyiszer is nagyítunk rá, akár a végtelenségik próbálkozva se fogjuk látjuk meg a teljes képet. Ezt mellékesen megjegyezve még az első világháború ideje alatt kutatták ki matematikus kutatók a szépséges Francia honban. A feladat megoldása során, azaz a kódrészletet a licence-ben is szereplő weboldalról szereztem be. Az alapját az eredeti Mandelbrot-os program adta, ahol optimalizáltuk ennek a programnak a változóit, hogy szinergikus legyen az utóbb említett, letöltött verzióval.

```
            // Verzio: 3.1.3.cpp
// Forditas:
// g++ 3.1.3.cpp -lpng -O3 -o 3.1.3
// Futtatas:
// ./3.1.3 bmorf.png 800 800 10 -2 2 -2 2 .285 0 10
// Nyomtatas:
// a2ps 3.1.3.cpp -o 3.1.3.cpp.pdf -1 --line-numbers=1  --left-footer=" ↩
   BATF41 HAXOR STR34M" --right-footer="https://bhaxor.blog.hu/" --pro= ↩
   color
// ps2pdf 3.1.3.cpp.pdf 3.1.3.cpp.pdf.pdf
//
// BHAX Biomorphs
// Copyright (C) 2019
// Norbert Batfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
```

```
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↩
   Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

    int szelesseg = 1920;
    int magassag = 1080;
    int iteraciosHatar = 255;
    double xmin = -1.9;
    double xmax = 0.7;
    double ymin = -1.3;
    double ymax = 1.3;
    double reC = .285, imC = 0;
    double R = 10.0;

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

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

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

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

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

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

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

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

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

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

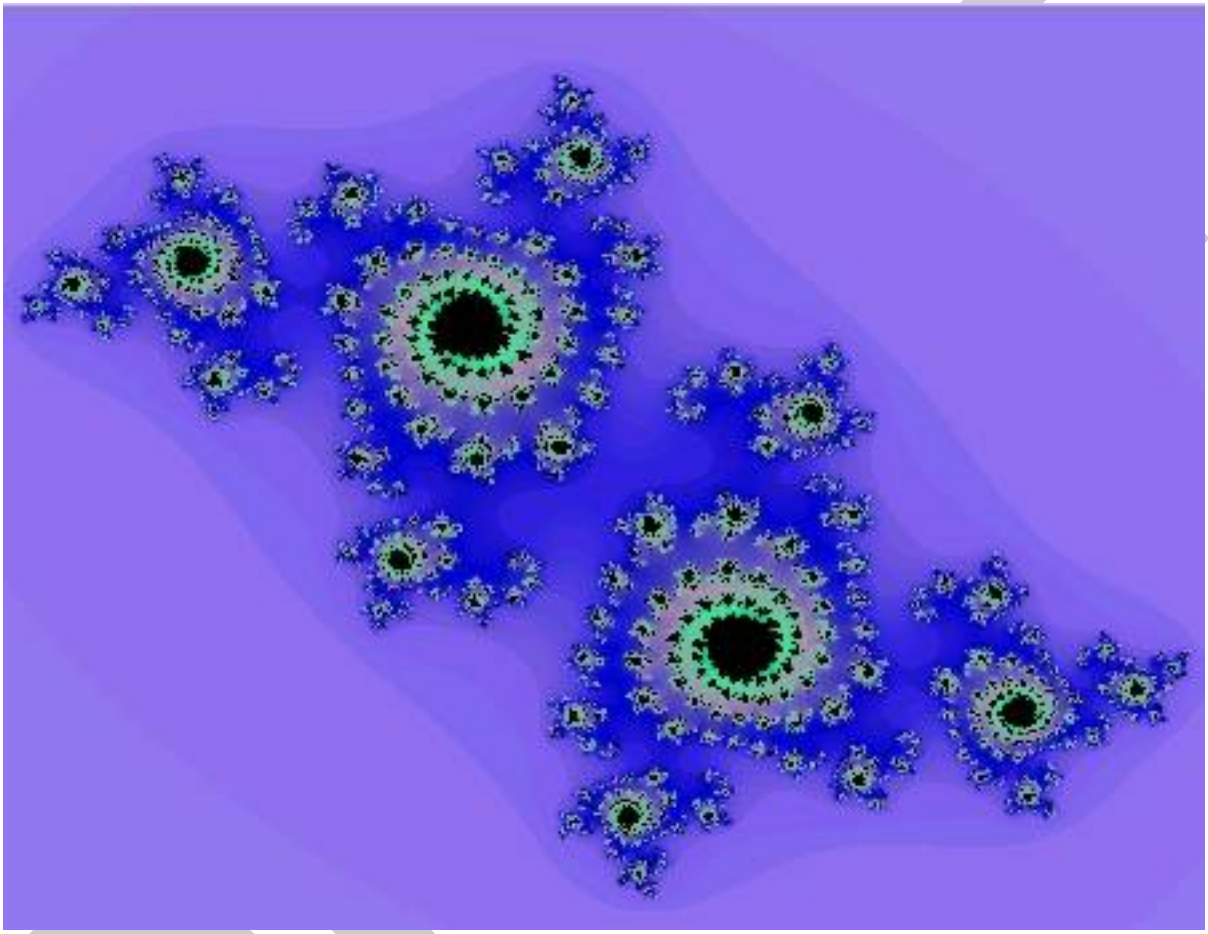
                z_n = std::pow(z_n, 3) + cc;
                //z_n = std::pow(z_n, 2) + std::sin(z_n) + cc;
                if(std::real ( z_n ) > R || std::imag ( z_n ) > R)
                {
                    iteracio = i;
                    break;
                }
            }

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

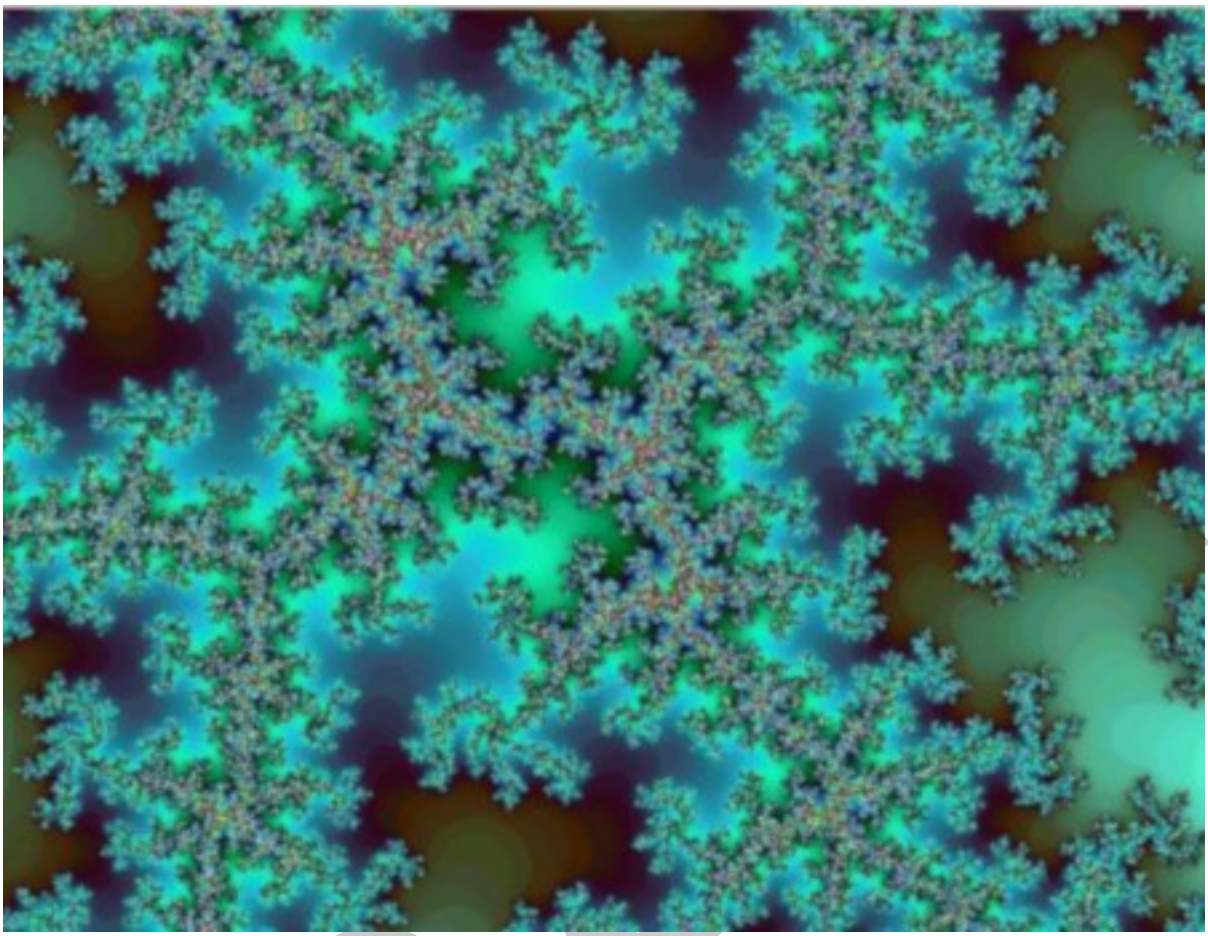
        int szazalek = ( double ) y / ( double ) magassag * 100.0;
        std::cout << "\r" << szazalek << "%" << std::flush;
    }
```

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

}
```



A kép, egy Júlia halmazt ábrázol, mindenféle bele nagyítás nélkül.
http://www.t-es-t.hu/minden/kaosz/mandel.htm



A kép, a már eléggé bele nagyított előbbi Júlia halmazt ábrázolja.
http://www.t-es-t.hu/minden/kaosz/mandel/halmaz2.htm

5.4. A Mandelbrot halmaz CUDA megvalósítása

Megoldás forrása:

CUDA megvalósítás.

Most pedig következzen a szokásos bevezetés, egy új dolog bevezetele, a CUDA miatt. Na de mi is akar ez lenni? Ez egy olyan algalmazásprogramozási felület, amit párhuzamos számítási platform gyanánt fejlesztett ki a az egyik legnagyobb számítógépes hardvare gyártó cég, akik a videókártyákra fókuszálnak, az Nvidia. Hatalmas kezdeményezés ez az ezen téren szoftvertfejlesztők körében, mivel olyan felületet biztosít számukra a munkájuk során, amellyel teljes hozzáférést biztosít a GPU teljes virtuálisan elérhető parancs készletéhez,és ez még nem minden. Ehhez jön hozzá a párhuzamos számításokhoz használatos ehhez szükséges magok vezérlése. Az ezzel használatos programozási nyelvek között szerepel a C és a C++ is. Maga a CUDA jelentése pedig a Compute Unified Device Architecture-ból ered, azaz Egységes Eszközarchitektúra Kiszámítás. A kódrészlet teljes bemásolása megint csak amiatt történik, hogy még tisztábban lehessen összevetni az előző Mandelbrot halmazos megoldásokkal. Emellett ez a megoldás gyorsabb lefutási időt is eredményez a jobb hardvare-es specifikus használat miatt.

```
            // mandelpngc_60x60_100.cu
// Copyright (C) 2019
```

```
// Norbert Bátfai, batfai.norbert@inf.unideb.hu
//
//  This program is free software: you can redistribute it and/or modify
//  it under the terms of the GNU General Public License as published by
//  the Free Software Foundation, either version 3 of the License, or
//  (at your option) any later version.
//
//  This program is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU General Public License for more details.
//
//  You should have received a copy of the GNU General Public License
//  along with this program.  If not, see <https://www.gnu.org/licenses/>.
//
// Version history
//
//  Mandelbrot png
//  Programozó Páternoszter/PARP
//   https://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0063 ↩
   _01_parhuzamos_prog_linux
//
//  https://youtu.be/gvaqijHlRUs
//

#include <png++/image.hpp>
#include <png++/rgb_pixel.hpp>

#include <sys/times.h>
#include <iostream>


#define MERET 600
#define ITER_HAT 32000

__device__ int
mandel (int k, int j)
{
  // Végigzongorázza a CUDA a szélesség x magasság rácsot:
  // most eppen a j. sor k. oszlopaban vagyunk

  // 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;

  // c = (reC, imC) a rács csomópontjainak
  // megfelelő komplex szám
  reC = a + k * dx;
  imC = d - j * dy;
  // z_0 = 0 = (reZ, imZ)
  reZ = 0.0;
  imZ = 0.0;
  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;

    }
  return iteracio;
}


/*
__global__ void
mandelkernel (int *kepadat)
{
  int j = blockIdx.x;
  int k = blockIdx.y;
  kepadat[j + k * MERET] = mandel (j, k);
}
*/

__global__ void
mandelkernel (int *kepadat)
{

  int tj = threadIdx.x;
  int tk = threadIdx.y;

  int j = blockIdx.x * 10 + tj;
```

```
  int k = blockIdx.y * 10 + tk;

  kepadat[j + k * MERET] = mandel (j, k);

}

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

  int *device_kepadat;
  cudaMalloc ((void **) &device_kepadat, MERET * MERET * sizeof (int));

  // dim3 grid (MERET, MERET);
  // mandelkernel <<< grid, 1 >>> (device_kepadat);

  dim3 grid (MERET / 10, MERET / 10);
  dim3 tgrid (10, 10);
  mandelkernel <<< grid, tgrid >>> (device_kepadat);

  cudaMemcpy (kepadat, device_kepadat,
        MERET * MERET * sizeof (int), cudaMemcpyDeviceToHost);
  cudaFree (device_kepadat);

}

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

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

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

  int kepadat[MERET][MERET];

  cudamandel (kepadat);

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

  for (int j = 0; j < MERET; ++j)
    {
```

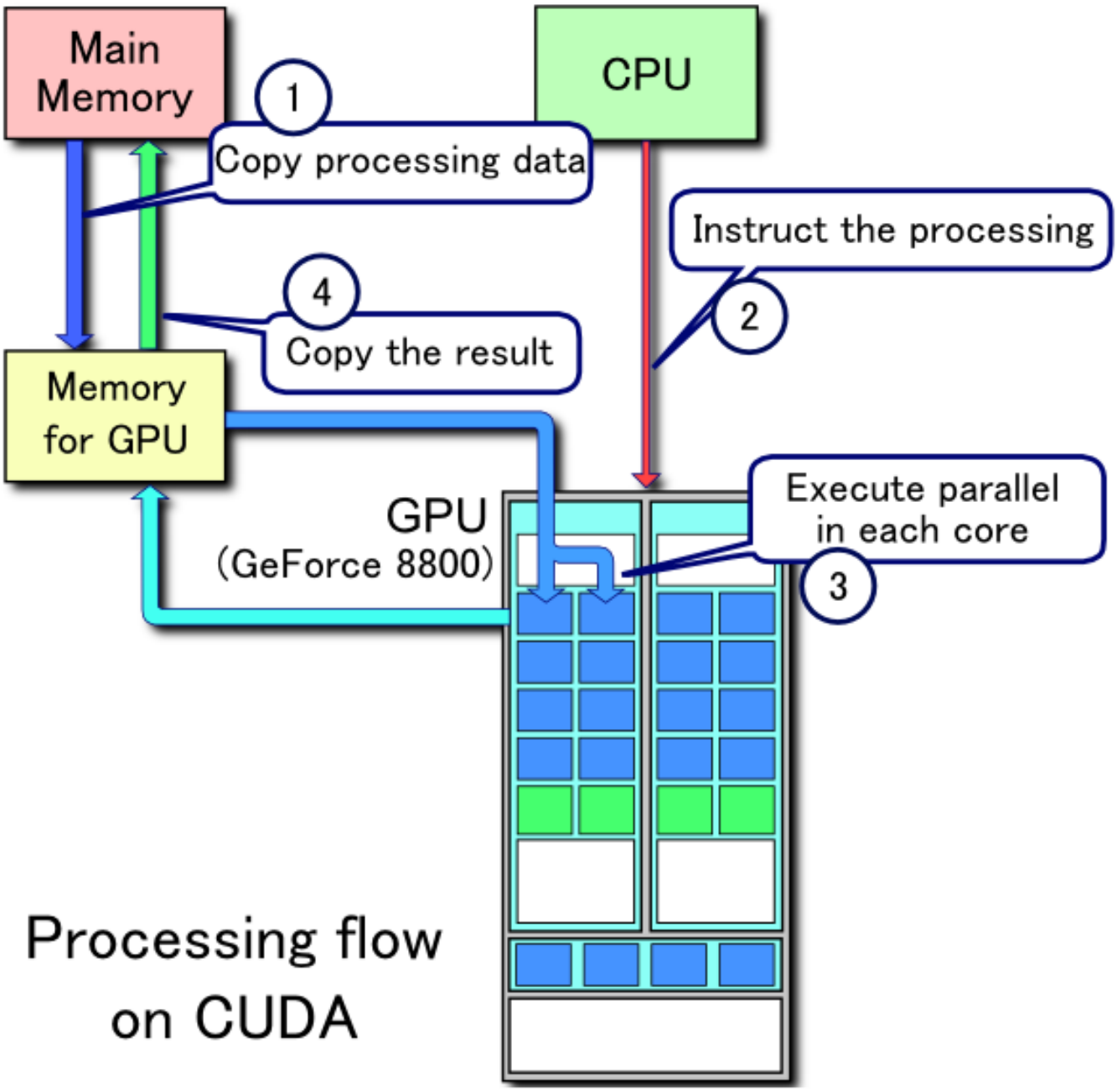
```
      //sor = j;
      for (int k = 0; k < MERET; ++k)
  {
    kep.set_pixel (k, j,
       png::rgb_pixel (255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT,
            255 -
            (255 * kepadat[j][k]) / ITER_HAT));
  }
    }
  kep.write (argv[1]);

  std::cout << argv[1] << " mentve" << std::endl;

  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;

}
```



A kép a CUDA folyamat ábráját mutatja be.
https://en.wikipedia.org/wiki/CUDA#/media/File:CUDA_processing_flow_(En).PNG

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:

A kódok forrása innen származik: UDPROG-os repó SourceForge-ról.

A program elmélete innen származik: Progpater.

Mandelbrot nagyító és utazó C++ nyelven.

Fontos tisztázni az elején, hogy kiemelt fontosságú a: sudo apt-get install libqt4-dev könyvtárat. Innentől folytatólagosan már majd hogy nem semmit sem értek. Önmagában a programmal azt tudjuk elérni a Qt felületét is használva, hogy általunk mozgatva is tudunk folytonos nagyítást eredményezni, ami olyan hatást kelt mintha egy egy örökké tartó gif-et néznénk ami a végtelenségbe tart, ami végülis igaz is abból a szempontból, hogy a fraktálos halmazunknak ez lenne a célja.

5.6. Mandelbrot nagyító és utazó Java nyelven

Megoldás forrása:

Fő forrás, ami alapján készült: A Tankönyvtári java-t tanítok részlege.

Mandelbrot Java-ban.

Ugyan az mint a C-s verziója csak a feladatban említett nyelvben.

A kódsor megint csak a megfelelő különbözeti ábrázolás miatt szerepel inkább a könyvben, a könnyebb egymáshoz hasonlítás végett a többi Mandelbrot-os halmazos feladattal. Itt kiemelendő még, hogy a futtatás után a beleimplementált algoritmussal létrehozzuk az osztályát is, ez által bővítve a forrásmappánk tartalmát.

Futtatása eléggé egyszerű. Annyit kell csak tennünk, hogy ugyebár a konzolt megnyitva a megfelelő mappába megyünk, és ott szépen beírjuk, hogy: javac MandelbrotHalmaz.Java. Itt nagyon fontos az, hogy a fájl neve a megadott példa szerint legyen, mert ez által tudja csak megfelelően megalkotni a kódsorban is meglakotott osztályt az alkotáshoz. Ezután sima: java MandelbrotHalmaz kifejezést használva elérjük, hogy lekreálja a Mandelbrot Halmazt, úgy ahogy mi akartuk, Java nyelven. Az oldal alapján, ahonnan származtatva lett a program meglátjuk, hogy akár az S billentyűgomb lenyomásával lekreálunk egy képet, amelyet abba a mappába tesz bele a program, amelyben a forráskód is szerepel. Erre lentebb láthatunk egy kész példát a kód alatt.

```
        /*
 * MandelbrotHalmaz.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A Mandelbrot halmazt kiszámoló és kirajzoló osztály.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class MandelbrotHalmaz extends java.awt.Frame implements Runnable {
    /** A komplex sík vizsgált tartománya [a,b]x[c,d]. */
    protected double a, b, c, d;
    /** A komplex sík vizsgált tartományára feszített
     * háló szélessége és magassága. */
    protected int szélesség, magasság;
```

```
    /** A komplex sík vizsgált tartományára feszített hálónak megfelelő kép ↩
       .*/
    protected java.awt.image.BufferedImage kép;
    /** Max. hány lépésig vizsgáljuk a z_{n+1} = z_n * z_n + c iterációt?
     * (tk. most a nagyítási pontosság) */
    protected int iterációsHatár = 255;
    /** Jelzi, hogy éppen megy-e a szamítás? */
    protected boolean számításFut = false;
    /** Jelzi az ablakban, hogy éppen melyik sort számoljuk. */
    protected int sor = 0;
    /** A pillanatfelvételek számozásához. */
    protected static int pillanatfelvételSzámláló = 0;
    /**
     * Létrehoz egy a Mandelbrot halmazt a komplex sík
     * [a,b]x[c,d] tartománya felett kiszámoló
     * <code>MandelbrotHalmaz</code> objektumot.
     *
     * @param      a              a [a,b]x[c,d] tartomány a koordinátája.
     * @param      b              a [a,b]x[c,d] tartomány b koordinátája.
     * @param      c              a [a,b]x[c,d] tartomány c koordinátája.
     * @param      d              a [a,b]x[c,d] tartomány d koordinátája.
     * @param      szélesség      a halmazt tartalmazó tömb szélessége.
     * @param      iterációsHatár a számítás pontossága.
     */
    public MandelbrotHalmaz(double a, double b, double c, double d,
            int szélesség, int iterációsHatár) {
        this.a = a;
        this.b = b;
        this.c = c;
        this.d = d;
        this.szélesség = szélesség;
        this.iterációsHatár = iterációsHatár;
        // a magasság az (b-a) / (d-c) = szélesség / magasság
        // arányból kiszámolva az alábbi lesz:
        this.magasság = (int)(szélesség * ((d-c)/(b-a)));
        // a kép, amire rárajzoljuk majd a halmazt
        kép = new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        // 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 's', 'n' és 'm' gombok lenyomását figyeljük
            public void keyPressed(java.awt.event.KeyEvent e) {
                if(e.getKeyCode() == java.awt.event.KeyEvent.VK_S)
```

```
                    pillanatfelvétel();
                // Az 'n' gomb benyomásával pontosabb számítást végzünk.
                else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_N) {
                    if(számításFut == false) {
                        MandelbrotHalmaz.this.iterációsHatár += 256;
                        // A számítás újra indul:
                        számításFut = true;
                        new Thread(MandelbrotHalmaz.this).start();
                    }
                // Az 'm' gomb benyomásával pontosabb számítást végzünk,
                // de közben sokkal magasabbra vesszük az iterációs
                // határt, mint az 'n' használata esetén
                } else if(e.getKeyCode() == java.awt.event.KeyEvent.VK_M) {
                    if(számításFut == false) {
                        MandelbrotHalmaz.this.iterációsHatár += 10*256;
                        // A számítás újra indul:
                        számításFut = true;
                        new Thread(MandelbrotHalmaz.this).start();
                    }
                }
            }
        });
        // Ablak tulajdonságai
        setTitle("A Mandelbrot halmaz");
        setResizable(false);
        setSize(szélesség, magasság);
        setVisible(true);
        // A számítás indul:
        számításFut = true;
        new Thread(this).start();
    }
    /**
     * A halmaz aktuális állapotának kirajzolása.
     */
    public void paint(java.awt.Graphics g) {
        // A Mandelbrot halmaz kirajzolása
        g.drawImage(kép, 0, 0, this);
        // Ha éppen fut a számítás, akkor egy vörös
        // vonallal jelöljük, hogy melyik sorban tart:
        if(számításFut) {
            g.setColor(java.awt.Color.RED);
            g.drawLine(0, sor, getWidth(), sor);
        }
    }
    // Ne villogjon a felület (mert a "gyári" update()
    // lemeszelné a vászon felületét).
    public void update(java.awt.Graphics g) {
        paint(g);
    }
    /**
```

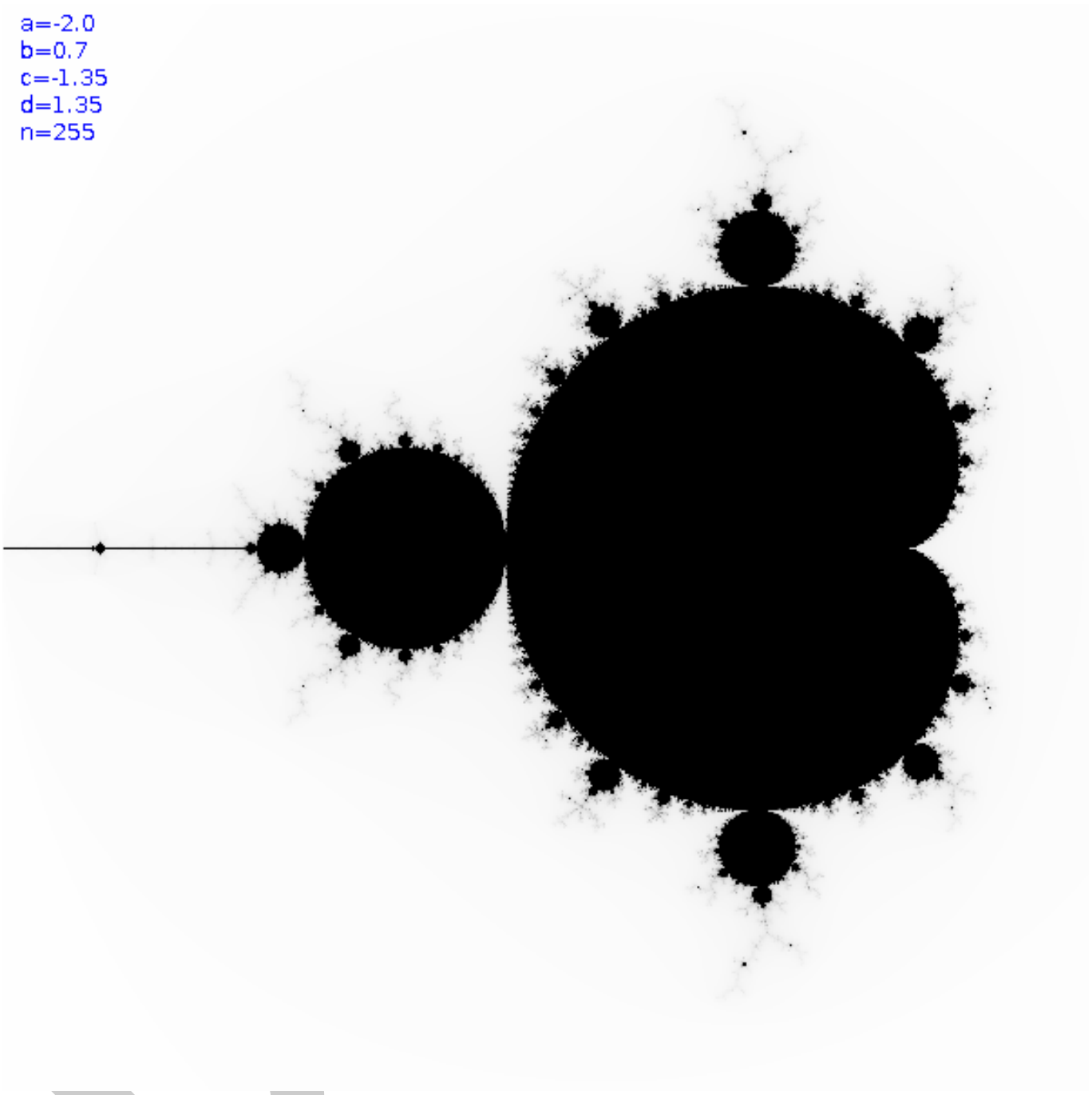
```
     * Pillanatfelvételek készítése.
     */
    public void pillanatfelvétel() {
        // Az elmentendő kép elkészítése:
        java.awt.image.BufferedImage mentKép =
                new java.awt.image.BufferedImage(szélesség, magasság,
                java.awt.image.BufferedImage.TYPE_INT_RGB);
        java.awt.Graphics g = mentKép.getGraphics();
        g.drawImage(kép, 0, 0, this);
        g.setColor(java.awt.Color.BLUE);
        g.drawString("a=" + a, 10, 15);
        g.drawString("b=" + b, 10, 30);
        g.drawString("c=" + c, 10, 45);
        g.drawString("d=" + d, 10, 60);
        g.drawString("n=" + iterációsHatár, 10, 75);
        g.dispose();
        // A pillanatfelvétel képfájl nevének képzése:
        StringBuffer sb = new StringBuffer();
        sb = sb.delete(0, sb.length());
        sb.append("MandelbrotHalmaz_");
        sb.append(++pillanatfelvételSzámláló);
        sb.append("_");
        // A fájl nevébe belevesszük, hogy melyik tartományban
        // találtuk a halmazt:
        sb.append(a);
        sb.append("_");
        sb.append(b);
        sb.append("_");
        sb.append(c);
        sb.append("_");
        sb.append(d);
        sb.append(".png");
        // png formátumú képet mentünk
        try {
            javax.imageio.ImageIO.write(mentKép, "png",
                    new java.io.File(sb.toString()));
        } catch(java.io.IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * A Mandelbrot halmaz számítási algoritmusa.
     * Az algoritmus részletes ismertetését lásd például a
     * [BARNSLEY KÖNYV] (M. Barnsley: Fractals everywhere,
     * Academic Press, Boston, 1986) hivatkozásban vagy
     * ismeretterjesztő szinten a [CSÁSZÁR KÖNYV] hivatkozásban.
     */
     public void run() {
        // A [a,b]x[c,d] tartományon milyen sűrű a
        // megadott szélesség, magasság háló:
```

```
        double dx = (b-a)/szélesség;
        double dy = (d-c)/magasság;
        double reC, imC, reZ, imZ, ujreZ, ujimZ;
        int rgb;
        // Hány iterációt csináltunk?
        int iteráció = 0;
        // Végigzongorázzuk a szélesség x magasság hálót:
        for(int j=0; j<magasság; ++j) {
            sor = j;
            for(int k=0; k<szélesség; ++k) {
                // c = (reC, imC) a háló rácspontjainak
                // megfelelő komplex szám
                reC = a+k*dx;
                imC = d-j*dy;
                // z_0 = 0 = (reZ, imZ)
                reZ = 0;
                imZ = 0;
                iteráció = 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 && iteráció < iterációsHatár) {
                    // z_{n+1} = z_n * z_n + c
                    ujreZ = reZ*reZ - imZ*imZ + reC;
                    ujimZ = 2*reZ*imZ + imC;
                    reZ = ujreZ;
                    imZ = ujimZ;

                    ++iteráció;

                }
                // ha a < 4 feltétel nem teljesült és a
                // iteráció < iterációsHatár sérülésével lépett ki, azaz
                // feltesszük a c-ről, hogy itt a z_{n+1} = z_n * z_n + c
                // sorozat konvergens, azaz iteráció = iterációsHatár
                // ekkor az iteráció %= 256 egyenlő 255, mert az esetleges
                // nagyítasok során az iteráció = valahány * 256 + 255
                iteráció %= 256;
                // így a halmaz elemeire 255-255 értéket használjuk,
                // azaz (Red=0,Green=0,Blue=0) fekete színnel:
                rgb = (255-iteráció)|
                        ((255-iteráció) << 8) |
                        ((255-iteráció) << 16);
                // rajzoljuk a képre az éppen vizsgált pontot:
                kép.setRGB(k, j, rgb);
            }
```

```
            repaint();
        }
        számításFut = false;
    }
    /**
     * Példányosít egy Mandelbrot halmazt kiszámoló obektumot.
     */
    public static void main(String[] args) {
        // A halmazt a komplex sík [-2.0, .7]x[-1.35, 1.35] tartományában
        // keressük egy 400x400-as hálóval:
        new MandelbrotHalmaz(-2.0, .7, -1.35, 1.35, 600, 255);
    }
}
```



A kép a feladat végeredményét mutatja be.

6. fejezet

Helló, Welch!

6.1. Első osztályom

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

Megoldás forrása:

Elméleti forrás: Java-s tankönyvtár.

Gyakorlati forrás: UDPROG repó.

Első osztályaim.

A feladat megvalósítása során megalkotott programunk feladata az, hogy a polár transzformációs algoritmussal dolgozzon. A program tíz alkalommal számol, és ez alapján, hogy tárol-e tag visszatérítést, akkor azt is figyelembe veszi.

Az első c++ osztályom bemutatása:

```
            #include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;

class Random
{

  public:

    Random();

    ~Random(){}

    double get();
```

```
  private:

    bool exist;
    double value;

};


Random::Random()
{
  exist = false;
  srand (time(NULL));
};


double Random::get()
{
  if (!exist)
  {
    double u1, u2, v1, v2, w;

    do
    {
      u1 = rand () / (RAND_MAX + 1.0);
      u2 = rand () / (RAND_MAX + 1.0);
      v1 = 2 * u1 - 1;
      v2 = 2 * u2 - 1;
      w = v1 * v1 + v2 * v2;
        }
    while (w > 1);

    double r = sqrt ((-2 * 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 < 2; ++i) cout << rnd.get() << endl;

}
```

Az első Java osztályom bemutatása:

```
            public class PolárTranszF {

    boolean létezik_tárolt = false;
    double tárolt;

    public PolárTranszF() {

        létezik_tárolt = false;

    }

    public double matek_rész() {

        if(!létezik_tárolt) {

            double u1, u2, v1, v2, w;
            do {
                u1 = Math.random();
                u2 = Math.random();

                v1 = 2*u1 - 1;
                v2 = 2*u2 - 1;

                w = v1*v1 + v2*v2;

            } while(w > 1);

            double r = Math.sqrt((-2*Math.log(w))/w);

            tárolt = r*v2;
            létezik_tárolt = !létezik_tárolt;

            return r*v1;

        } else {
            létezik_tárolt = !létezik_tárolt;
            return tárolt;
        }
    }

    public static void main(String[] args) {

        PolárTranszF g = new PolárTranszF();
```

```
        for(int i=0; i<2; ++i)
            System.out.println(g.matek_rész());

    }

}
```

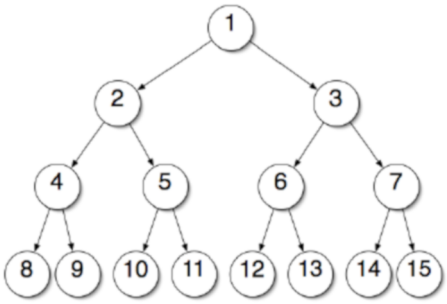
6.2. LZW

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

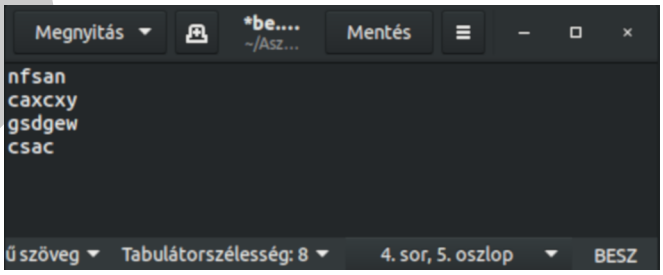
Megoldás forrása:

Fa építés.

Bevezetés gyanánt, a bináris fa egy adatszerkezet, amelyről annyit érdemes és fontos tudni, hogy minden egyes node-jának, azaz csomópontjának csak további két csomópontja, "becézve" gyereke lehet. Ezt tisztázva rájöhetünk, hogy ez által egy a végtelenségig futó fa struktúra szerű elemet hozunk létre. Ennek az alapja a Linux rendszerén, az UNIX-ra létrehozott fájltömörítésén alapszik. A mi esetünkben egy bemeneti szöveges fájlból fok a program egy bináris elemekre leosztott "fa" szerű képet létrehozni, így tárolva el benne az információt, azaz az adatot. Lentebb ábrázolni fogom az elméleti kialakítását, továbbá egy lefuttatott program során kialakított kiexportált eredményt egy másik erre a célre meghatározott szöveges fájlba.



A kép egy bináris fát ábrázol.
https://www.researchgate.net/figure/A-binary-tree-with-15-nodes-The-node-number-indicates-the-order-in-which-the-node-was_fig1_221496921



A kép a bemeneti txt fájl tartalmát ábrázolja.

A kép a kimenetet mutatja be a konzolon.

6.3. Fabejárás

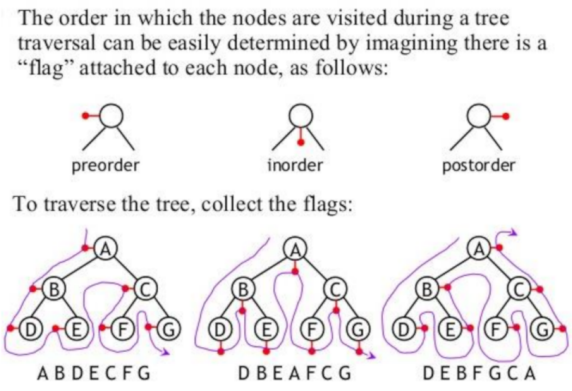
Járd be az előző (inorder bejárású) fát pre- és posztorder is!

Megoldás forrása:

Fabejárás.

Alapvetően már megtárgyaltuk, hogy mi is az a bináris fa, úgyhogy most itt az ideje egy kis csavarnak a történetben. Minek által a fánk ugyebár tudjuk, hogy gyökereket épít maga "alá" még mindig van egy olyan kérdés a levegőben, ami az ebben az adatszerkezetben nem annyira jártasaknak fel sem merül. Na de milyen irányba épül fel a "gyökérzete a fának? Milyen sorrendben és milyen rendszer alapján? Ez a feladat erre fog választ adni nekünk.

Futtatása a következő: gcc valemelyiktipusubejaras_binfa.c -lm -o valamelyiktipusubejaras_binfa. Majd: ./valamelyiktipusubejaras_binfa "balra mutató nyíl" kimenetifajl.txt



A kép a bináris fa bejárási módjait reprezentálja.
https://computersciencewiki.org/images/7/7c/Binary_tree_traversal.png

6.4. Tag a gyökér

Az LZW algoritmust ültesd át egy C++ osztályba, legyen egy Tree és egy beágyazott Node osztálya. A gyökér csomópont legyen kompozícióban a fával!

Megoldás forrása:

Tag a gyökér.

Röviden és tömören a lényege az, hogy most a C++ magasszintű programozási nyelvvel megkreált, de szerkezetében szinte teljesen azonos bináris fánkat némi módosítással ellátva alakítsunk ki. Jelen esetben a csomópontot be kell iktatnunk a gyökérbe, ezáltal könnyebben elérhetővé téve azt. Az is célunk ez által, hogy maga az LZW is megfelelően tudjon dolgozni vele. A futtatása a szokásos g++ megoldással működik: g++ fa.cpp -o fa, ezek után pedig kelleni fog a kiíratáskor valamilyen kimeneti szöveges fájl, és kész. A megoldás kódkiemelve lentebb szerepel.

```
            #include <iostream>

class LZWTree
{
public:
    LZWTree (): fa(&gyoker){}

    ~LZWTree ()
    {
        szabadit (gyoker.egyesGyermek ());
        szabadit (gyoker.nullasGyermek ());
    }

    void operator<<(char b)
    {
        if (b == '0')
        {

            if (!fa->nullasGyermek ())
            {
                Node *uj = new Node ('0');
                fa->ujNullasGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->nullasGyermek ();
            }
        }
        else
        {
            if (!fa->egyesGyermek ())
            {
                Node *uj = new Node ('1');
                fa->ujEgyesGyermek (uj);
                fa = &gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
```

```
            }
        }
    }
    void kiir (void)
    {
        melyseg = 0;
        kiir (&gyoker);
    }
    void szabadit (void)
    {
        szabadit (gyoker.jobbEgy);
        szabadit (gyoker.balNulla);
    }

private:

    class Node
    {
    public:
        Node (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
        ~Node () {};
        Node *nullasGyermek () {
            return balNulla;
        }
        Node *egyesGyermek ()
        {
            return jobbEgy;
        }
        void ujNullasGyermek (Node * gy)
        {
            balNulla = gy;
        }
        void ujEgyesGyermek (Node * gy)
        {
            jobbEgy = gy;
        }

    private:
        friend class LZWTree;
        char betu;
        Node *balNulla;
        Node *jobbEgy;
        Node (const Node &);
        Node & operator=(const Node &);
    };

    Node gyoker;
    Node *fa;
    int melyseg;
```

```
    LZWTree (const LZWTree &);
    LZWTree & operator=(const LZWTree &);

    void kiir (Node* elem)
    {
        if (elem != NULL)
        {
            ++melyseg;
            kiir (elem->jobbEgy);

            for (int i = 0; i < melyseg; ++i)
                std::cout << "---";
            std::cout << elem->betu << "(" << melyseg - 1 << ")" << std:: ↩
               endl;
            kiir (elem->balNulla);
            --melyseg;
        }
    }
    void szabadit (Node * elem)
    {
        if (elem != NULL)
        {
            szabadit (elem->jobbEgy);
            szabadit (elem->balNulla);
            delete elem;
        }
    }

};

int
main ()
{
    char b;
    LZWTree binFa;

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

    binFa.kiir ();
    binFa.szabadit ();

    return 0;
}
```

6.5. Mutató a gyökér

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

Megoldás forrása:

Mutató a gyökér.

Ez a feladat végülis az előző feladat megbonyolítása, ezzel is bővítve programozási ismereteinket, tudásunkat. A mostani helyzetünkben nem fog úgy működni a programunk mint ahogy azt mi elképzeljük, mivel a gyökér és a fa osztályok különálló szegmensekre lettek elosztva. Így alakult ki az a szituáció, hogy mindkét fél pointerként, azaz mutatóként "él" tovább a kódunkban. Na már most, hogy ebből megint egy használható eszközt tudjuk kifaragni szükségünk lesz pár dolog átalakítására. De mik is ezek a dolgok? Az első lépés, hogy magunkban helyre rakjuk azt a kis finomságot, hogy mostantól pointeren át tudjuk elérni a gyökeret. A lényegi része pedig az, hogy át kell pakolni az értéket úgy, hogy a gyökér eljusson a pointeren keresztül az épülendő csomópontokhoz.

```
            #include <iostream>

class LZWTree
{
public:
    LZWTree ()
    {
        gyoker  = new Node();
        fa = gyoker;
    }

    ~LZWTree ()
    {
         szabadit (gyoker->egyesGyermek ());
         szabadit (gyoker->nullasGyermek ());
         delete gyoker;
    }

    void operator<<(char b)
    {
        if (b == '0')
        {

            if (!fa->nullasGyermek ())
            {
                Node *uj = new Node ('0');
                fa->ujNullasGyermek (uj);
                fa = gyoker;
            }
            else
            {
                fa = fa->nullasGyermek ();
            }
        }
```

```
        else
        {
            if (!fa->egyesGyermek ())
            {
                Node *uj = new Node ('1');
                fa->ujEgyesGyermek (uj);
                fa = gyoker;
            }
            else
            {
                fa = fa->egyesGyermek ();
            }
        }
    }
    void kiir (void)
    {
        melyseg = 0;
        kiir (gyoker);
    }
    void szabadit (void)
    {
        szabadit (gyoker->jobbEgy);
        szabadit (gyoker->balNulla);
    }

private:

    class Node
    {
    public:
        Node (char b = '/'):betu (b), balNulla (0), jobbEgy (0) {};
        ~Node () {};
        Node *nullasGyermek () {
            return balNulla;
        }
        Node *egyesGyermek ()
        {
            return jobbEgy;
        }
        void ujNullasGyermek (Node * gy)
        {
            balNulla = gy;
        }
        void ujEgyesGyermek (Node * gy)
        {
            jobbEgy = gy;
        }

    private:
        friend class LZWTree;
```

```
        char betu;
        Node *balNulla;
        Node *jobbEgy;
        Node (const Node &);
        Node & operator=(const Node &);
    };

    Node *gyoker;
    Node *fa;
    int melyseg;

    LZWTree (const LZWTree &);
    LZWTree & operator=(const LZWTree &);

    void kiir (Node* elem)
    {
        if (elem != NULL)
        {
            ++melyseg;
            kiir (elem->jobbEgy);

            for (int i = 0; i < melyseg; ++i)
                std::cout << "---";
            std::cout << elem->betu << "(" << melyseg - 1 << ")" << std:: ↩
               endl;
            kiir (elem->balNulla);
            --melyseg;
        }
    }
    void szabadit (Node * elem)
    {
        if (elem != NULL)
        {
            szabadit (elem->jobbEgy);
            szabadit (elem->balNulla);
            delete elem;
        }
    }

};

int
main ()
{
    char b;
    LZWTree binFa;

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

```
    }

    binFa.kiir ();
    binFa.szabadit ();

    return 0;
}
```

6.6. Mozgató szemantika

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

Megoldás forrása:

Mozgató szemantika.

Ebben a feladatban továbbra is a bináris fával fogunk foglalkozni ugyan úgy, ahogy azt ebben az egész Welch nevezetű feladatcsokorban tettük. Ám de eljutva az utolsó feladathoz találkozhatunk csak a legnagyobb kihívással amit ez a csokor elénk állíthat. A mozgató szemantika, azaz konstruktor lényegi szerepe, hogy célzott szegmenseket és értékeket egy másik szegmensbe tudjuk "átpakolni". Még mielőtt tovább mennénk tisztáznunk kell, hogy ez az eset teljesen jól reprezentálható egy olyan hétköznapi példán keresztül is, mint két cipősdoboz és egy kis labda esete. Ugyanis a mozgató konstruktor működési elve szerint, ha az egyik dobozból átrakjuk a labdát a másikba, akkor az eredeti dobozunk tartalmilag üres lesz. Ez érvényes teljeskörűen a programunk esetére is, ahol ugyebár az eddigi feladatok alapjának szolgáló binfákkal is a pakolás során, az első kipakolt fa kiürül. Erőforrás alapján ráadásul kevesebbet igényel mint a másoló társa. A feladat leírását elemezve először különös lehet, hogy ott van a konstruktor és értékadás kifejezés is. Ez azért van mert technikai különbségek adódnak a két fajtája között. A legfőbb ilyen szempont az, hogy deklarálva van-e már az a tárgy, amit mozgatni szeretnénk. Amennyiben a válasz igen, akkor értékadásról van szó, mivel ezt majd csak a továbbiakban pakoljuk. Ezzel szemben pedig akkor van szó konstruktorról, ha a meghatározás során kap egy értéket.

A későbbi nyomonkövetési feltételnek is megfeleltetve van megadva a forrás.

```
            LZWBinFa binFa2, binFa4;
binFa2 << '1' << '1' << '1' << '1' << '1' << '1' << '1' << '1' << '1' << '1 ↩
   ';

binFa4 << '0' << '0' << '0' << '0' << '0' << '0';
std::cout << "egy"<< binFa2 << "egy" << std::endl;
std::cout << "ketto" << binFa4 << "ketto" <<std::endl;

binFa4 = binFa2;

std::cout << "egy"<< binFa2 << "egy" << std::endl;
std::cout << "ketto" << binFa4 << "ketto" <<std::endl;
```

```
        /*std::cout <<"Swappelünk" << std::endl;
        std::swap(binFa2,binFa4);
        std::cout <<"Vector-ba rakjuk" << std::endl;
        std::vector<LZWBinFa> v;
        v.push_back(std::move (binFa ));
        std::cout <<"Új fa" << std::endl;
        LZWBinFa binFa3 = std::move (binFa2);*/
```

7. fejezet

Helló, Conway!

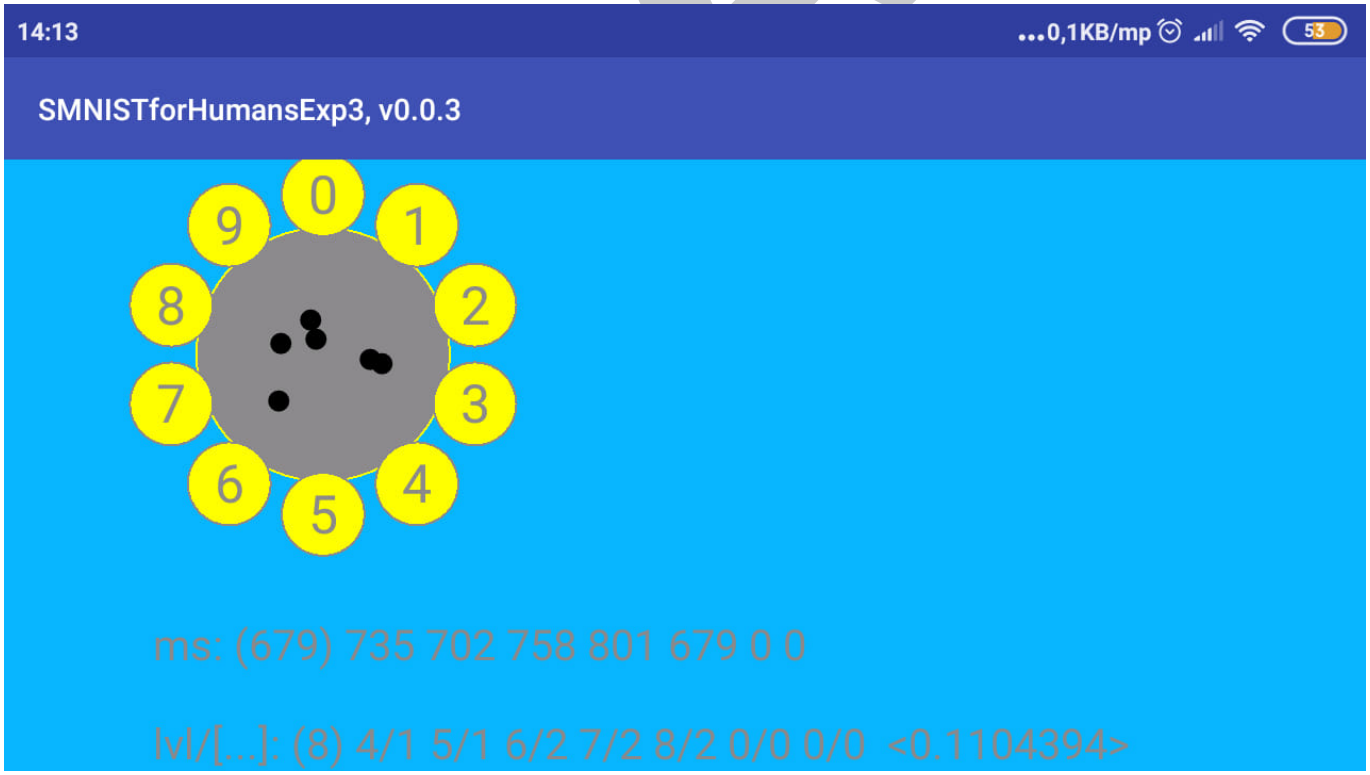
7.1. Hangyaszimulációk

Írj Qt C++-ban egy hangyaszimulációs programot, a forrásaidról utólag reverse engineering jelleggel készíts UML osztálydiagramot is!

Megoldás forrása:

Myrmecologist.

Elméleti rész erről a linkről származik, ami a bhaxor.blog.hu-ra vezet.



A feladatot passzoltam a kiváltható feladat elvégzésével, melyet ez a kép igazol.

7.2. Java életjáték

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

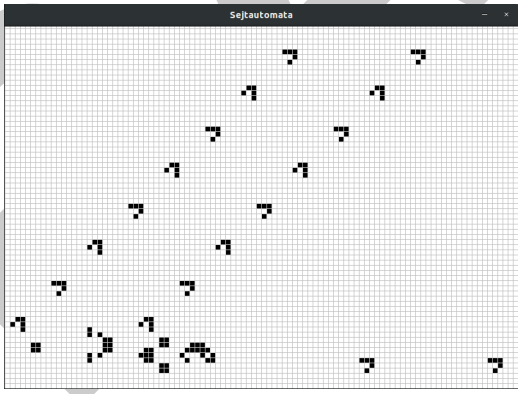
Megoldás forrása:

Java életjáték.

Az elméleti, azaz háttérhez kellő forrás: A Takönyvtár Java részlegén található rész.

A sejtautomaták világában talán a leírásban szereplő személy neve a legismertebb. A program alapjául az általa leírt Életjáték szolgál, ami megírva 1970-ben lett. Maga a játék, hogy is mondjam, elég érdekes, mert nem is igazán egy játékról beszélünk, mivel tőlünk nem vár be inputot, és a deklarált szabályok szerint a program önmagát futtatva játszik "önmagával". Ezek közül a szabályok közül egy példa, hogyha van egy élő sejt, aminek kevesebb mint két db másik élő sejt szomszédja van, akkor az ezt követő roundban, azaz menetben vagy körben meg fog halni az adott sejt. Ebből láthatóan állapotukat vizsgálva két féle sejtet különböztetünk meg, az élőt és a holt sejtet. Továbbá leszögezhető, hogy egymás között maximálisan csak nyolc darab sejt tud egyszerre interaktálni egymással. A feladat leírásában arra utalnak, hogy ezek a sejtek egymással fel tudnak venni bizonyos alakzatokat, amelyet a "siklók" lőnek ki magukból. Ehhez öt darab még életben levő sejtre van szükség.

Futtatása, amihez megjegyzem az alosztály kreálások miatt fontos, hogy a neve a programnak ne változzon. javac Sejtautomata.java, majd a java Sejtautomata parancssorokkal meg is kapjuk a kívánt eredményt. Ezen felül, a program már futása közben használhatunk négy gombot a billentyűzetünkről, amelyekkel valamennyire igazgathatjuk a játékot. Az N és a K páros a nagyításra és a kicsinyítésre szolgál az ablak módosításakor. Továbbá a G gomb a gyorsítása a körök lezajlásának, és az L, ami pedig a lassításért felel.



A kép a lefuttatás utáni eredményt kívánja prezentálni.

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása:

C++ életjáték.

Elméleti háttérként szolgált a: ProgPater.

Ez a feladat teljesen ugyan az, mint az előbb említett társa, annyi különbséggel, hogy itt most egy integrált környezetben tudjuk fejleszteni a kis életjátékunkat. Ehhez hozzá tartozik még az, hogy a Java-s verzióval

szemben, itt nem tudjuk megoldani egyetlen forrásfájllal az egész lefutást, hanem különböző header-öket létrehozva karöltve meghívjuk más cpp-s forrásokkal őket a programunk fő részébe. Részeit tekintve elsődleges a main, majd a sejtablak.h, ahol méretek vannak beállítva, majd a sejtablak.cpp, sejtszal.h és a sejtszal.cpp.

Kis csipet bemutatás végett:

```
            #include <QApplication>
#include "sejtablak.h"
#include <QDesktopWidget>

int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  SejtAblak w(100, 75);
  w.show();

  return a.exec();
}
```

7.4. BrainB Benchmark

Megoldás forrása:

BrainB Benchmark.

Ez a feladat ki lett hagyva, behúzással.

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python

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

MNIST

TensorFlow

Progpater-es link

Az MNIST egy olyan adatbázis, ahol az általunk választott, azaz megadott és kézzel beírt számokat tárólhatjuk. Ezeket az adatokat a legtöbb esetben képfeldogozó és képeket felismerő programok szokták használni, alkalmazni. A program a feladat leírása alapján, Python magasszintű programozási nyelven íródott, amelyhez a képfelismerés miatt még szükségünk van továbbá a TensorFlow nevezetű kiegészítő programra. A feldolgozás során 60000 adategységet tartalmazó adatbázist fogjuk felhasználni, amely 28*28-as felbontású, méretű képekkel dolgozik, amelyek fekete és fehér pixelekből állnak. Jelen feladatban, ahogy az a mellékelt videó alapján leírva lesz az a lényege, hogy vesszük az alap adatbázisunkat, ami 60000 képpel tanulja meg amit kiadunk neki. Ebben a példában a már említett felbontásban tálalt 0-tól 10-ig terjedő fekete pontú és fehér alapú képeket. A tanulás eredményét pedig egy 10000 darabból álló kontroll képpel tudjuk letesztelni.

A telepítés menete, hogy használni tudjuk a TensorFlow-ot a programunk futásához:

I. lépés -> sudo apt install python3-dev python3-pip

II. lépés -> sudo pip3 install -U virtualenv # system-wide install

III. lépés -> virtualenv --system-site-packages -p python3 ./venv

IV. lépés -> source ./venv/bin/activate # sh, bash, ksh, or zsh

V. lépés -> pip install --upgrade pip

VI. lépés -> pip install --upgrade tensorflow

Ezek után még feltétlenül szükségünk lesz a TensorFlow-wal kapcsolatos megfelelő head-örök implementálására.

A futtatáshoz szükségünk van az SMNIST programhoz, amellyel 60000 képet készítünk, amihez még kelleni fog 10000 db kép a kontrollhoz. Futtatni a következőképpen kell:

I. lépés -> A programunkat tartalmazó mappába belépve beírjuk, hogy: g++ smnistg.cpp -o smnistg -lpng

II. lépés -> ./smnistg train-labels-idx1-ubyte train-images-idx3-ubyte 60000

III. lépés -> ./smnistg t10k-labels-idx1-ubyte t10k-images-idx3-ubyte 10000

A futtatási részt magamtól nem igazán értettem, ezért az ihletért Nagy Lajos-hoz fordultam.

8.2. Mély MNIST

Python

Megoldás forrása:

TensorFlow weboldal a feladathoz, mint forrás amelyből a megoldást fordítottam.

TensorFlow MNIST

Lényegbe foglalva a feladatban szereplő témát, segítségül hívom az előbbi feladat során szerzett ismereteinket. Rövid bevezőként némi összehasonlítás során megállapíthatjuk, hogy a jelenlegi kiértékelendő koncepciónkban egy többrétegő soft mnist-et kapunk, ami ebből adódóan pontosabban végzi el a kapott műveleteket amelyeket beleimplementáltunk és meghatároztunk. Továbbá némi "hátrányt" is megfigyelhetünk, amit a megfigyeléseink során a nagyobb erőforrásigényben találhatunk meg. Ismeeretünket kiterjesztve közlendő, hogy a TensorFlow egy nagyon erős könyvtár, ami a széles skálájú számokkal dolgozik. Ezáltal is elősegítve az implementáló és tanuló mély neutrális működést.

Kezdésként fontos tisztázni, hogy az első két sor feltétlenül szükséges, mivel boilerplate-ként szerepel jelen esetben. Ugyanis ezáltal töltünk és olvasunk adatot atomataként. A következő pár sor fogja tartalmazni a kiszámítási gráf építését fogja lekreálni, és a futtató gráfot is. Tovább építve szükséges a Softmax Regressziós Modelt, ahol egy gyökérpontokat készítünk a bemeneti képhez és a célkimeneti osztályokhoz. Ezeket az x és y_ változókkal jelöljük. Az első változó értéke a 28*28-as pixeles MNIST-es képet jelöli. A tanulás maga a model kereteiben belül zajlik majd le. A TensorFlow egyik előnye ezen a téren, hogy nem csak sima bemenetként tuddjuk kezelni a mélységet és a tendenciát, hanem változókként.

8.3. Minecraft-MALMÖ

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

Megoldás forrása:

Minecraft-MALMÖ bhax-blog forrás

A programot a Python programnyelven írtuk meg, amely a Mojang által készített Minecraft nevezetű játékához kapcsolódik. A feladat fő szegmense, hogy a játékban megtalálható karakterünknek megadunk egy általunk választott lépésszámot, ami egy bármekkora pozitív egész lehet, de a figyelem arra irányul, hogy a bábunak, azaz a karakternek nem szabad megállnia, pontosabban elakadnia sehol sem. A program működése során ugyebár végig, a haladás során beolvassuk a karakterünk körüli blokkokat (a játék kockából álló világból áll) és amíg lehet, addig egyenesen halad. Az elakadás esetén a beolvasott környezeti blokkokat elemezve azt az utat választjuk ki, amelyik hamarabb esik a keresési folyamatba. Ezt úgy vizsgáljuk

meg, hogy a folyamat során a karakterünk irányától jobbra kezdjük az átvizsgálást, és az első lehetőséget kihasználva megyünk tovább előre. De mi van akkor ha ez ne működik? Akkor azt vizsgáljuk meg, hogy van e olyan blokk (ami eggyel magasabb mint a környezetünk) amire fel tudunk ugrani, és ha ilye van, akkor élünk az ebből adódó lehetőséggel. Ezzel az útkeresési és választó aktív folyamatnak hála, a karakterünk hibátlanul, elakadás mentesen tudja megtenni az általunk, a feladat elején megadott megtenni vágyott lépéseknek a számát.

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása:

Iterációs kép

Rekurzív kép

Feladatok

Tutorom: Pankotai Kristóf

Először is szeretném bevezetni a feladatban szereplő programozási nyelv általános tudnivalóit, hogy még érthetőbb legyen a feladat későbbi magyarázata és értelmezése az olvasó számára. Az alkalmazása elég széles körben elterjedt, ez alapján pedig egy ismert nyelvről lesz szó. A feladat során a Scheme dialektust kell használnunk. A program, és a feladat megoldásához szükségünk van a Gimp képmoduláló programjában található beépített környezetre, a Script-fu-ra, ami a Szűrők elnevezésű panelben lelhető meg.

A forráshoz hozzárendelt iteratív faktoriálisként megírt programkód bemutatásával kezdem a bekezdést. Itt először is definiáljuk az n-t ami lehetőleg valami olyan nevet kapjon, ami a faktoriális kifejezésre hajaz, ezáltal megkönnyítve a későbbi olvasását a kódchipetnek. Emellé még meg kell adnunk egy product nevezetű változót is, amivel az értékvizsgálat fog lezajlani olyan módon, ami majd a rekurzív faktoriálisos feladatban is kelleni fog. Egy értékvizsgálati eljárás során megnézzük, hogy a kapott eredmény megegyezik-e eggyel, és ha igen, akkor önmagát adjuk vissza válaszként. Egyéb esetben, ahol az értéke nagyobb lesz az előbb említett számnál, azaz az egynél, akkor kiszámolja a faktoriálist, és azt adja meg válaszul.

A feladatban említett rekurzív faktoriális megalkotása a következő módon jelenik meg. Először is az n változót, ami egy szám lesz, definiálnunk kell, lehetőleg faktoriálisnak elnevezve az egyszerűség szempontja és átláthatóság végett. Ezt követően egy értékvizsgálattal megállapítjuk az értékét. Amennyiben a kapott érték nagyobb mint egy, ekkor a faktoriális képletbeli folytatása lesz a kövezkező lépés, ami megadja a kapott n értékének a faktoriálisát. Amennyiben az értéke megegyezik eggyel, ugyanazt fogja visszaadni, azaz önmagát.

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:

Króm effekt

A címben szereplő feladat során ismételten igénybe vesszük, az előző feladat alapján már jól ismert Gimp nevezetű képszerkesztő progamot amelyhez most egy szkript megírásával egy új bővítményt, azaz kiegészítőt fogunk hozzárendelni a következőféleképpen. Ahogy a leírás is megfogalmazza, egy krómium effektet fogunk létrehozni a programhoz. A forrásban megosztott kód hordozza és foglalja magába azokat a beállításokat, amelyekkel elérjük a krómos hatást. Alapjában véve ehhez szükségünk lesz az előző feladatban is használt rekurzív függvényt, de erről majd később fejtek ki részletesebben tudnivalókat. A kis kódrészletünkben szerepelnek olyan tömbök, amelyek a színskálákat ölelik fel, plusz egy rekurzív függvény alapú szegmenst, amely a listát felhasználva vesz ki részeket/elemeket, majd egy ehhez hasonlót ami a szélességét elemzi és intermediál a szöveggel, majd a magasságával is. Ezeket használva érjük el a szkript végleges alakját, és ezzel a célunkat is, hogy elérjük a króm alapú és kinézetű hatást. Legvégül pedig ne felejtsük el, hogy a megírt szkript-et be kell építenünk a képszerkesztő programunkba úgy, hogy a fájlunkat bemásoljuk a Gimp megfelelő, direkt az ilyen célre konstruált mappába, amit majd a későbbiekben a kiegészítők közt találunk meg.

A kép a feladatban meghatározott effekt eredményét illusztrálja.

9.3. Gimp Scheme Script-fu: név mandala

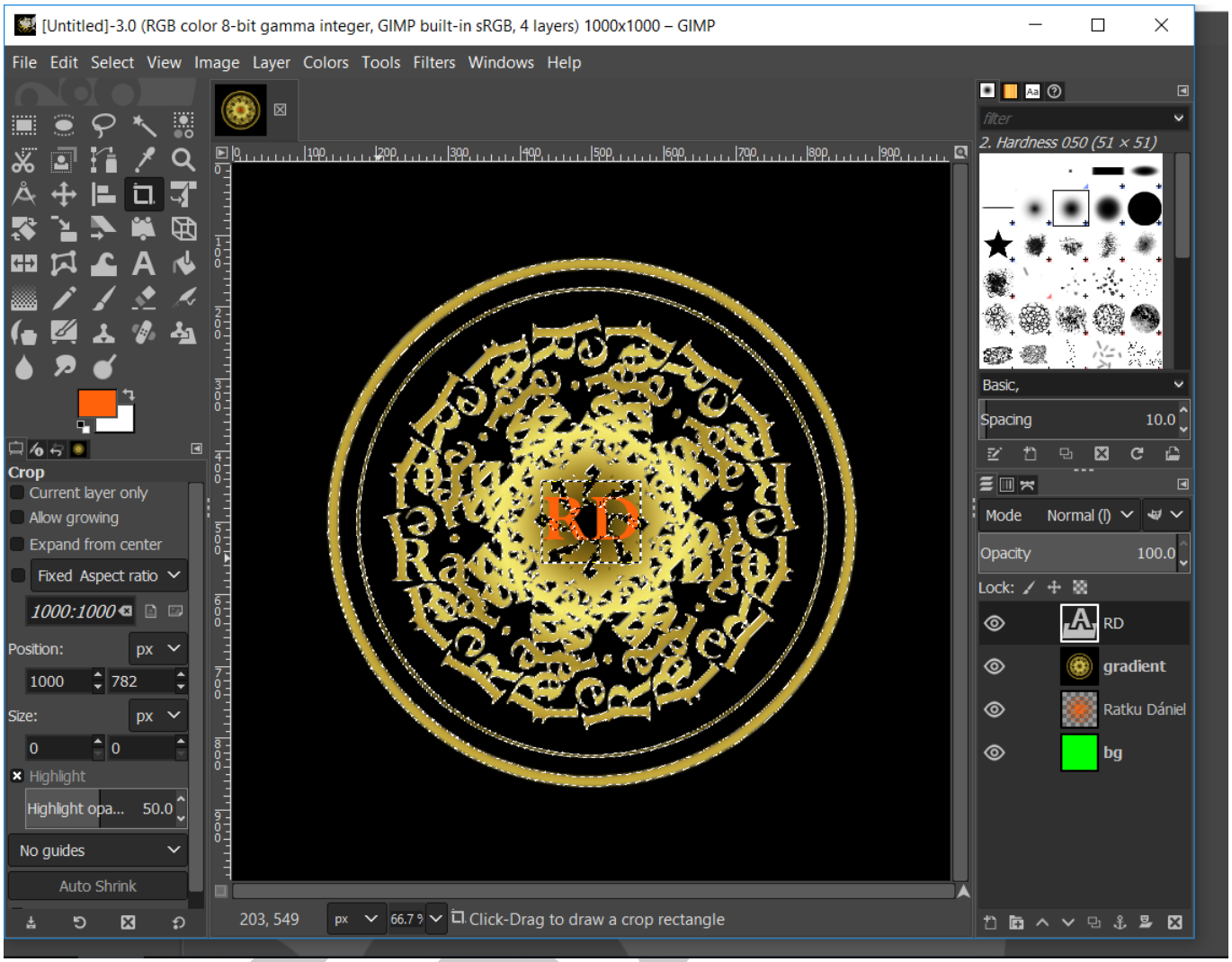
Írj olyan script-fu kiterjesztést a GIMP programhoz, amely név-mandalát készít a bemenő szövegből!

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

Megoldás forrása:

Script

A program amelyet megvalósítunk eléggé hasonlít az előző feladatban felvetett koncepcióhoz. Itt is a cél egy szkript megírása annyi különbséggel, hogy most nem egy konkrét színefektet hozunk létre, hanem a címben is szereplő mandala-t, amelynek az eredete indiai képmegalkotási mód. Első sorban, mint általában most is a változók létrehozásával, és definiálásával kell kezdenünk. Ezek után a szoftverünk átmegy a képalkotási lépcsőkön, hogy elérjük amit szeretnénk, azaz a mandala megalkotását. Első valós lépésünk a kívánt hatásért fontos meghatároznunk, hogy milyen szót, és betűtípust választunk ki, hogy erre vegyük igénybe az általunk megírt, vagy kölcsönvett és használt szkriptet. A kódcsipet folyamatosan pörög ezalatt a procedúra alatt a háttérben, hogy használja a meghatározott folyamatot a célért.



A kép a feladat megoldását illusztrálja.

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

[?]

Az elején egy bevezetést kapunk a programozási nyelvek általánosságairól, főleg az alaptani dolgokról, és a típusokról. Háromféle programozási nyelvet különböztetünk meg egymástól. Ez az assembly nyelv, ami úgy ismerhetünk, mint a gépi nyelvhez legközelebb eső és álló programozási nyelv. Másodikként beszélhetünk a gépi nyelvről, és ami a mi esetünkben most a legfontosabb, a magasszintű programozási nyelvek. Az ebben megírt típusú kódok elnevezése a forrás lesz, amíg a nyelvtani szabálykeretrendszert szintaktikai szabályként definiálunk. Először a megírt forrást fordítani kell, hogy az gépi nyelvű legyen, ezáltal megértetve a számítógéppel amit szeretnénk, és majd csak ezután jön a tényleges futtatás. Bevezetésre kerül még az interpreteres módszer is, amely az előzővel ellenben nem készíti el a tárgyprogramot. Rengeteg futtatási környezet van már a világon.

Tovvábiakban szó lesz az imperatív nyelvekről és a jelölő rendszerekről. Kezdjünk is bele egyből az előbb említett résszel. Önmagában ez egy olyan algoritmus, ami a processzor működéséért felel, ezáltal kezeli az utasításokat, és eljárásokat is. Emellé megemlítendő a deklaratív nyelv is, amely nem feltétlen hasonló mint az előbbi társa, ugyanis ez eltér a Neumann elvektől. Most pedig elindulhatunk kielemezni az első mondatban említett második főelemet. Szóval, első sorban tisztázzuk, hogy ebből többféle is létezik. Például terminális és akármilyen meglepő is lehet, de a nem terminális is. Ez kiemelendően fontos a szintaktikai szabályok megalkotásához.

Előző fejezetek után jött egy elég komoly szakasz, ahol a különböző adattípusokról ejtett szót az író, és hogy ez milyen összetevője is pontosan az általunk ismert programozásnak. Erre példa lehet mondjuk valaminek az értéke. Léteznek absztrakt típusok és alaptípusok. Az elsőt vizsgálva három összetevője van. Első a tartomány, a kapcsolódó műveletek, és a reprezentáció. Az altípussal pedig inkább lekérdezésekhez használjuk, ha valamilyen típus érdekel minket.

A soron következő témakör az I/O volt ami az oprendszerrel áll kapcsolatban. Működése során a memóriával foglalatoskodik, azan onnak fogad és ad információt, adatot. Rengetek típus van. Karakter, ami lehet string. Egész ami fixpontos, és valós, ami lebegő pontos. Van még logikai, ami lehet igaz és hamis értékű. Ezt az informatikában egyessel és nullással jelöljük. Végül pedig a mutató, ami érték kérő. Név, típus és érték az a három alkotó elem, amely a konstansot nevesítetté teszi.

10.2. Programozás bevezetés

[KERNIGHANRITCHIE]

A könyv azzal kezdődik, hogy összefoglalja a C programozási nyelv eredet mítoszát, és annak közeli kapcsolatát a UNIX-xal, ami egy operációs rendszer. Olyan igazságokat foglal magában, mint a C nyelv megtanulásának legjobb módja, ami míly meglepő a gyakorlásban rejlik. Továbbá kifejti mi is a nyelvnek az alap sója és borsa. Ilyen eset például a vezérleti elv, ami a szintaktikát foglalja magába, és kiemeli a ";" végi fontosságot, és a változók bevezetését.

Fő felhasználatáról beszél, hogy milyen jól lehet vele operációs rendszerek kidolgozásakor. Folytatólagosan kitér az If függvény működésére, mégpedig hogy ha az If igaz , akkor az jó, ha meg nem, akkor az else ágon lesz a továbbfutása a programnak. Ezután sorra veszi a While ciklust és annak elvét. Kibővíti a Do-While-al, ami addig csinálja az általunk kért folyamatot amíg el nem éri a megint csak általunk meghatározott feltételt. Na meg hát ki ne maradjon a For ciklus se ebből a halomból, így hát róluk is esett szó, amelynek működési elvben nagyban hasonlít a Do-ra.

Kiemelődik mi nem elérhető a nyelvben. Például hogy nincs benne objektum, ami akár jó is lehet a szabad forráskód írás tekintetében. Visszatéregetünk az előző fejezetben leírt ciklusok megszakítására és újra folytatására amit a break-kel érhetünk el, az utóbb említettet pedig a continue-val. Végül, az általam kiolvasott utolsó kiadott fejezetben pedig arról kellett kiértékelést adnom az olvasottakról, hogy minek által tudunk kombinált utasítást írni. Ezt a könyv, mint az eddigieket is több bekezdésen keresztül részletesen taglalja a lehető legegyszerűbb és a lehető legtisztább, gyorsabb megértés érdekében. Célja ezzel, hogy egy parancs kiadásával tudjuk többet is futtatni.

10.3. Programozás

[BMECPP]

Mint az előzőben, itt is egy programozási nyelv bevezetésről lesz szó, minek által valahogy csak el kell kezdeni egy programozás specifikus könyv megírását. Ebben a könyvben már a C++ magasszintű programozási nyelvről lesz szó, amelyben a magasszintű programozási 1 nevezetű egyetemi második féléves tárgyunk nagyja épült, főleg a feladatcsokrok második felétől indulóan, ahol már szerves részévé vált a kidolgozásunkban. Elkezdi az eredettörténetét, ahol kapcsolatba állítja a C programozási nyelvvel és összeveti az utóbb említett nyelv hasonlóságaival, és az ellentéteivel.

A könyv bőszen megemlíti a program formális szabványait, és ezt úgy, hogy több különböző esetre is levetíti. Erre példát statuálhat amikor az alap szabványt említi és amikor argumentumokat is szeretnénk alkalmazni a programunkban a kódolás során. Még megemlítésre kerül egy olyan különbség az elődjével szemben (ami a C), hogy itt új típussal is találkozunk, még pedig a bool-al, ami a logikához kapcsolódik, mert ezáltal kreálhatunk mostantól olyan változót, aminek logikai értéke van. Plusz a wchar, ami sok bájt tárolására hasznos, és emiatt is lett létrehozva.

Hatalmas újítások közt szerepel még, hogy mostantól az elődjével szemben képes az operátor és függvény túlterhelésre is. Azaz képesek vagyunk arra is, hogy egyazon elnevezésű függvényeket tudjunk használni abban az esetben, ha azok argomentumaikban különböznek egymástól.

Kifejti a referencia típus előnyeit, melyet fokoz a sablonok használatának hasznosságáról. Ezek után jön, hogy mivel is dolgozik maga a nyelv, mégpedig stream és bájt sorozatok tömkelegével. Ámbár ekkor már

tudjuk, hogy milyen boilerplate szerepe van az iostream implementálásának a programunkba, ugyanis ezáltal tudunk manipulátorokkal is dolgozni, ami azért elég hasznos. Ezt veti össze hasznosság és produktivitás szempontjából az endl és a setprecision között. Továbbá ehhez szükség van az állománykezeléshez is, amely külső fájlokkal való dolgozást jelenti. Elődjével szembeni másik kiválósága hogy tudunk vele írni és ovlasni is külső forrásból. Az utolsó érdemleges szegmens pedig nem más, mint ami az egyetemi tanulmányaink során a bevezetés a programozásba nevezetű tárgyunkban már szintén szóba került hiba kereséses elkapás, azaz a try-catch nevezetű hiba kiírás. Ennél deklarálnunk kell hogy milyen hibát is keressen és adjon ki.

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.