Univerzális programozás

Így neveld a programozód!

Ed. BHAX, DEBRECEN,
2020. március 22, v. 0.0.7

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

COLLABORATORS

	TITLE : Univerzális programozás	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Bátfai, Norbert, Bátfai, Mátyás, Bátfai, Nándor, Ã©s Bátfai, Margaréta	2020. október 7.	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.1	2019-02-12	Az iniciális dokumentum szerkezetének kialakítása.	nbatfai
0.0.2	2019-02-14	Inciális feladatlisták összeállítása.	nbatfai
0.0.3	2019-02-16	Feladatlisták folytatása. Feltöltés a BHAX csatorna https://gitlab.com/nbatfai/bhax repójába.	nbatfai
0.0.4	2019-02-19	A Brun tételes feladat kidolgozása.	nbatfai
0.0.5	2020-03-02	Az Chomsky/$a^nb^nc^n$ és Caesar/EXOR csokor feladatok kiírásának aktualizálása (a heti előadás és laborgyakorlatok támogatására).	nbatfai

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
0.0.6	2020-03-21	A MALMÖ projektes feladatok átvezetése, minden csokor utolsó feladata Minecraft ágensprogramozás ezzel. Mottók aktualizálása. Prog1 feladatok aktuzalizálása. Javasolt (soft skill) filmek, elméletek, könyvek, előadások be.	nbatfai
0.0.7	2020-03-22	Javítások.	nbatfai

Ajánlás

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

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

„Csak kicsi hatást ért el a videójáték-ellenes kampány. A legtöbb iskolában kétműszakos üzemben dolgoznak a számítógépek, értő és áldozatos tanárok ellenőrzése mellett."

„Minden számítógép-pedagógus tudja a világon, hogy játékokkal kell kezdeni. A játékot követi a játékprogramok írása, majd a tananyag egyes részeinek a feldolgozása.„

—Marx György, *Magyar Tudomány, 1987 (27) 12.*, [MARX]

„I can't give complete instructions on how to learn to program here — it's a complex skill. But I can tell you that books and courses won't do it — many, maybe most of the best hackers are self-taught. You can learn language features — bits of knowledge — from books, but the mind-set that makes that knowledge into living skill can be learned only by practice and apprenticeship. What will do it is (a) reading code and (b) writing code."

—Eric S. Raymond, *How To Become A Hacker, 2001.*, http://www.catb.org/~esr/faqs/hacker-howto.html

„I'm going to work on artificial general intelligence (AGI)."

I think it is possible, enormously valuable, and that I have a non-negligible chance of making a difference there, so by a Pascal's Mugging sort of logic, I should be working on it.

For the time being at least, I am going to be going about it "Victorian Gentleman Scientist" style, pursuing my inquiries from home, and drafting my son into the work."

—John Carmack, *Facebook post, 2019.*, in his private Facebook post

Tartalomjegyzék

Ábrák jegyzéke

Előszó

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

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

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

Hogyan forgasd

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

Miért univerzális? Mert az olvasótól (kvázi az írótól) függ, hogy kinek szól a könyv. Alapértelmezésben gyerekeknek, mert velük készítem az iniciális változatot. Ám felhasználjuk az egyetemi programozás oktatásban is: a reguláris programozás képzésben minden hallgató otthon elvégzendő labormérési jegyzőkönyvként, vagy kollokviumi jegymegajánló dolgozatként írja meg a saját változatát belőle. 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?

Magam is ezeken gondolkozok. Szerintem a programozás lesz a jegyünk egy másik világba..., hogy a galaxisunk közepén lévő fekete lyuk eseményhorizontjának felületével ez milyen relációban van, ha egyáltalán, hát az homályos...

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 [BMEPY] könyv 25-49, kb. 20 oldalas gyorstalpaló részét.

1.3. Milyen filmeket, előadásokat nézzek meg, könyveket olvassak el?

A kurzus kultúrájának élvezéséhez érdekes lehet a következő elméletek megismerése, könyvek elolvasása, filmek megnézése.

Elméletek.

- Einstein: A speciális relativítás elmélete.
- Schrödinger: Mi az élet?
- Penrose-Hameroff: Orchestrated objective reduction.
- Julian Jaynes: Breakdown of the Bicameral Mind.

Könyvek.

- Carl Sagan, Kapcsolat.
- Roger Penrose, A császár új elméje.
- Asimov: Én, a robot.
- Arthur C. Clarke: A gyermekkor vége.

Előadások.

- Mariano Sigman: Your words may predict your future mental health, https://youtu.be/uTL9tm7S1Io, hihetetlen, de Julian Jaynes kétkamarás tudat elméletének legjobb bizonyítéka információtechnológiai...
- Daphne Bavelier: Your brain on video games, https://youtu.be/FktsFcooIG8, az esporttal kapcsolatos sztereotípiák eloszlatására („The video game players of tomorrow are older adults”: 0.40-1:20, „It is not true that Screen time make your eyesight worse”: 5:02).

Filmek.

- 21 - Las Vegas ostroma, https://www.imdb.com/title/tt0478087/, benne a Monty Hall probléma bemutatása.
- Rain Man, https://www.imdb.com/title/tt0095953/, az [SMNIST] munkát ihlette, melyeket akár az MNIST-ek helyett lehet csinálni.
- Kódjátszma, https://www.imdb.com/title/tt2084970, benne a kódtörő feladat élménye.
- Interstellar, https://www.imdb.com/title/tt0816692.
- Middle Men, https://www.imdb.com/title/tt1251757/, mitől fejlődött az internetes fizetés?
- Pixels, https://www.imdb.com/title/tt2120120/, mitől fejlődött a PC?

- Gattaca, https://www.imdb.com/title/tt0119177/.
- Snowden, https://www.imdb.com/title/tt3774114/.
- The Social Network, https://www.imdb.com/title/tt1285016/.
- The Last Starfighter, https://www.imdb.com/title/tt0087597/.
- What the #$*! Do We (K)now!?,, https://www.imdb.com/title/tt0399877/.
- I, Robot, https://www.imdb.com/title/tt0343818.

Sorozatok.

- Childhood's End, https://www.imdb.com/title/tt4171822/.
- Westworld, https://www.imdb.com/title/tt0475784/, Ford az első évad 3. részében konkrétan meg is nevezi Julian Jaynes kétkamarás tudat elméletét, mint a hosztok programozásának alapját...
- Chernobyl, https://www.imdb.com/title/tt7366338/.
- Stargate Universe, https://www.imdb.com/title/tt1286039, a Desteny célja a mikrohullámú háttér struktúrája mögötti rejtély feltárása...
- The 100, https://www.imdb.com/title/tt2661044/.
- Genius, https://www.imdb.com/title/tt5673782.

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 videó: https://youtu.be/lvmi6tyz-nI

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-f.c, bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/infty-w.c.

Számos módon hozhatunk és hozunk létre végtelen ciklusokat. Vannak esetek, amikor ez a célunk, például egy szerverfolyamat fusson folyamatosan és van amikor egy bug, mert ott lesz végtelen ciklus, ahol nem akartunk. Saját péláinkban ilyen amikor a PageRank algoritmus rázza az 1 liter vizet az internetben, de az iteráció csak nem akar konvergálni...

Egy mag 100 százalékban:

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

  return 0;
}
```

vagy az olvashatóbb, de a programozók és fordítók (szabványok) között kevésbé hordozható

```
int
#include <stdbool.h>
main ()
{
  while(true);

  return 0;
}
```

Azért érdemes a `for(;;)` hagyományos formát használni, mert ez minden C szabvánnyal lefordul, másrészt a többi programozó azonnal látja, hogy az a végtelen ciklus szándékunk szerint végtelen és nem szoftverhiba. Mert ugye, ha a `while`-al trükközünk egy nem triviális `1` vagy `true` feltétellel, akkor ott egy másik, a forrást olvasó programozó nem látja azonnal a szándékunkat.

Egyébként a fordító a `for`-os és `while`-os ciklusból ugyanazt az assembly kódot fordítja:

```
$ gcc -S -o infty-f.S infty-f.c
$ gcc -S -o infty-w.S infty-w.c
$ diff infty-w.S infty-f.S
1c1
<    .file "infty-w.c"
---
>    .file "infty-f.c"
```

Egy mag 0 százalékban:

```
#include <unistd.h>
int
main ()
{
  for (;;)
    sleep(1);

  return 0;
}
```

Minden mag 100 százalékban:

```
#include <omp.h>
int
main ()
{
#pragma omp parallel
{
  for (;;);
}
  return 0;
}
```

A **gcc infty-f.c -o infty-f -fopenmp** parancssorral készítve a futtathatót, majd futtatva, közben egy másik terminálban a **top** parancsot kiadva tanulmányozzuk, mennyi CPU-t használunk:

```
top - 20:09:06 up  3:35,  1 user,  load average: 5,68, 2,91, 1,38
Tasks: 329 total,   2 running, 256 sleeping,   0 stopped,   1 zombie
%Cpu0 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
```

```
%Cpu3 : 99,7 us, 0,3 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu4 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu5 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu6 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu7 :100,0 us, 0,0 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem :16373532 total,11701240 free, 2254256 used, 2418036 buff/cache
KiB Swap:16724988 total,16724988 free,       0 used. 13751608 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 5850 batfai    20   0   68360    932    836 R 798,3  0,0   8:14.23 infty-f
```

Werkfilm

- https://youtu.be/lvmi6tyz-nI

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

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

Megoldás videó:

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

```
Program T100
{

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

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

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

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

akár önmagára

```
T100(T100)
false
```

ezt a kimenetet adja.

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

```
Program T1000
{

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

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

  main(Input Q)
  {
    Lefagy2(Q)
  }

}
```

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

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.3. Változók értékének felcserélése

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

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

Megoldás forrása: ugyanott.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.4. Labdapattogás

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

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

Megoldás forrása: ugyanott.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.5. Szóhossz és a Linus Torvalds féle BogoMIPS

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU, https://youtu.be/KRZlt1ZJ3qk, .

Megoldás forrása: `bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Turing/bogomips.c`

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.6. Helló, Google!

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

Megoldás forrása: a második előadás 55-63 fólia.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.7. A Monty Hall probléma

Írj R szimulációt a Monty Hall problémára!

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

2.8. 100 éves a Brun tétel

Írj R szimulációt a Brun tétel demonstrálására!

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

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

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

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

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

Majd vizsgáljuk meg az a sorozatot:

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

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

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

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

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

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

```
#    Copyright (C) 2019  Dr. Norbert Bátfai, nbatfai@gmail.com
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU General Public License as published by
#    the Free Software Foundation, either version 3 of the License, or
#    (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU General Public License for more details.
#
#    You should have received a copy of the GNU General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>

library(matlab)

stp <- function(x){

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

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

Soronként értelemezzük ezt a programot:

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

Kiszámolja a megadott számig a prímeket.

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

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

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

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

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

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

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

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

Kivette a primes-ból a párok első tagját.

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

A párok második tagját az első tagok kettő hozzáadásával képezzük.

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

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

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

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

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

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

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

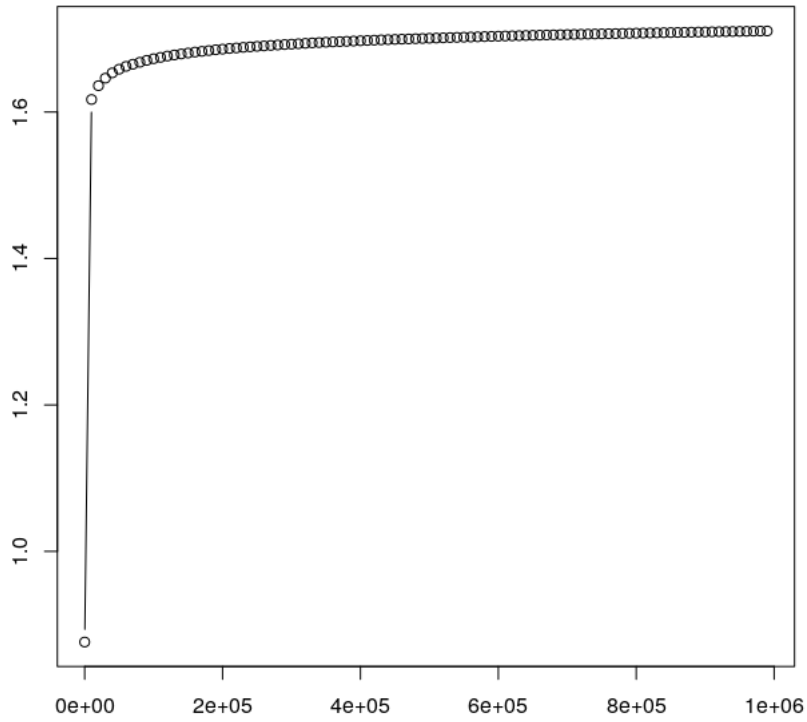
Nincs más dolgunk, mint ezeket a törteket összeadni a `sum` függvénnyel.

```
sum(rt1plust2)
```

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

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

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



2.1. ábra. A B_2 konstans közelítése

Werkfilm

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

2.9. Vörös Pipacs Pokol/csiga folytonos mozgási parancsokkal

Megoldás videó: https://youtu.be/uA6RHzXH840

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3. fejezet

Helló, Chomsky!

3.1. Decimálisból unárisba átváltó Turing gép

Állapotátmenet gráfjával megadva írd meg ezt a gépet!

Megoldás forrása: az első előadás 27 fólia.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.2. Az $a^nb^nc^n$ nyelv nem környezetfüggetlen

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

Megoldás forrása: az első előadás 30-32 fólia.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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: az első előadás 63-65 fólia.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.4. Saját lexikális elemző

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

Megoldás videó: https://youtu.be/9KnMqrkj_kU (15:01-től).

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/realnumber.l

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.5. Leetspeak

Lexelj össze egy l33t ciphert!

Megoldás videó: https://youtu.be/06C_PqDpD_k

Megoldás forrása: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/l337d1c7.l

```
%{
  #include <stdio.h>
  #include <stdlib.h>
  #include <time.h>
  #include <ctype.h>

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

  struct cipher {
    char c;
```

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

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

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

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

%}
%%
. {

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

```
    {

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

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

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

        found = 1;
        break;
      }

    }

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.6. A források olvasása

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

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

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

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

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

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

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

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

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

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

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

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

Megoldás videó: BHAX 357. adás.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.7. Logikus

Hogyan olvasod természetes nyelven az alábbi Ar nyelvű formulákat?

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

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

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

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

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

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

Tanulságok, tapasztalatok, magyarázat...

3.8. Deklaráció

Vezesd be egy programba (forduljon le) a következőket:

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

Mit vezetnek be a programba a következő nevek?

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

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

Megoldás videó: BHAX 357. adás.

Az utolsó két deklarációs példa demonstrálására két olyan kódot írtunk, amelyek összahasonlítása azt mutatja meg, hogy miért érdemes a **typedef** használata: bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr.c, bhax/thematic_tutorials/bhax_textbook_IgyNeveldaProgramozod/Chomsky/fptr2.c.

```
#include <stdio.h>

int
sum (int a, int b)
{
    return a + b;
}

int
mul (int a, int b)
{
    return a * b;
}

int (*sumormul (int c)) (int a, int b)
{
    if (c)
        return mul;
    else
        return sum;

}

int
main ()
{

    int (*f) (int, int);

    f = sum;

    printf ("%d\n", f (2, 3));

    int (*(*g) (int)) (int, int);

    g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));
```
```

```
    return 0;
}
```

```
#include <stdio.h>

typedef int (*F) (int, int);
typedef int (*(*G) (int)) (int, int);

int
sum (int a, int b)
{
    return a + b;
}

int
mul (int a, int b)
{
    return a * b;
}

F sumormul (int c)
{
    if (c)
        return mul;
    else
        return sum;
}

int
main ()
{

    F f = sum;

    printf ("%d\n", f (2, 3));

    G g = sumormul;

    f = *g (42);

    printf ("%d\n", f (2, 3));

    return 0;
}
```

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

3.9. Vörös Pipacs Pokol/csiga diszkrét mozgási parancsokkal

Megoldás videó: https://youtu.be/Fc33ByQ6mh8

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4. fejezet

Helló, Caesar!

4.1. double ** háromszögmátrix

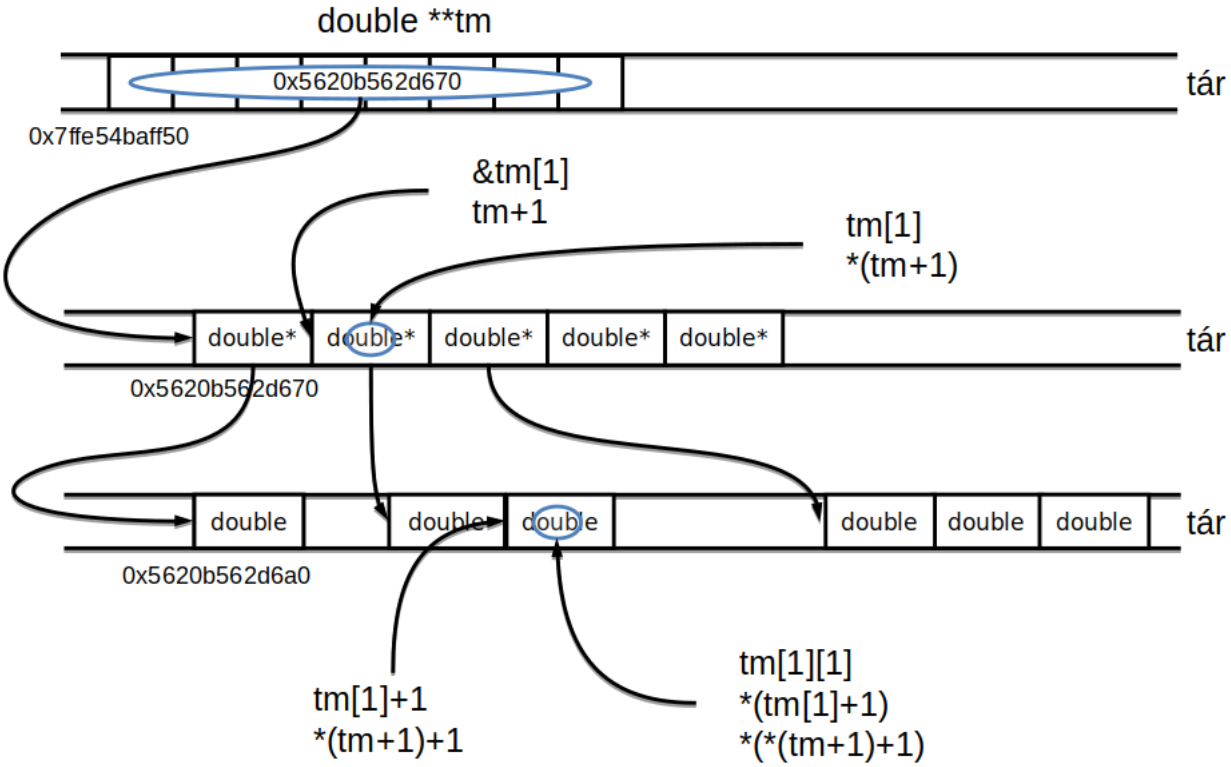
Írj egy olyan `malloc` és `free` párost használó C programot, amely helyet foglal egy alsó háromszög mátrixnak a szabad tárban!

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

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

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

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

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

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

    }

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

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

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

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

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

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

    free (tm);

    return 0;
}
```



4.1. ábra. A double ** háromszögmátrix a memóriában

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.2. C EXOR titkosító

Írj egy EXOR titkosítót C-ben!

Megoldás forrása: egy részletes feldolgozása az posztban, lásd az e.c forrást.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.3. Java EXOR titkosító

Írj egy EXOR titkosítót Java-ban!

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.4. C EXOR törő

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

Megoldás forrása: egy részletes feldolgozása az posztban, lásd az t.c forrást.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.5. Neurális OR, AND és EXOR kapu

R

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.6. Hiba-visszaterjesztéses perceptron

Fontos, hogy ebben a feladatban még nem a neurális paradigma megismerése a cél, hanem a többrétegű perceptron memóriakezelése (lásd majd a változó argumentumszámú konstruktorban a double *** szerkezetet).

Megoldás videó: https://youtu.be/XpBnR31BRJY

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

4.7. Vörös Pipacs Pokol/írd ki, mit lát Steve

Megoldás videó: https://youtu.be/-GX8dzGqTdM

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

5. fejezet

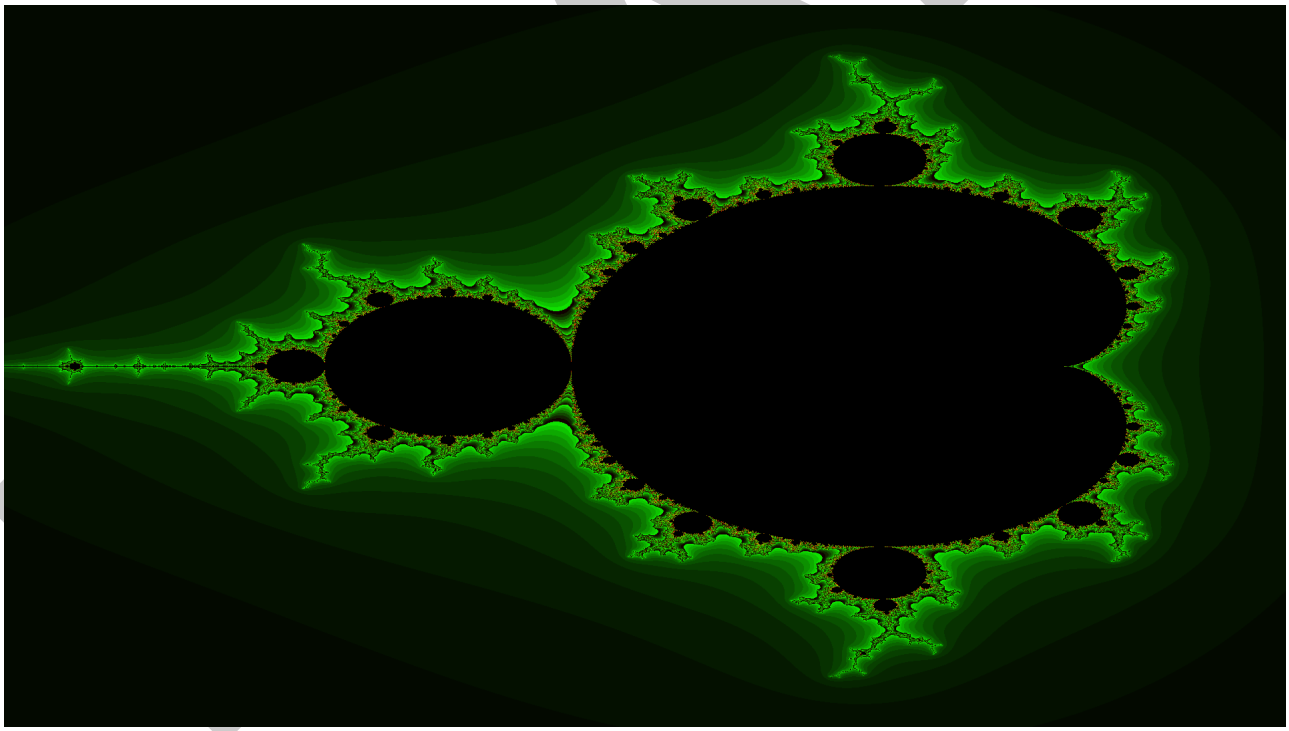
Helló, Mandelbrot!

5.1. A Mandelbrot halmaz

Írj olyan C programot, amely kiszámolja a Mandelbrot halmazt!

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

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



5.1. ábra. A Mandelbrot halmaz a komplex síkon

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

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

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

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

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

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

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

Megoldás forrása: BHAX repó, https://gitlab.com/nbatfai/bhax/-/blob/master/attention_raising/Mandelbrot/-3.1.2.cpp

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

```
// Verzio: 3.1.2.cpp
// Forditas:
// g++ 3.1.2.cpp -lpng -O3 -o 3.1.2
// Futtatas:
// ./3.1.2 mandel.png 1920 1080 2040  ↩
   -0.019473810573093663922605855987058021128l8  ↩
   -0.019473810572541341845642648422654019668 7  ↩
   0.798505756933826860155534177465597167611 1  ↩
   0.798505756934379196110285192844457924366
// ./3.1.2 mandel.png 1920 1080 1020  ↩
   0.412765541820958925534057470940751954913 1  ↩
   0.412765541824581805308014281763462349772 5  ↩
   0.213538705176874649138696327097512154281  ↩
   0.213538705180497528912653137922461610287 4
```

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


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

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

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

  if ( argc == 9 )
    {
      szelesseg = atoi ( argv[2] );
      magassag =  atoi ( argv[3] );
      iteraciosHatar =  atoi ( argv[4] );
      a = atof ( argv[5] );
      b = atof ( argv[6] );
      c = atof ( argv[7] );
      d = atof ( argv[8] );
    }
```

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

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

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

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

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

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

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

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

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

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

            ++iteracio;
          }

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

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

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

}
```

5.3. Biomorfok

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

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

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

A különbség a Mandelbrot halmaz és a Julia halmazok között az, hogy a komplex iterációban az előbbiben a c változó, utóbbiban pedig állandó. A következő Mandelbrot csipet azt mutatja, hogy a c befutja a vizsgált összes rácspontot.

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

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

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

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

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

              ++iteracio;
            }
```

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

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

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

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

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

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

```
//
// Version history
//
// https://youtu.be/IJMbgRzY76E
// See also https://www.emis.de/journals/TJNSA/includes/files/articles/ ↩
   Vol9_Iss5_2305--2315_Biomorphs_via_modified_iterations.pdf
//

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

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

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

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

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

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

    double dx = ( xmax - xmin ) / szelesseg;
```

```
    double dy = ( ymax - ymin ) / magassag;

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

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

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

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

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

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

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

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

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

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

}
```

5.4. A Mandelbrot halmaz CUDA megvalósítása

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

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

5.5. Mandelbrot nagyító és utazó C++ nyelven

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

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

Megoldás forrása: az ötödik előadás 26-33 fólia, illetve https://sourceforge.net/p/udprog/code/ci/master/-tree/source/binom/Batfai-Barki/frak/.

5.6. Mandelbrot nagyító és utazó Java nyelven

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

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

5.7. Vörös Pipacs Pokol/fel a láváig és vissza

Megoldás videó: https://youtu.be/I6n8acZoyoo

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

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: a második előadás 17-22 fólia.

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

6.2. LZW

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

Megoldás forrása: https://progpater.blog.hu/2011/03/05/labormeres_otthon_avagy_hogyan_dolgozok_fel_egy_pe

6.3. Fabejárás

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

Megoldás forrása:

6.4. Tag a gyökér

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

Megoldás videó: https://youtu.be/_mu54BDkqiQ

Megoldás forrása: ugyanott.

6.5. Mutató a gyökér

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

Megoldás videó: https://youtu.be/_mu54BDkqiQ

Megoldás forrása: ugyanott.

6.6. Mozgató szemantika

Írj az előző programhoz másoló/mozgató konstruktort és értékadást, a mozgató konstruktor legyen a mozgató értékadásra alapozva, a másoló értékadás pedig a másoló konstruktorra!

Megoldás videó: https://youtu.be/QBD3zh5OJ0Y

Megoldás forrása: ugyanott.

6.7. Vörös Pipacs Pokol/5x5x5 ObservationFromGrid

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7. fejezet

Helló, Conway!

7.1. Hangyaszimulációk

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

Megoldás videó: https://bhaxor.blog.hu/2018/10/10/myrmecologist

Megoldás forrása: https://gitlab.com/nbatfai/bhax/-/tree/master/attention_raising%2FMyrmecologist

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.2. Java életjáték

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

Megoldás forrása: https://regi.tankonyvtar.hu/hu/tartalom/tkt/javat-tanitok-javat/apb.html#conway

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.3. Qt C++ életjáték

Most Qt C++-ban!

Megoldás forrása: https://bhaxor.blog.hu/2018/09/09/ismerkedes_az_eletjatekkal

Megoldás videó: a hivatkozott blogba ágyazva.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.4. BrainB Benchmark

Megoldás videó: initial hack: https://www.twitch.tv/videos/139186614

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

7.5. Vörös Pipacs Pokol/19 RF

Megoldás videó: https://youtu.be/VP0kfvRYD1Y

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8. fejezet

Helló, Schwarzenegger!

8.1. Szoftmax Py MNIST

Python (lehet az SMNIST [SMNIST] is a példa).

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8.2. Mély MNIST

Python (MNIST vagy SMNIST, bármelyik, amely nem a softmaxos, például lehet egy CNN-es).

Megoldás videó: https://bhaxor.blog.hu/2019/03/10/the_semantic_mnist

Megoldás forrása: SMNIST, https://gitlab.com/nbatfai/smnist

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8.3. Minecraft-MALMÖ

Most, hogy már van némi ágensprogramozási gyakorlatod, adj egy rövid általános áttekintést a MALMÖ projektről!

Megoldás videó: initial hack: https://youtu.be/bAPSu3Rndi8. Red Flower Hell: https://github.com/nbatfai/-RedFlowerHell.

Megoldás forrása: a Red Flower Hell repójában.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

8.4. Vörös Pipacs Pokol/javíts a 19 RF-en

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

9. fejezet

Helló, Chaitin!

9.1. Iteratív és rekurzív faktoriális Lisp-ben

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

Megoldás forrása: ugyanott.

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

9.2. Gimp Scheme Script-fu: króm effekt

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

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

9.3. Gimp Scheme Script-fu: név mandala

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

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

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

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

9.4. Vörös Pipacs Pokol/javíts tovább a javított 19 RF-eden

Megoldás forrása: https://github.com/nbatfai/RedFlowerHell

Tanulságok, tapasztalatok, magyarázat... ezt kell az olvasónak kidolgoznia, mint labor- vagy otthoni mérési feladatot! Ha mi már megtettük, akkor használd azt, dolgozd fel, javítsd, adj hozzá értéket!

10. fejezet

Helló, Gutenberg!

10.1. Programozási alapfogalmak

Rövid olvasónapló a [PICI] könyvről.

10.2. C programozás bevezetés

Rövid olvasónapló a [KERNIGHANRITCHIE] könyvről.

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

10.3. C++ programozás

Rövid olvasónapló a [BMECPP] könyvről.

10.4. Python nyelvi bevezetés

Rövid olvasónapló a [BMEPY] könyvről.

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, Berners-Lee!

11.1. Java vs C++

A Java nyelv nagyon hasonlít a C és a C++ nyelvekhez, hiszen szintaxisa ezen nyelvekbol származtatható. Egy alapveto különbség a Java és a C++ nyelvek között már a fordítás során előjön. Míg a C++ egy natív nyelv (azaz fordítás sorás egy az egybe gépi kódot állít elő), addig a Java egy interpreteres nyelv, azaz a fordítás során egy bájtkódot (Javabytecode) állít elő. Ez a Javabytecode a Java Virtual Machine számára értelmezheto kód. Bármilyen rendszeren amin rajta van a JVM, az előbb említett bájtkód értelmezhető lesz. Egyszerre előonye és hátránya is ez a Java-nak. Mivel a bájtkód előállítása majd annak tovább fordítása időigényes, ezért ez nagy hátrány is lehet. Viszont ha azt nézzük, hogy a JVM által egy adott Java kódot bármilyen rendszeren futtathatjuk nagy előnyre teszünk szert.

Javaban szintén jelen vannak az operátorok, pl az = jel, amelynek egy felhasználása pl. az értékadás egy változónak. A konstansok, vagyis állandók, illetve a megjegyzések jelölése szintén azonos a nyelvekben. C-ben nincs, de Javaban és C++-ban van osztálykezelés. Javaban a karakterláncok, vagyis a nem csak egy karakterből álló tömb kezelésére új osztályt hoznak létre (string). A new operátorral lehet objektumot létrehozni, ugyanis ez az operátor foglalja le a szükséges méretű helyet, majd inicializálja azt, és innentől kezdve referenciaként használja, azzal tér ide vissza. Egyszerű típusok esetében lehet a new operátor nélkül is inicializálni. Azok az elemek, amelyeket static-kal deklaráltunk, azok az adott osztályhoz tartoznak, nem az objektumhoz. Javában nincs eszközünk ahhoz, hogy megszüntessünk egy objektumot

Amikor Javaban egy metódust szeretnénk alkalmazni, akkor a szerkezetet egy class szóval kell bevezetnünk. A metódusok létrehozásánál is vannak bizonyos szabályok, amelyeket muszáj betartanunk. A kivételkezelés szintén fontos szerepet tölt be a nyelvekben, ugyanis ezek a megbízhatóság növelésére szolgálnak. Például ekkor érdemes használni a try-catch blokkot. A nyelvek karakterkészlete szintén sok hasonlóságot mutat, ugyanis az ASCII táblázatot használják, illetve a Unicode karaktereket. Javaban a típusok között lényeges különbség van, ugyanis nem mindegy a használat szempontjából, hogy primitív típusú a változó (pl int változó) vagy sem (pl az Intiger típusú objektumhivatkozást tartalmaz). Literálokat kell használnunk, ha egyszerű típusokat vagy objektumokat szeretnénk inicializálni, erre több féle literál szolgál. Amikor változót deklarálunk mindenképp nevet kell adni annak, ez igaz a C++-ra és a Javara is. Értéket adni neki nem muszáj azonnal. Javaban a tömb egy típust jelöl, ebben különbözik a C++-tól, ugyanis nem csak a mutató jelölése másképp, de több dimenziós tömb viszont nem létezik, ha azt szeretnénk, akkor a tömbbe újabb tömböt kell tenni.

Léteznek felsorolási típusok, amiket a tömbökhöz tudnék hasonlítani működés szempontjából (indexelés), sorszám szerint tudunk értéket adni ezeknek. Az operátorok szerepe az, hogy megadja a kiértékelés sor-

rendjét. A Java nyelv erősen típusos, ezért van szükség típuskonverzióra, amely során megvizsgálja, hogy léteznek e összegyeztethető típusok, vagy konverzióval azonos típusra lehet e alakítani a különböző elemeket. Javaban egy struktúra egyik részelemének/tagjának eléréséhez a "." karaktert használhatjuk az elemek közötti elhelyezéssel. Itt nincs megkülönböztetve az osztályok elérése mint C++-ban, ott erre a "::" operátor szolgál.

Az utasításokon belül 2 fajta utasítást tudunk megkülönböztetni. Ezek a kifejezés- és deklaráció-utasítás. Mindkettőt azonosan ; zárja, előbbi értékadásra, postfix vagy prefix képezésre, metódushívásra vagy példányosításra szolgál, még utóbbi egy lokális változó létrehozására/inicializálására. Ha Javaban elágazási szerkezetet szeretnénk készíteni, azt kétféleképpen tehetjük meg, az egyik az egyszerű elégazás, amelyhez egy if-szerkezet szükségez, még a másikhoz, az öszetett elágazáshoz switch-case-szerkezet szükséges. A Java összesen 4 féle ciklust ismer, amelyekben a vizsgálati helyek változnak. Ezek az elöl-, hátultesztelő ciklusok, a léptető (for/while) és bejáró (for) ciklus. Léteznek utasítások, amelyekkel a program vezérelhető, pl bizonyos körülmények között megállítható, folytatható, vagy az adott ponton átugrik egy másik pontra (utasításra). Ezek a: címkék, break és continue utasítások, a visszatérés és a goto.

A Java alapja az osztálykezelés, a legkisebb önálló egység benne az osztály. Az osztályok egyfajta modellezési szerepet töltenek be, egy rendszert épít fel. Példányok, objektumok és egyedek szerepelnek bennük, bár ezek a kifejezések ugyan azt jelentik, de lényeges tudnunk, hogy melyik osztályban vannak és ezt jelezni valahogyan. Az osztályoknak saját változó készlete van, így akár 2 különböző osztályban is használhatunk azonos változónevet pl. Az osztály egyik szerepe az egységbe zárás, tehát az adatok és műveletek összefogása. Biztonsági funkciója is van, pl adatrejtés. Ha osztály változójára hivatkozunk, akkor a változó neve előtti ponttal elválasztva megadhatjuk, hogy melyik más objektum változójára hivatkozunk. Ennek akár a metódusok működésében is lehet fontos szerepe. Ezeket a metódusokat metódusdefiníciók írják le, valamint fej és törzs részből áll. A fej előtt a módosítók állnak, a törzs pedig az utasításblokk. Ebben azonos a Java és a C++, hogy a törzs és a fej nincs külön választva. Ha metódushívást alkalmazunk ügyelnünk kell a paraméterlistában megadott dolgokra, ugyanis ezek kötött tényezők a használat során (típusoknak egyezni kell, és a paraméterszámnak). Ha példányra szeretnénk alkalmazni, akkor a metódusnévvel ellátott objektummal kell hívnunk.

A metódus zárójelezésére szintén ügyelni kell, ugyanis ha nincs paraméterlista is szükséges a () zárójelezés. Egy metódusnevet többször is használhatunk, csak a paraméterlistájuk ne egyezzen meg (paraméterek száma/típusa), ezt nevezzük metódusnevek túlterhelésének. A helyes használatot utána a Java fordító tudni fogja, a jót választja ki még a lefordítás idejében. Az objektumok a példányosítással hozhatók létre a new operátor használatával. A folyamat közben a new operátor memóriát foglal le, ahol az objektum változóit elhelyezi, és a memória kezdőcímét adja vissza. Az ezzel kapott referenciát a megfelelő osztály típusú változónak adható csak át. Ha már nincs szükségünk egy objektumra, akkor érdemes törölni azt. Itt különbség van a C++ és Java között, ugyanis mint sok nyelvben a C++-ban a programozónak kell törölni azt (így könnyen ütközhet hibába a használat miatt), még a Javaban viszont a program futás közben vizsgálja, és ha már nincs rá szükség akkor törli magától, ez a szemétgyűjtő mechanizmus. Az osztályoknál figyelnünk kell a hozzáférhetőségekre. Erre szolgál pl a private, public vagy protected hozzáférési kategória, tehát a láthatóságra. Ez alapján több csoportra bonthatjuk szét: félnyilvános, nyilvános, privát vagy leszármazottban hozzáférhető tagok.

Az osztály szintű tagokat példányváltozóknak nevezzük. Az osztályváltozók és metódusok más néven statikus tagok elé a static módosítót helyezzük. Ezek az osztályokhoz kapcsolódnak. Az osztályváltozók és a példányváltozók inicializálása is az előfordulási sorrend szerint történik, de az előbbi kezdőértéket csak egyszer kap, még utóbbi minden meghíváskor inicializálódok. Ugyan úgy hivatkozunk rájuk. Az osztálymetódus az az osztályon belüli műveletet takarja, ehhez csak az osztályváltozók használhatóak fel. Az osztályokon belül konstruktorokat es destruktorokat használunk. A destruktorok feladata a törlés, amiről

nemrég írtam, hogy C++-ban a programozók (röviden és tömören) ennek a segítségével szüntethetnek meg meg. A konstruktor lényegében példányosítás, de a hibalehetőségek száma kisebb. A konstruktordefiníció leginkább a metódusdefinícióhoz hasonlítható. Ha példányosítani szeretnénk, a new operátor számára paraméterek is megadhatóak, ezek a paraméterek azok amelyeket a konstruktornak szeretnénk adni. Öröklődést pedig úgy tudnám jellemezni, hogy egy osztályt egy másik osztállyal bővítünk. Ezután a szülő osztályon keresztül elérhető a gyermek osztály, valamint a gyermek osztály örökli a szülő tulajdonságait a létrehozásakor. A gyermek osztály lényegében a szülőnek a kibővítése. Viszont a szülő megadhatja, hogy egy metódusa például public vagy private. Ami private, azt a gyermek már nem látja (ha public vagy protected, akkor látja a szülő metódusait).

Ez igaz a Javára és C++-ra is. Az osztályhierarchiáról, metódustúlterhelésről és hatáskörről korábban már írtam úgy, hogy az mindkét nyelvre igaz legyen, így áttérnék a többire. A polimorfizmusban létezik egy szülő osztály, és azon belül lehet bármennyi gyermek osztály, amelyek öröklik a szülő tulajdonságait. Ha a szülő osztályt példányosítjuk, akkor utána polimorfizmussal a szülőn keresztül elérhetőek a gyermekek. Például a példányosított szülő osztályban létrehozunk egy objektumot, és a szülőn belül vannak (extend), leszármaztatott gyermek osztályok. Ezeket a gyermekeket pedig akár öbbet is bele tölthetünk az említett adott objektumba. Emiatt nevezzük a polimorfizmust más szóval többalakúságnak, vagyis inkább ha lefordítjuk a szót. A polimorfizmus során a gyermekek kaphatnak még azonos metódusokat, de ezeket külön-külön kezelik, hiába azonos a metódus (mintha a két gyereknek ugyanaz a jellemzője lenne más tulajdonságú).

Ha absztakt osztályokról és az interfészekről beszélünk, akkor azonnal eszünkbe juthat, hogy vannak absztakt függvények és metódusok is. Ha ezeket szeretnénk használni, akkor a definiálásuk során használhatjuk az abstract vagy public kulcsszót, de ezek nem kötelezőek. Javaban az interfész nem tartozik már az osztályokhoz. Ez lényegében az előbb említett absztakt metódusokat és konstansokat használja. Az interfész lényegében egy felület, és a bennetalálható metódusok csak deklarálva vannak, kifejtve nincsenek. A interfészeken belül is létezik örödlődés, a szülő-gyermek kapcsolat. Az absztrakt és az osztályok metódusai között annyi különbség van a deklarációjukban, hogy az absztraktnak nincs törzse. C++-ban az absztakt osztály pedig tulajdonképpen az interfészben megjelenő műveletek és objektumok felsorolása. Ahogy olvastam a 2 könyvet én úgy vettem észre, hogy a lényege és a főbb összetevők (pl absztakt metódusok) azonosak, a megvalósításukban látok különbséget.

Java és C++ kód különbségek

Main function:

```
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println( "Hello, World" );
    }
}
```

```
int main( int argc, char* argv[])
{
    printf( "Hello, world" );
}
```

Osztály deklaráció:

```
class Bar {}
```

```
class Bar {};
```

Objektum deklaráció:

```
// always allocated on the heap (also, always need parens for constructor)
myClass x = new myClass();
```

```
// on the stack
myClass x;

// or on the heap
myClass *x = new myClass;
```

Referencia vs mutató:

```
// references are mutable and store addresses only to objects; there are
// no raw pointers
myClass x;
x.foo(); // error, x is a null ``pointer''

// note that you always use . to access a fiel
```

```
// references are immutable, use pointers for more flexibility
int bar = 7, qux = 6;
int& foo = bar;
```

Öröklődés:

```
class Foo extends Bar
    { ... }
```

```
class Foo : public Bar
    { ... };
```

Logikai változó:

```
boolean foo;
```

```
bool foo;
```

Hiba dobás:

```
int foo() throws IOException
```

```
int foo() throw (IOException)
```

Tömbök:

```
int[] x = new int[10];
// use x, memory reclaimed by the garbage collector or returned to the
// system at the end of the program's lifetime
```

```
int x[10];
// or
int *x = new x[10];
// use x, then reclaim memory
delete[] x;
```

11.2. Bevezetés a mobilprogramozásba

A Python magasszintű, általános célú nyelv, szkriptnyelvként szokták emlegetni. A kódokat egy futtatókörnyezeten keresztül futtatják általában, viszont vannak már kísérleti stádiumban natív kódot generáló fordítóprogramok is. Mind a procedurális, és az objektumorientált programozást támogatja Könyvátárnak mérete a nyelv egyik erősségének mondható, melyben még http támogatás is megtalálható. C, cpp nyelven készült modulok is egyszerűen adhatók hozzá a környezethez. Népszerűségét az egyszerűségnek köszönheti, szinte bármilyen feladatot meg lehet oldani a nyelvben, viszont elsősorban kliensszoftverek készítésére alkalmazzák. A mobileszközökön is futtatható, írható python kód, viszont ez nem újdonság, hiszen java ill. C++ kódokat is futtathatunk manapság.

A Symbian OS mobiltelefon operációs rendszer egyike a napjainkban legtöbbet alkalmazott operációs rendszereknek. Mind C++-ban, Python-ban és Java-ban is írhatónk rá kódokat, viszont a rendszer felett még ott van egy GUI(Grafikus felhasználói felület), ezekből manapság 2 típus ismert: S60(korábban Series60) és az UIQ. Hasonló funkcionalitás lelhető fel mindkét rendszerben, viszont ennek ellenére a kettő nem kompatibilitis egymással, az egyikre írt programok nem fognak a másikon elfutni. Az operációs rendszer alapfunkcióiban azonban megegyezik, tehát csak az UI-hez kötődő részeket kell külön megírnunk kétféleképpen.

A Windows Mobile pedig a Microsoft mobiltelefonokhoz fejlesztett operációsrendszer,amely a Windows CE rendszeren alapul. A Symbian-hoz képest több lehetőségünk van a programnyelvek használatát illetően. A C++, Python és Java-n kívül natív alkalmazásokat készíthetünk C nyelven is, és persze az ugyancsak Microsoft által fejlesztett .NET compact framework technológiával egyaránt. Ez utóbbi a Microsoft .NET telefonokra kifejlesztett változata.

A Harmadik operációsrendszer mobiltelefonokra a nyílt forráskódú, Linux-ra épülő Maemo, ez a rendszer a Nokia fejlesztése alatt áll. A rendszer nagyrészben az internetes elérésekre,megoldásokra fókuszál, viszont az alapvető funkciókat ezen a rendszeren is el tudjuk végezni. Jelenleg a Bluetooth és WLAN funkciókat támogatja, de már a Wimax támogatását is híresztelik.

12. fejezet

Helló, Arroway!

12.1. Kódolás from scratch

```
/*
 * PiBBP.java
 *
 * DIGIT 2005, Javat tanítok
 * Bátfai Norbert, nbatfai@inf.unideb.hu
 *
 */
/**
 * A BBP (Bailey-Borwein-Plouffe) algoritmust a Pi hexa
 * jegyeinek számolását végző osztály. A könnyebb olvahatóság
 * kedvéért a változó és metódus neveket megpróbáltuk az algoritmust
 * bemutató [BBP ALGORITMUS] David H. Bailey: The BBP Algorithm for Pi.
 * cikk jelöléseihez.
 *
 * @author Bátfai Norbert, nbatfai@inf.unideb.hu
 * @version 0.0.1
 */
public class PiBBP {
    /** A Pi hexa kifejtésében a d+1. hexa jegytől néhány hexa jegy.*/
    String d16PiHexaJegyek;
    /**
     * Létrehoz egy <code>PiBBP</code>, a BBP algoritmust a Pi-hez
     * alkalmazó objektumot. A [BBP ALGORITMUS] David H. Bailey: The
     * BBP Algorithm for Pi. alapján a
     * {16^d Pi} = {4*{16^d S1} - 2*{16^d S4} - {16^d S5} - {16^d S6}}
     * kiszámítása, a {} a törtrészt jelöli.
     *
     * @param   d   a Pi hexa kifejtésében a d+1. hexa jegytől
     * számoljuk a hexa jegyeket
     */
    public PiBBP(int d) {

        double d16Pi = 0.0d;
```

```
        double d16S1t = d16Sj(d, 1);
        double d16S4t = d16Sj(d, 4);
        double d16S5t = d16Sj(d, 5);
        double d16S6t = d16Sj(d, 6);

        d16Pi = 4.0d*d16S1t - 2.0d*d16S4t - d16S5t - d16S6t;

        d16Pi = d16Pi - StrictMath.floor(d16Pi);

        StringBuffer sb = new StringBuffer();

        Character hexaJegyek[] = {'A', 'B', 'C', 'D', 'E', 'F'};

        while(d16Pi != 0.0d) {

            int jegy = (int)StrictMath.floor(16.0d*d16Pi);

            if(jegy<10)
                sb.append(jegy);
            else
                sb.append(hexaJegyek[jegy-10]);

            d16Pi = (16.0d*d16Pi) - StrictMath.floor(16.0d*d16Pi);
        }

        d16PiHexaJegyek = sb.toString();
    }
    /**
     * BBP algoritmus a Pi-hez, a [BBP ALGORITMUS] David H. Bailey: The
     * BBP Algorithm for Pi. alapján a {16^d Sj} részlet kiszámítása.
     *
     * @param   d   a d+1. hexa jegytől számoljuk a hexa jegyeket
     * @param   j   Sj indexe
     */
    public double d16Sj(int d, int j) {

        double d16Sj = 0.0d;

        for(int k=0; k<=d; ++k)
            d16Sj += (double)n16modk(d-k, 8*k + j) / (double)(8*k + j);

        /* (bekapcsolva a sorozat elejen az első utáni jegyekben növeli pl.
            a pontosságot.)
        for(int k=d+1; k<=2*d; ++k)
            d16Sj += StrictMath.pow(16.0d, d-k) / (double)(8*k + j);
         */

        return d16Sj - StrictMath.floor(d16Sj);
    }
```

```
    /**
     * Bináris hatványozás mod k, a 16^n mod k kiszámítása.
     *
     * @param   n   kitevő
     * @param   k   modulus
     */
    public long n16modk(int n, int k) {

        int t = 1;
        while(t <= n)
            t *= 2;

        long r = 1;

        while(true) {

            if(n >= t) {
                r = (16*r) % k;
                n = n - t;
            }

            t = t/2;

            if(t < 1)
                break;

            r = (r*r) % k;

        }

        return r;
    }
    /**
     * A kiszámolt néhány hexa jegy visszaadása. A használt lebegőpontos
     * aritmentia függvényében mondjuk az első 6 pontos peldául
     * d=1000000 esetén.
     *
     * @return String a kiszámolt néhány hexa jegy
     */
    public String toString() {

        return d16PiHexaJegyek;
    }
    /** Példányosít egy BBP algoritmust implementáló obektumot.*/
    public static void main(String args[]) {
        for(int i=0; i<3000; i+=1) {
            PiBBP piBBP = new PiBBP(i);
            System.out.print(piBBP.toString().charAt(0));
        }
    }
```

```
}
```

12.2. Yoda

Feladat: Írjunk olyan Java programot, ami java.lang.NullPointerEx-el leáll, ha nem követjük a Yoda conditions-t! https://en.wikipedia.org/wiki/Yoda_conditions

```
public class Yoda{

     public static void main(String []args){
        String test = null;
        if (test.equals("")) {

        }
     }
}
```

Hibát dob vissza, mert a .equals metódus minden objektumra használható, de a NULL nem egy objektum, ezért a .equals-t nem használhatjuk a test nevű String-re.

12.3. EPAM: Java Object metódusok

1. toString(): toString Objektumot alakít át String-é. A toString() alap viselkedése hogy kiírja az osztály nevét, egy @-ot, majd az object hexadecimális hash códját. Mindig ajánlott felülírni a toString() metódust, hogy a saját magunk String formáját kapjuk meg az Object-nek.

2. hashCode(): A JVM minden objektumhoz generál egy unique számot ami a hashcode. Különböző inteket küld vissza különböző objektekhez. Az objektumok hash code alapjáni mentésének nagy előnye hogy megkönnyíti a keresést. A hashCode() metódus felülírásának meg kell történnie mivel minden objektumnak egyedi számot generálunk.

3. equals(Object obj): Két objektet összehasonlít. Ajánlott felülírni a equals(Object obj) metódust, hogy a kívánt összehasonlítási kondíció alapján hasonlítsuk össze.

4. getClass(): Lekéri a class metaadatait.

```
public class Test
{
    public static void main(String[] args)
    {
        Object obj = new String("test");
        Class c = obj.getClass();
        System.out.println("Class of Object obj is : "
                           + c.getName());
    }
}
```

Output: Class of Object obj is : java.lang.String

5. finalize(): Ez a metódus meghívásra kerül mielőtt egy objektum a garbage collectorba kerül. A Garbage Collercor hívja meg egy objektumon amikor a garbage collector eldönti hogy már nincs több utalás az objektumra.

6. clone(): Visszaküld egy objektumot, amir a másolata a meghívott objektumnak.

7. A maradék 3 metódus a wait(), notify(), notifyAll() az egyidejűséghez kapcsolódik.

12.4. EPAM: Eljárásorientál vs Objektumorientált

Objektum-orientált programozás (OOP) meghatározása

Az OOP-t az „objektum", „osztályok", „adatkapsulálás vagy absztrakció", „öröklés" és „polimorfizmus / túlterhelés" alapfogalma alapján fejlesztették ki. Az OOP-ban a programokat kisebb részekre lehet felosztani az adatok és a funkciók elkülönítése útján, amelyek további sablonokként felhasználhatók modulok új példányainak létrehozásához. Ezért ez a megközelítés megkönnyíti a programok modulálását azáltal, hogy particionált memóriaterületet épít fel az adatok és a funkciók számára.

Az eljárásorientált programozás (POP) meghatározása

A POP a programozás szokásos módja. Az eljárási programozás során az elsődleges hangsúly a feladat sorrendben történő elvégzésére irányul. A folyamatábra megszervezi a program irányításának folyamatát. Ha a program kiterjedt, akkor néhány kisebb egységben felépíti, úgynevezett függvényekben, amelyek megosztják a globális adatokat.

POP előnyei: Lehetővé teszi, hogy ugyanazt a kódot több helyen is újra felhasználhassuk. Megkönnyíti a programáramlás követését. Képes modulok építésére.

POP hátrányai: A globális adatok sebezhetők. Az adatok a programon belül szabadon mozoghatnak. Nehéz az adatok helyzetét ellenőrizni. A funkciók cselekvésorientáltak. A funkciók nem képesek kapcsolódni a probléma elemeihez. A valós problémák nem modellezhetők. A kód részei egymástól függenek. Az egyik alkalmazáskód nem használható másik alkalmazásban. Az adatok továbbítása a függvények használatával történik.

OOP előnyei: Az objektumok segítenek a feladat particionálásában a projektben. A biztonságos programokat adatrejtés segítségével lehet felépíteni. Potenciálisan leképezheti az objektumokat. Lehetővé teszi az objektumok különböző osztályokba sorolását. Az objektum-orientált rendszerek könnyedén frissíthetők. A redundáns kódok öröklés útján kiküszöbölhetők. A kódok az újrafelhasználhatósággal kibővíthetők. Nagyobb modulitás érhető el. Az adatok absztrakciója növeli a megbízhatóságot. Rugalmas a dinamikus kötési koncepció miatt. Az információ elrejtésével elválasztja az alapvető specifikációt a megvalósításától.

OOP hátrányai: Több erőforrást igényel. Az objektumok dinamikus viselkedése RAM tárolást igényel. Az észlelés és a hibakeresés nehezebb az összetett alkalmazásokban, amikor az átadást végrehajtják. Az öröklés osztályaikat szorosan összekapcsolja, ami befolyásolja a tárgyak újrahasznosíthatóságát

12.5. EPAM: Objektum példányosítás programozási mintákkal

Abstract Factory: Esetet csinál több osztály családra.

Builder: Elküldöníti a objektum szerkezetét a megjelenítésétől.

Factory Metho: Példát csinál több származtatott osztályra.

Object Pool: A nem használt objektumokat újrahasznosítja.

Prototype: Egy egyedet lemásol vagy clónoz.

Singleton: Egy osztály ahol csak egy eset létezhet.

13. fejezet

Hello, Liskov!

13.1. EPAM: Interfész evolúció Java-ban

Java 8 előtt az interfészekben csak absztrakt metódusok lehettek. Minden interface metódusa public és absztrakt volt alapjáratból. Java 8 engedi hogy az interfészeknek legyen default és statikus metódusa. Azért jó hogy tudunk adni default metódusokat az interfészbe, mivel így tudunk adni új metódusokat az interfészbe anélkül hogy azok befolyásolnák a classokat amik implementálják az interfészeket.

Java 8 példa: Default metódus az interfészben:

```
interface MyInterface {

    //Ez egy default metódus, ezért nem kell implementálni az osztályokban.
    default void newMethod() {
        System.out.println("Newly added default method");
    }

    //Ez egy public és absztrakt metódus, ezért implementálnunk kell az ↩
       osztályokban.
    void existingMethod(String str);

}

public class Example implements MyInterface {

  //absztrakt metódus implementálása
    public void existingMethod(String str) {
        System.out.println("String is: " + str);
    }

    public static void main(String[] args) {
      Example obj = new Example();

      //a default metódus hívása
        obj.newMethod();
```

```
        //az absztrakt metódus hívása
        obj.existingMethod("This is an abstract method");
    }
}
```

Output: Newly added default method String is: This is an abstract method

Java 8 példa: Default metódus az interfészben:

```
interface MyInterface {

    //Ez egy default metódus, ezért nem kell implementálni az osztályokban.
    default void newMethod() {
        System.out.println("Newly added default method");
    }

    //Ez egy static metódus. A static metódusok hasonlóak a defaulthoz,  ↩
       viszont nem tudjuk felülírni őket az osztályokban.
    static void anotherNewMethod() {
      System.out.println("Newly added static method");
    }

    //Ez egy public és absztrakt metódus, ezért implementálnunk kell az  ↩
       osztályokban.
    void existingMethod(String str);
}
public class Example implements MyInterface {

  //absztrakt metódus implementálása
    public void existingMethod(String str) {
        System.out.println("String is: "+str);
    }

    public static void main(String[] args) {

      Example obj = new Example();

      //a default metódus hívása
        obj.newMethod();

        //a static metódus hívása
        MyInterface.anotherNewMethod();

        //az absztrakt metódus hívása
        obj.existingMethod("This is an abstract method");
    }
}
```

Output: Newly added default method Newly added static method String is: This is an abstract method

13.2. EPAM: Liskov féle helyettesíthetőség, öröklődés

```
class Vehicle {

    public Vehicle() {
        System.out.println("Vehicle constructor");
    }

    public void start() {

    }
}

class Car extends Vehicle {

    public Car() {
        System.out.println("Car constructor");
    }

    @Override
    public void start() {

    }
}

class Supercar extends Car {

    public Supercar() {
        System.out.println("Supercar constructor");
    }

    @Override
    public void start() {

    }
}

public class Liskov{

     public static void main(String []args){

          //Készítünk egy új Supercar példányt amit Vehicle-ben rárolunk.
          //Az 3 construktor meghívódik.
        Vehicle firstVehicle = new Supercar();

        //A supercar osztályban definiált start lefut, mivel minden ↩
           alosztályban felülírtuk a starot.
        firstVehicle.start();
```

```
        //Vizsgáljuk hogy az objektum példánya a Car osztálynak.
        // Visszaad true-t, szóvla igen.
        System.out.println(firstVehicle instanceof Car);

        //Hasonlóan az előző 3 sorhoz, csak most a firstVehicle objektumot ↩
           a Car-ra castolva is true-t ír ki.
        Car secondVehicle = (Car) firstVehicle;
        secondVehicle.start();
        System.out.println(secondVehicle instanceof Supercar);

        //Itt hibát fog kiadni, mivel a Vehicle nem konvertálható Supercar- ↩
           rá.
        //Supercar thirdVehicle = new Vehicle();
        //thirdVehicle.start();
    }
}
```

13.3. Interfész, Osztály, Absztrak Osztály

Interfész: Az interfész egy olyan absztract osztály ami összevon egymáshoz kapcsolódó metódusokat üres body-val.

```
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Output: The pig says: wee wee Zzz

Absztrakt osztály: Az absztrakt osztály olyan osztály amit abstract kulcsal van deklarálva. Ilyen osztályból nem készíthető objektum a new kulcsal. Az absztrakt class elérhető egy konkrét subclassal, vagy meghatározva az összes absztrakt metódussal.

```
abstract class DemoAbstractClass {
    abstract void display();
}

public class JavaApplication {
    public static void main(String[] args)
    {
        DemoAbstractClass AC = new DemoAbstractClass() {
            void display()
            {
                System.out.println("Hi.");
            }
        };
        AC.display();
        System.out.println("How are you?");
    }
}
```

Output: Hi. How are you?

Konkrét osztály: A konkrét class egy olyan subclass, ami implementál minden absztrakt metódusát az abstrakt classnak.

```
abstract class DemoAbstractClass {
    abstract void display();
}

class ConcreteClass extends DemoAbstractClass {
    void display()
    {
        System.out.println("Hi.");
    }
}

public class JavaApplication {
    public static void main(String[] args)
    {
        ConcreteClass C = new ConcreteClass();
        C.display();
        System.out.println("How are you?");
    }
}
```

Output: Hi. How are you?

13.4. Szülő-gyerek

Olyan java és c++ osztálydefiníciót kell írni, ami mutatja hogy az ősün keresztül csak az ős üzenetei külhetőek.

```
public class Parent {

    public void run() {
        System.out.println("Parent method");
    }
}

class Child extends Parent {

    public void runChild() {
        System.out.println("Child mehod");
    }
}

public class ParentChild {
     public static void main(String []args){

        Parent parentOne = new Parent();
        Parent parentTwo = new Child();

        parentOne.run();
        parentTwo.runChild();
     }
}
```

Error: location: variable parentTwo of type Parent.

```
#include <iostream>

using namespace std;

class Parent {
    public:
    void run() {
        std::cout << "Parent method" << endl;
    }
};

class Child : public Parent {
    public:
    void runChild() {
        std::cout << "Child method" << endl;
    }
};
```

```
int main()
{
    Parent* parentOne = new Parent();
    Parent* parentTwo = new Child();

    parentOne -> run();
    parentTwo -> runChild();

    return 0;
}
```

Error: main.cpp:33:18: error: class Parent' has no member named 'runChild'

IV. rész

Irodalomjegyzék

13.5. Általános

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

[PICI] Juhász, István, *Magas szintű programozási nyelvek I.*

[SMNIST] Norbert Bátfai, Dávid Papp, Gergő Bogacsovics, Máté Szabó, Viktor Szilárd Simkó, Márió Bersenszki, Gergely Szabó, Lajos Kovács, Ferencz Kovács, and Erik Szilveszter Varga, *Object file system software experiments about the notion of number in humans and machines*, Cognition, Brain, Behavior. An Interdisciplinary Journal , DOI 10.24193/cbb.2019.23.15 , 2019.

13.6. C

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

13.7. C++

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

13.8. Python

[BMEPY] Ekler, Péter, Forstner, Bertalan, Ã©s Kelényi, Imre, *Bevezetés a mobilprogramozásba - Gyors prototípusfejlesztés Python és Java nyelven*, Bp., Szak Kiadó, 2008.

13.9. 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.

A tananyag elkészítését az EFOP-3.4.3-16-2016-00021 számú projekt támogatta. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.